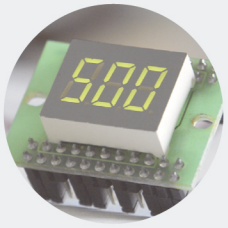
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Reconhecimento de Movimentos Usando Técnicas de Deep Learning

DANIEL ANDRÉ PINTO DA COSTA GONÇALVES
Junho de 2024

POLITÉCNICO
DO PORTO

Politécnico do Porto
Instituto Superior de Engenharia do Porto

Reconhecimento de Movimentos Usando Técnicas de *Deep Learning*

Daniel André Pinto da Costa Gonçalves

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização em Automação e Sistemas

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

Junho, 2024

Esta dissertação satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores, Área de Especialização em Automação e Sistemas.

Candidato: Daniel André Pinto da Costa Gonçalves, Nº 1180812,
1180812@isep.ipp.pt

Orientação Científica: Ramiro de Sousa Barbosa, rsb@isep.ipp.pt

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

Junho, 2024

"Se eu vi mais longe, foi por estar sobre ombros de gigantes."
Isaac Newton

Agradecimentos

Aproveito esta secção para agradecer a todas as pessoas que, de uma forma mais ou menos direta, contribuíram para o meu sucesso académico e, sobretudo, pelo apoio durante a elaboração deste trabalho científico.

Começando pelo Engº Ramiro Barbosa, que além de me ter orientado ao longo da dissertação, colaborou sempre com vista a que este projeto avançasse com todas as condições requeridas. Além disso, quero agradecer a sua dedicação, até mesmo quando o projeto teve um desenvolvimento mais lento.

Aos meus pais, que me apoiaram na continuação o meu percurso académico e desde sempre me motivaram para querer ser cada vez melhor.

À minha namorada, que além do apoio que me presta constantemente, torna-me um ser humano mais dedicado e, acima de tudo, encoraja-me a seguir sempre o caminho que me faz mais feliz.

Aos meus amigos e às conversas que partilhamos sobre tudo o que nos rodeia.

À minha família que sempre preza pelo meu sucesso e felicidade na vida.

À Armis Group, que me acolheu enquanto estagiário curricular no final da licenciatura e, até hoje, me mantém como colaborador, prestando o auxílio necessário, tanto a nível académico como pessoal.

Um muito obrigado a todos.

Resumo

Este documento descreve a utilização de técnicas de *Deep Learning* para a criação de um sistema de reconhecimento de movimentos efetuados pelo ser humano, tais como andar, subir e descer escadas, entre outros. Esses movimentos são captados por sensores presentes num *smartphone* comum, tais como giroscópios e acelerómetros. Esta aplicação torna-se interessante, pois poder-se-á tornar num produto de monitorização de movimentos para pessoas com dificuldades de mobilidade ou idosos com essas e outras dificuldades.

O principal objetivo deste projeto é o de utilizar metodologias de aprendizagem máquina para a resolução do problema, recorrendo a diferentes tipos de redes neuronais que, aproveitando as capacidades de aprendizagem, levarão a uma posterior análise do seu desempenho. Deste modo, o trabalho apresentado terá o propósito de avaliar qual ou quais as redes que melhor desempenho demonstraram no reconhecimento dos movimentos.

Palavras-Chave: Redes Neuronais, LSTM, CNN, FNN, *Machine Learning*, *Deep Learning*, Inteligência Artificial.

Abstract

The present document aims to describe the use of Deep Learning techniques to create a human motion recognition system, that can recognize walking, climbing up or down stairs, and so on. The motions are captured by smartphone sensors, like accelerometers and gyroscopes. This approach is interesting, since it can become a product to monitor the motion of people with mobility problems and old people that have those problems and more.

The main objective of this project is to use Machine Learning methodologies to solve this problem, using different types of Neural Networks that, taking advantage of learning capabilities, can lead to a posterior analysis of its performance. Therefore, the purpose of this project is to evaluate the best Neural Networks to recognize motions.

Keywords: Neural Networks, LSTM, CNN, FNN, Machine Learning, Deep Learning, Artificial Intelligence.

Índice

Lista de Figuras

Lista de Tabelas

Listagens

Lista de Acrónimos

AWS	*Amazon Web Services*
Bi-LSTM	*Bidirectional Long Short-Term Memory*
CNN	*Convolutional Neural Network*
CSV	*Comma-separated Values*
DL	*Deep Learning*
FFT	*Fast Fourier Transform*
FNN	*Feedforward Neural Network*
FPGA	*Field-programmable Gate Array*
GPS	*Global Positioning System*
GRU	*Gated Recurrent Unit*
HAR	*Human Activity Recognition*
IA	Inteligência Artificial
IMC	*Internal Model Control*
LSTM	*Long Short-Term Memory*
ML	*Machine Learning*
OMS	Organização Mundial de Saúde
RNN	*Recurrent Neural Network*
SVM	*Support Vector Machines*
TTS	*Text to Speech*

Capítulo 1

Introdução

A Inteligência Artificial (IA) representa uma área da ciência da computação em constante evolução nos dias de hoje, que se foca essencialmente no desenvolvimento de algoritmos e sistemas que realizem tarefas que, de certa forma, exijam inteligência humana. O *Deep Learning* (DL) é uma subcategoria da IA, cujo foco se prende no treino de redes neuronais, para que possam aprender a partir de dados brutos (sem qualquer tipo de tratamento prévio), sem a necessidade de intervenção humana.

Estas técnicas têm-se expandido para várias áreas, incluindo reconhecimento de fala, visão computacional e processamento de linguagem natural. Um grande exemplo desta última é o *ChatGPT*, que se tem vindo a tornar uma ferramenta muito útil no dia a dia de muitas pessoas.

Já o DL tem sido aplicado no desenvolvimento de vários sistemas, como o caso dos sistemas de reconhecimento de movimentos que poderão futuramente, ajudar no diagnóstico e tratamento de várias condições de saúde: deteção de sinais precoces de doenças neurológicas, como *Parkinson*, ou no auxílio de pacientes na reabilitação após lesões que afetam a capacidade motora.

Com respeito à utilização cada vez maior de IA nas mais diversas áreas, a Organização Mundial de Saúde (OMS) publicou recentemente, um relatório sobre o mesmo tema [1], destacando as oportunidades e desafios resultantes da crescente aplicação da IA na área da saúde. Também fornece orientações sobre como maximizar os benefícios da sua utilização, enquanto minimiza os seus riscos, enfatizando a importância de colocar a ética e os direitos humanos no centro do projeto, aplicação e utilização destas tecnologias.

Concluindo, o DL é uma técnica poderosa, com cada vez mais aplicações nas mais diversas áreas. A implementação de sistemas de reconhecimento de movimentos pode, futuramente, auxiliar no diagnóstico e tratamento de várias condições de saúde. No entanto, é importante garantir que a ética e os direitos humanos sejam sempre cumpridos no contexto de utilização desta tecnologia.

1.1 Contextualização

A IA tem vindo a ocupar uma posição de grande relevância na atualidade, tendo ressurgido após um hiato de décadas.

Os objetivos do estudo desta tecnologia prendem-se com aplicações de auxílio em tarefas do dia a dia, como um recurso para pesquisa - como o caso do *ChatGPT*, tema amplamente discutido na atualidade, sendo já objeto de integração em motores de pesquisa e navegadores de *Internet*. O projeto vem sendo arquitetado com o objetivo de complementar a informação obtida através dos resultados de uma pesquisa, acelerando o processo de aquisição de conhecimento por parte do utilizador.

Na área da robótica e automação, são vários os exemplos de sistemas controlados por algoritmos de IA. Esta aplicação justifica-se, principalmente, em sistemas de controlo não lineares dado que, quando comparados com algoritmos puramente matemáticos de controlo de sistemas, como o método *Ziegler-Nichols* ou até mesmo, do método *Internal Model Control* (IMC), as técnicas de *Machine Learning* (ML) e DL oferecem uma melhor resposta a perturbações, aumentado a sua rapidez e precisão.

O presente documento tem o objetivo de explicitar e descrever a aplicação de métodos de DL num contexto de reconhecimento de movimentos, detetados por vários tipos de sensores, geralmente presentes nos *Smartphones*, com vista a executar simulações sobre os métodos utilizados e concluir qual se enquadra melhor no problema.

1.2 Definição do Problema

O DL permite resolver várias questões relacionadas com os movimentos do ser humano. Algumas delas são:

1. **Reconhecimento de gestos em contexto de jogos ou realidade virtual:** o reconhecimento de movimentos pode ser aplicado em jogos, na robótica, na realidade virtual, entre outros. Um exemplo desta aplicação foi criado pela *Google*, aquando do desenvolvimento de um sistema que faz reconhecimento de gestos humanos para controlar dispositivos eletrónicos [2].

2. **Análise de movimento humano:** neste caso, o DL é usado para fazer a análise de movimentos humanos em tempo real, emitindo *feedback* sobre a postura e a forma física do utilizador. Esta aplicação pode ser útil na área do desporto, fisioterapia, e, essencialmente em áreas relacionadas com a saúde [3].

3. **Deteção de anomalias:** estas aplicações permitem detetar quando o utilizador sofre uma queda ou quando efetua um movimento estranho. Isto pode tornar-se útil em ambiente hospitalar ou de internamento domiciliário, pois permite monitorizar a segurança dos pacientes ou idosos com um menor esforço por parte do cuidador [4].

Se focarmos a problemática no aumento da população idosa, verificamos que, com o passar do tempo, a taxa de mortalidade tem vindo a diminuir, prevendo-se até que, em 2060, haja cerca de 30 % de população idosa em todo o mundo [5]. Tendo isso em conta, a utilização de DL para monitorização dos movimentos desta fatia de população permitirá manter os cuidados de saúde, reduzindo a necessidade de ter um recurso humano constantemente a vigiar o estado de saúde de um idoso nestas condições.

1.3 Objetivos

Esta dissertação tem o objetivo de efetuar a aplicação e posterior análise de várias metodologias de DL. O objetivo passará por avaliar métricas como a precisão, e as matrizes de confusão geradas através do teste dos modelos treinados.

Através de simulações, pretende-se identificar determinadas atividades do ser humano, tais como andar, sentar-se, subir e descer escadas, entre outras.

1.3.1 Resultados esperados

No final da execução da proposta em causa, é expectável que se possa ter uma perspetiva geral dos resultados da aplicação de algumas tipologias de redes neuronais na problemática definida, podendo avaliar qual ou quais as melhores alternativas para o treino de uma rede com as *features* selecionadas.

1.4 Plano de Trabalho

A execução do sistema de reconhecimento de movimentos estendeu-se ao longo de dezasseis meses, tendo sido afetada por um hiato de cerca de quatro meses, em que, por questões de ordem pessoal e profissional, o desenvolvimento foi muito baixo ou até mesmo, nulo.

Para facilitar a visualização do progresso do projeto e o respetivo plano, foi criado o diagrama de Gantt da Figura 1.1.

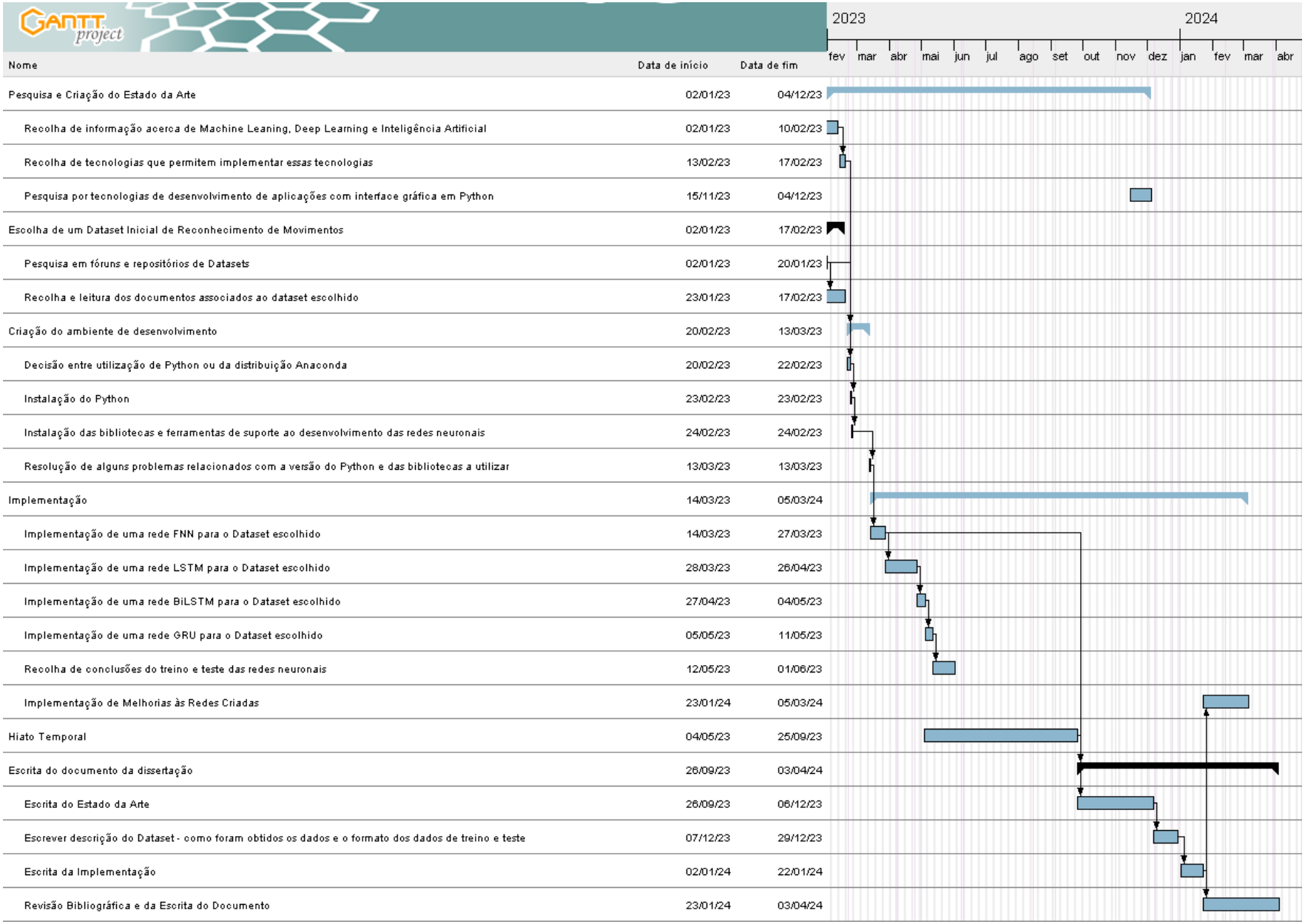


Figura 1.1: Planeamento do Projeto de Reconhecimento de Movimentos.

1.5 Organização da Dissertação

Este documento está organizado da seguinte forma:

- O presente capítulo destina-se essencialmente, a apresentar a problemática a resolver ao longo do projeto, assim como a introduzir as tecnologias que estiveram na base da implementação de sistemas de reconhecimento de movimentos.
- O segundo capítulo encarrega-se de apresentar o estado da arte, centrado na temática do reconhecimento de movimentos, mas também, na evolução da IA, ML e DL. Também são apresentados os diferentes métodos de obtenção dos dados de movimentos, que permitiram criar um *dataset* robusto, que foi usado como ponto de partida para o treino das redes neuronais criadas.
- O capítulo 3 lista e detalha as diferentes redes neuronais possíveis de aplicar no contexto de DL. Também apresenta as diferentes métricas utilizadas para avaliar a precisão de uma rede neuronal e diferentes metodologias de otimização de hiperparâmetros.
- O quarto capítulo aborda o tópico do *UCI HAR Dataset*, que foi escolhido para a base do treino das redes neuronais. Neste capítulo, são apresentadas algumas tecnologias e sensores utilizados na obtenção e categorização dos dados de movimentos.
- O quinto capítulo descreve as diferentes fases de implementação das redes neuronais, com exemplos de código usado para o tratamento dos *arrays* multi dimensionais usados na entrada das redes, e o subsequente treino das mesmas.
- O capítulo 6 destina-se à discussão dos diferentes resultados obtidos com cada rede neuronal.
- O sétimo e último capítulo deste documento apresenta as conclusões retiradas ao longo da criação e pesquisa que culminaram no sistema de reconhecimento de movimentos, nomeadamente, a rede neuronal que apresentou o melhor desempenho.

Capítulo 2

Estado da Arte

O presente capítulo aborda os conceitos relacionados com aprendizagem máquina e as metodologias de implementação prática de cada um deles. Também são abordadas questões relacionadas com as vantagens e os potenciais perigos de utilizar sistemas inteligentes em detrimento de outras técnicas. Na parte final, são descritas algumas aplicações das metodologias de aprendizagem máquina em vários contextos do dia a dia.

2.1 Inteligência Artificial

A IA concentra-se essencialmente no desenvolvimento de algoritmos e sistemas, que permitam simplificar o trabalho humano em várias tarefas rotineiras [6]. O ML é um subgrupo da IA, que se foca no desenvolvimento de algoritmos que permitem aos computadores aprender a partir de dados [7]. Já o DL é um subconjunto do ML cujo foco se prende com o desenvolvimento de redes neuronais profundas (daí a utilização do termo "*Deep*"), capazes de aprender a partir de dados não estruturados, como imagem e som [8].

Cada vez mais é possível verificar o rápido avanço da tecnologia a nível mundial. Após um elevado desenvolvimento das ferramentas computacionais (tanto a nível de *hardware* como de *software*), começou-se a questionar até que ponto as máquinas poderiam ser "ensinadas", de um modo semelhante ao que se faz com os seres humanos. No caso de sucesso, este tipo de equipamentos poderiam funcionar como uma ferramenta de auxílio ao ser humano em várias tarefas cognitivas do dia-a-dia,

podendo evitar erros gerados, por exemplo, por episódios de fadiga ao longo de um dia de trabalho. Além disso, a precisão desse tipo de sistema poderia atingir valores muito superiores aos de um ser humano, permitindo um desenvolvimento cada vez mais rápido e sustentável nos vários ramos de aplicação. Foi através desta premissa que se iniciou a investigação na área da IA e aprendizagem máquina.

Segundo Tom M. Mitchell no livro intitulado "*Machine Learning*" [9]: "A successful understanding of how to make computers learn would open up many new uses of computers and new levels of competence and customization. And a detailed understanding of information processing algorithms for machine learning might lead to a better understanding of human learning abilities (and disabilities) as well.". Pode inferir-se que, quando se começou a analisar as elevadas capacidades computacionais, surgiu um conceito que propôs criar uma ferramenta que permitisse conhecer melhor as capacidades (e incapacidades) do ser humano, dando uma oportunidade para que a máquina pudesse ter um papel acrescido às capacidades humanas de resolução de problemas.

A IA acaba por ter várias aplicações possíveis, tais como:

- Aprendizagem destinada ao reconhecimento do discurso verbal e não verbal;
- Condução autónoma;
- Deteção de tumores e outras patologias;
- Aprendizagem da mecânica de jogo como xadrez, damas, gamão, entre outros.

A IA é, na sua essência, um método de simulação do comportamento humano, com o objetivo de promover a automatização de processos.

Segundo dados fornecidos pela *Mathworks* [10], estima-se que, em 2030, a IA contribua para uma receita de 13 biliões de dólares em valor económico.

São imensas as possibilidades de utilização da IA, mas, atualmente, a maioria dos projetos estão relacionados com a condução autónoma, previsão de falhas em máquinas, análise de dados provenientes de sensores e a aplicação deste tipo de algoritmos em sistemas robóticos.

O processo de criação de uma IA baseia-se em vários princípios, tornando-o um pouco complexo e demorado. O modelo dessa inteligência é apenas uma pequena peça na vasta quantidade de processos necessários para obter um bom produto. Enumerando as etapas principais, distinguem-se 4 passos necessários [10]:

- Preparação dos dados;
- Criação do Modelo de Treino;
- Desenho da plataforma base em que o modelo vai ser utilizado;

- Implementação do modelo na plataforma criada, de forma a ser utilizada em contexto de produção.

A Figura 2.1 é uma representação gráfica das quatro etapas referidas acima.

Figura 2.1: Etapas da Criação de IA [10].

2.1.1 Preparação dos Dados

Geralmente, este é o passo que requer um maior esforço por parte de um criador de IA, dado que é necessário tratar uma quantidade elevada de dados que, muitas vezes, não apresentam qualquer tipo de coerência. De forma a efetuar o correto tratamento destes dados, é necessário ter conhecimento sobre a origem dos mesmos. Por exemplo: no caso de os dados a tratar se referirem a espetros de frequência, é importante que quem faz o tratamento entenda os conceitos principais do processamento digital de sinais.

O processo de tratamento implica, em muitos casos, a remoção de dados que possam eventualmente, constituir ruído, ou seja, ter informação irrelevante para o caso de estudo. A presença deste ruído afeta a precisão de um modelo de IA, pois o mesmo estará a ser criado, partindo de dados que não estão corretos.

Outro ponto menos positivo da preparação de dados prende-se com a necessidade de obter volumes elevados dos mesmos (para uma boa precisão, são necessários milhões de dados, sendo este número, em alguns casos, um valor baixo).

2.1.2 Criação do Modelo

Segundo o site *Dataconomy* [11]: "AI models are mathematical representations of a system that can make predictions or decisions based on the input data. AI models can range from simple linear models to complex neural networks.".

A *Mathworks* afirma que as *guidelines* principais para obter uma IA são [10]:

- Iniciar a criação do modelo com um conjunto de algoritmos e modelos pré-treinados. Desta forma, é possível reaproveitar imensos dados que já foram treinados previamente. Assim, é possível criar uma IA com base em modelos solidamente treinados.

- Utilizar sistemas abertos à integração com várias ferramentas de IA, tais como a bilioteca *PyTorch* [12], *Tensorflow* [13] ou até as tecnologias que a própria *Mathworks* apresenta (*MATLAB* e *Simulink*) [14].

2.2 Projeto da Plataforma Base

Todos os sistemas baseados em IA necessitam de uma plataforma para executar os seus algoritmos. Tomando como exemplo um sistema de condução autónoma, torna-se evidente a necessidade de interligar a IA com equipamentos de localização, como o *Global Positioning System* (GPS), e controlos de travagem e aceleração para que a tarefa possa ser executada com todas as ferramentas que necessita.

Para projetar uma plataforma base para um algoritmo de IA, é necessário seguir algumas etapas [15]:

- Deve-se definir o objetivo da plataforma e as necessidades para a execução do algoritmo.
- Escolher a infraestrutura de suporte para a plataforma, desde o *hardware* ao *software*.
- Projetar a arquitetura da plataforma, nomeadamente o tipo de algoritmo de IA a aplicar e a definição das camadas de processamento.

Um exemplo de uma plataforma base para um algoritmo de IA é o *Azure Machine Learning* da *Microsoft*. Este serviço fornece uma plataforma completa para criar e implementar modelos de ML, suportando vários tipos de algoritmos. Além disso, permite a integração com outras ferramentas do *Azure*, como o *Azure Data Factory* [16] e o *Azure Databricks* [17].

A *Amazon* também criou uma solução, denominada *Amazon SageMaker* [18], que permite criar, treinar e implementar modelos de ML, oferecendo integração com outras ferramentas do ecossistema da *Amazon Web Services* (AWS).

2.3 Implementação em Contexto de Produção

Chegada a última fase da criação de uma IA, é necessário aplicar o modelo estudado num sistema real, com todas as ferramentas estudadas no passo anterior. Para isso, são necessários processadores lógicos e gráficos e até *Field-programmable Gate Array* (FPGA) para dar o suporte necessário ao sistema. O material necessário à implementação varia consoante a necessidade de recursos de *hardware* para executar o objetivo planeado ou até decisões externas, tomadas pelo departamento financeiro da empresa que vai utilizar a tecnologia, por exemplo.

Em certos casos, é possível usufruir da capacidade do modelo criado para gerar código que se adapte à máquina na qual a tecnologia será aplicada. Desta forma, é possível otimizar a IA aos recursos disponíveis.

2.4 Prós e Contras da Inteligência Artificial

A IA é uma tecnologia que permite criar máquinas inteligentes, capazes de executar tarefas sem intervenção humana. As máquinas baseadas em IA são programadas para aprender e explorar tudo à sua volta, com o auxílio da análise de dados. No mundo real, pode-se ver várias aplicações da IA que permitem automatizar várias atividades do dia a dia.

Alguns exemplos de aplicação da IA são a mobilidade autónoma, *marketing* automatizado, *e-commerce*, sistemas de recomendação, deteção automática de fraudes, entre várias outras. Com as várias aplicações da IA, vão surgindo vantagens e desvantagens da sua utilização.

Alguns pontos a favor da IA são os seguintes:

Processamento de Dados sem Erros A execução de tarefas por humanos tem associada uma probabilidade elevada de erro. Esses erros são dificilmente evitáveis, variando consoante a habilidade intelectual de cada um. O mesmo não ocorre quando a mesma atividade é executada por um sistema baseado em IA. Desta forma, a precisão inerente à atividade desenvolvida está relacionada apenas com o treino da IA.

Auxílio em Tarefas Repetitivas Contrariamente aos humanos, as máquinas não necessitam de pausas ou outro tipo de concessões. Este facto permite que o tempo seja totalmente aproveitado para executar várias tarefas rotineiras, permitindo ao humano, aproveitar esse tempo para ser mais produtivo. Esse acréscimo de produtividade permite uma maior produção de bens num tempo significativamente menor, respondendo à elevada procura que se vem verificando.

Disponibilidade 24 Horas por Dia Em média, um trabalhador consegue ser produtivo por cerca de 7 a 8 horas por dia. Além disso, devem-se considerar todas as pausas que naturalmente, são necessárias efetuar ao longo dessas 7 a 8 horas, restando pouco tempo efetivamente produtivo.

No caso da IA, os tempos produtivos podem atingir facilmente as 24 horas sem quaisquer paragens. Estas máquinas podem trabalhar ininterruptamente em tarefas, sem necessidade de supervisão constante. Isto constitui um dos maiores prós da utilização desta tecnologia nos dias que correm. Um exemplo de aplicação deste conceito verifica-se nos *call centers*. Um *chatbot*, por exemplo, consegue responder a vários pedidos em simultâneo, durante as 24 horas que compõem o dia. Apesar da limitação em certos casos, conseguem dar resposta a vários problemas que lhes são colocados, sem necessidade de intervenção humana.

Tomada de Decisão Mais Correta Uma das vantagens da IA é a sua capacidade de tomar, na maioria das vezes, as decisões mais corretas. Não tendo qualquer conexão emocional, os sistemas com base nesta tecnologia conseguem tomar decisões puramente baseadas na lógica. Este facto poderá ter consequências positivas ou negativas, no sentido em que, algumas vezes, o conceito de ética entra em conflito com a racionalidade lógica, podendo tornar estas máquinas, potencialmente perigosas. A base destas tomadas de decisão centra-se na computação cognitiva.

Segundo Haluk Demirkan, Seth Earley e Robert R. Harmon, num artigo publicado na revista *IT Professional*, a computação cognitiva define-se como o conjunto de sistemas que conseguem aprender e interagir com humanos naturalmente, não sendo necessário programá-los explicitamente, pois conseguem aprender através da interação e das experiências passadas [19].

Apesar das vantagens listadas acima, também há algumas fragilidades associadas à utilização desta tecnologia. Tais são:

Altos Custos de Criação A criação de sistemas baseados em IA constitui um trabalho moroso e de elevado custo, pelo que não é monetariamente acessível a qualquer pessoa. Para projetos de maior escala, o custo pode rondar os milhões de dólares [20]. Os custos poderão variar consoante o *hardware* e/ou *software* que as empresas utilizam.

Aumento dos Despedimentos por Substituição do Trabalho Partindo da premissa de que a IA permite executar tarefas ininterruptamente, com elevados níveis de precisão, muitas empresas tendem a estudar a possibilidade de substituir a mão de obra humana pela digital, pois, à partida, as consequências que daí advém aparentam ser positivas. Por exemplo, não há necessidade de pagar um salário mensal a uma máquina que executa tarefas rotineiras. Ao invés disso, proceder-se-á ao despedimento de alguns funcionários, deixando apenas recursos suficientes para desenvolver atividades mais complexas, que não possam depender somente de sistemas baseados em IA.

Diminuição da Criatividade Se o trabalho passar a ser realizado somente por máquinas, poderá haver o risco de a criatividade desaparecer. Se não houver evolução nessa área, a tendência será de diminuição da criatividade ao longo do tempo, já que as máquinas não são capazes de inventar.

2.5 *Machine Learning*

ML é um conceito criado para reproduzir o método de ensino pela experiência em computadores, de forma a que o mesmo melhore o seu desempenho à medida que

o número de dados disponíveis para a aprendizagem aumenta [21]. De uma forma visual, o conceito de ML pode ser visto como um subconjunto da IA.

A aprendizagem máquina é conseguida através de algoritmos que implementam métodos computacionais para "aprender" diretamente a partir dos dados sem depender de modelizações matemáticas, cuja variação não é tão "inteligente".

O conceito de IA pode simplificar-se, assumindo que é um *software* que se tenta aproximar do método de pensamento dos seres humanos, com o objetivo de realizar tarefas complexas autonomamente. ML recorre a algoritmos puramente matemáticos, treinados a partir de vários dados, que têm a capacidade de executar as tais tarefas complexas.

Dessa forma, pode-se concluir que ML constitui um sub-grupo da IA, sendo a principal base de construção das mesmas na atualidade, segundo o artigo "Machine Learning vs. AI: Differences, Uses, and Benefits", publicado pela página Coursera [22]. Tal facto, explica a natural confusão que se gera quando se apresentam ambos os conceitos.

Os exemplos mais comuns da aplicação de ML são:

- Receção de recomendações em plataformas de *streaming* de vídeo, como o *Youtube*;
- Resolução de problemas, através da utilização de *chatbots*;
- Assistentes virtuais, tais como a *Google Assistant* ou *Siri*, que permitem responder a pedidos do utilizador, relacionados com um ecossistema de aplicações às quais tem acesso (calendário, música, entre outras).

À semelhança da IA, também o ML tem diversas aplicações por si só. Tais são:

Saúde As atividades de saúde que são praticadas diariamente produzem grandes quantidades de dados, que ficam registados sob as mais variadas formas. A agregação de todos estes dados permite criar modelos de aprendizagem eficazes no diagnóstico de doenças, auxiliando nas atividades mais demoradas e repetitivas. Um exemplo comum da utilização de ML neste contexto é a leitura de raios X, através da utilização de visão computacional.

Este conceito representa um campo da IA que permite aos computadores, simular a leitura visual de um ser humano, detetando padrões em cada imagem. Segundo a IBM: "If AI enables computers to think, computer vision enables them to see, observe and understand." [23].

Gestão de Recursos Vários algoritmos de ML são desenvolvidos para prever possíveis gastos num contexto empresarial, recorrendo a *Big Data*, que fornece uma

vasta quantidade de informação relacionada com a problemática em questão. Estes algoritmos permitem fazer uma gestão de recursos (financeiros, humanos, entre outros), de forma a atingir a máxima otimização possível.

Cadeias de Fornecimento As cadeias de fornecimento são responsáveis por imensas transações a nível mundial. Com o passar do tempo, estas transações vão-se tornando mais complexas, contribuindo para um aumento do erro humano aquando da sua gestão. Para reduzir a probabilidade de ocorrência desse erro, e também para acelerar os processos associados a esta temática, são usados algoritmos de ML que, de forma semelhante à gestão de recursos, fazem previsões de possíveis atrasos nas entregas, entre outras variáveis de estudo.

2.5.1 Tipos de *Machine Learning*

Os algoritmos de ML dividem-se em três tipos: aprendizagem supervisionada, não supervisionada e *Reinforcement Learning* [24].

Aprendizagem Supervisionada

A aprendizagem supervisionada é um algoritmo que treina uma rede neuronal com um conjunto de dados que já foram classificados ou categorizados. O objetivo do algoritmo é aprender a classificar novos dados com base no que foi aprendido durante o treino [21].

Aprendizagem Não Supervisionada

A aprendizagem não supervisionada constitui um método em que o algoritmo treina a rede neuronal com um conjunto de dados que não foram previamente classificados ou categorizados. O objetivo deste algoritmo é encontrar padrões nos dados fornecidos e agrupá-los em categorias [21].

Aprendizagem por Reforço (*Reinforcement Learning*)

Finalmente, o *Reinforcement Learning* é um método em que o algoritmo aprende a tomar decisões com base em *feedbacks* positivos ou negativos, recebidos durante o treino. Este *feedback* pode ser dado por utilizadores (humanos) e o principal objetivo do algoritmo é maximizar as recompensas[1] recebidas ao longo do tempo.

A Figura 2.2 representa os três tipos de ML existentes, enumerando também algumas aplicações de cada algoritmo.

[1]Por recompensa, entende-se o maior número de *feedback* positivo relativamente a uma determinada decisão do modelo [21].

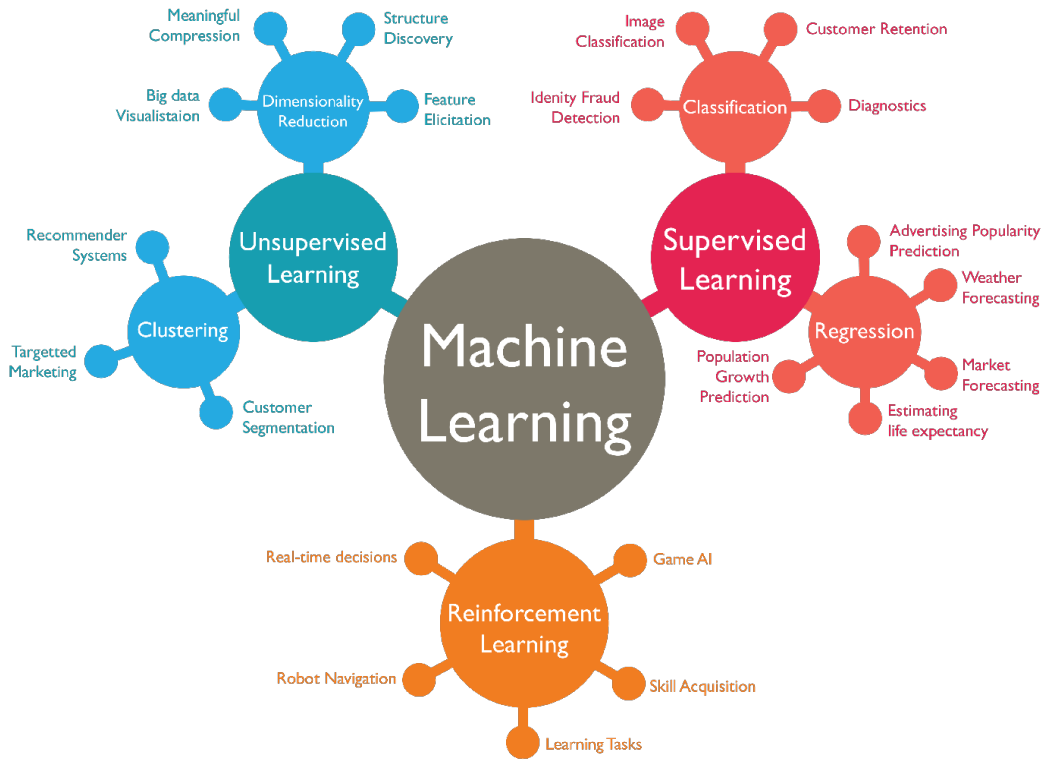


Figura 2.2: Diagrama com os Tipos de ML [25].

2.6 *Deep Learning*

Dentro do vasto universo do ML, há uma tipologia de aprendizagem denominada de *Deep Learning* (DL).

Esta variante é específica de modelos que aprendem a executar tarefas de classificação, partindo de imagens, texto ou som [21].

O reconhecimento de movimentos é uma aplicação do DL, que permite aos computadores, identificar e classificar vários movimentos humanos, em tempo real, com base em dados capturados por sensores.

O modelo de base para o DL é, na grande maioria das vezes, a rede neuronal, constituída por vários neurónios, organizados por camadas, que podem chegar às dezenas ou centenas, daí a utilização de "*Deep*" no conceito geral.

A profundidade de uma rede neuronal caracteriza-se pela quantidade de camadas que a constituem, sendo que, quanto mais camadas a rede possuir, mais profunda se considera.

O DL apresenta-se como um dos métodos de aprendizagem máquina mais difíceis de implementar, no sentido em que necessita de uma grande quantidade de dados para poder prover uma boa precisão. No entanto, satisfazendo este requisito, é dos métodos mais precisos atualmente para a geração de IA.

2.6.1 O Neurónio

O neurónio é a célula do sistema nervoso responsável pela condução de um impulso nervoso. É composto, essencialmente por dendrites, corpo e axónio. A separação entre o meio interno e o externo é denominada de membrana neuronal, a qual é sustentada por um esqueleto interno: o citoesqueleto. O neurónio é a unidade básica da estrutura do cérebro e do sistema nervoso humano [26].

As redes neuronais, tal como o nome indica, são sistemas computacionais que se inspiram no funcionamento do cérebro. As mesmas apresentam camadas de neurónios que processam dados, permitindo que, quando treinados, compreendam o discurso verbal e identifiquem alguns objetos. A informação é passada através de cada camada, com a saída da camada anterior a fornecer entrada para a camada seguinte [21]. A Figura 2.3 demonstra a similaridade entre um neurónio humano e o modelo de neurónio utilizado em DL.

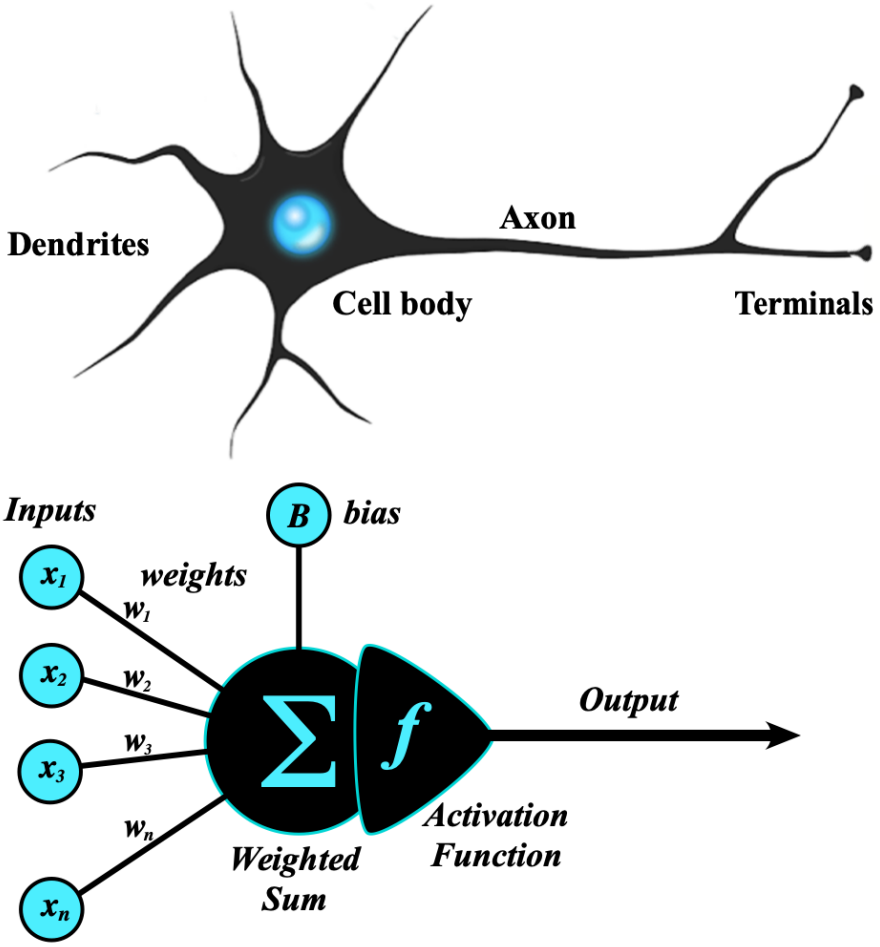


Figura 2.3: Neurónio Humano *vs.* Modelo de Neurónio em DL [27].

O conceito de neurónio no contexto das redes neuronais remonta à década de

1940, quando o neurofisiologista Warren McCulloch e o matemático Walter Pitts criaram um modelo matemático que permitia explicar o funcionamento de um neurónio humano e, consequentemente, da componente racional do cérebro.

Esse modelo iniciou um processo de pesquisa na área da ciência da computação, que dividiu o entendimento sobre redes neuronais em duas abordagens: a primeira, focada em processos biológicos no cérebro, enquanto a segunda era focada na aplicação de redes neuronais à IA [28].

Em 1949, Donald Hebb escreveu o livro "The Organization of Behavior", em que apontou o facto de que os caminhos neuronais são reforçados à medida que são usados. Este conceito tornou-se fundamental para explicar a forma como os humanos aprendem [29].

2.7 *Transfer Learning* e Redes Pré-Treinadas

O *Transfer Learning* é o termo dado ao aproveitamento do treino efetuado numa rede neuronal, que permite reduzir os tempos de treino para o novo caso de estudo, não impactando negativamente em variáveis como a precisão. Na prática, devem apenas ser ajustados os parâmetros necessários para satisfazer as necessidades do projeto, nomeadamente, a quantidade de saídas dessa mesma rede.

Por exemplo: assumindo que se quer criar um sistema de reconhecimento de cães e gatos, é possível recorrer a redes neuronais que já foram treinadas para um propósito semelhante (reconhecimento de animais), eliminado a necessidade de obter grandes quantidades de dados para poder treinar uma rede neuronal de raiz. Para ajustar a rede às necessidades do projeto, podem-se remover as saídas que não sejam cão ou gato, já que não fazem parte do âmbito.

As redes neuronais pré-treinadas em nada diferem das restantes redes neuronais. Este conceito é usado apenas quando se trata de redes que foram treinadas a partir de um grande conjunto de dados antes de serem usadas para uma tarefa específica. Estas redes são capazes de reconhecer padrões complexos em dados fornecidos, como imagens ou texto, e podem ser reutilizadas num vasto leque de tarefas [30].

A principal vantagem da utilização das redes neuronais pré-treinadas é a de poder economizar tempo, já que, por possuírem pesos ajustados, os tempos de treino subsequentes diminuem. No entanto, também se pode considerar o melhor desempenho destas redes, dado que as mesmas permitem extrair recursos mais úteis dos dados [31].

Alguns exemplos de redes neuronais pré-treinadas são:

***VGG16*:** é uma rede neuronal convolucional, treinada com o conjunto de dados *ImageNet*. É usada para tarefas de classificação de imagens [32].

***Inception-v3*:** é uma rede neuronal convolucional, semelhante à *VGG16*, sendo treinada igualmente com o conjunto de dados *ImageNet* [33].

***BERT*:** é uma rede neuronal criada para tarefas de processamento de linguagem natural, como classificação de texto [34].

***GPT-2*:** é uma rede neuronal semelhante à *BERT*, treinada com um grande conjunto de dados, com o objetivo de processar linguagem natural [35].

2.8 Redes Monocamada e Multicamada

Uma rede neural monocamada é o formato de rede neuronal mais simples, pois possui apenas uma camada de neurónios - a camada de entrada. Este tipo de rede é usado em tarefas de classificação binária ou regressão, como é o caso dos sistemas de deteção de *spam* ou até, voltando ao exemplo da secção 2.7, a classificação de imagens de gatos ou cães.

No segundo caso, a rede monocamada teria um único neurónio, que representa a probabilidade de ser um gato. Caso a saída seja maior que 0.5, a entrada é classificada como um gato, caso contrário, como um cão. Uma rede monocamada tem o aspeto da Figura 2.4.

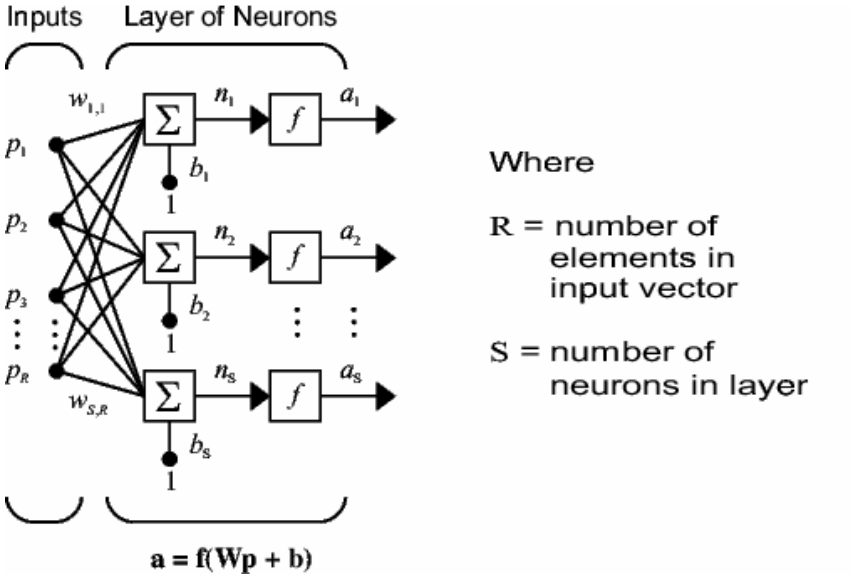


Figura 2.4: Rede Monocamada [36].

Já as redes neuronais multicamada, possuem várias camadas ocultas - que são camadas que se encontram entre as de entrada e saída. Essas camadas ocultas contêm neurónios intermediários, que processam informações de maneira não linear. Considerando como exemplo, a classificação de flores *Iris*, a rede multicamada poderá ter [21]:

1. **Camada de Entrada:** conjunto de neurónios que representa a largura e o comprimento das pétalas e sépalas.
2. **Camada Oculta:** constituída por uma quantidade de neurónios à escolha do programador.
3. **Camada de Saída:** apresenta 3 neurónios, correspondendo às diferentes classes de flores *Iris* (*Setosa*, *Virginica* e *Versicolor*).

A Figura 2.5 mostra o formato de uma rede multicamada.

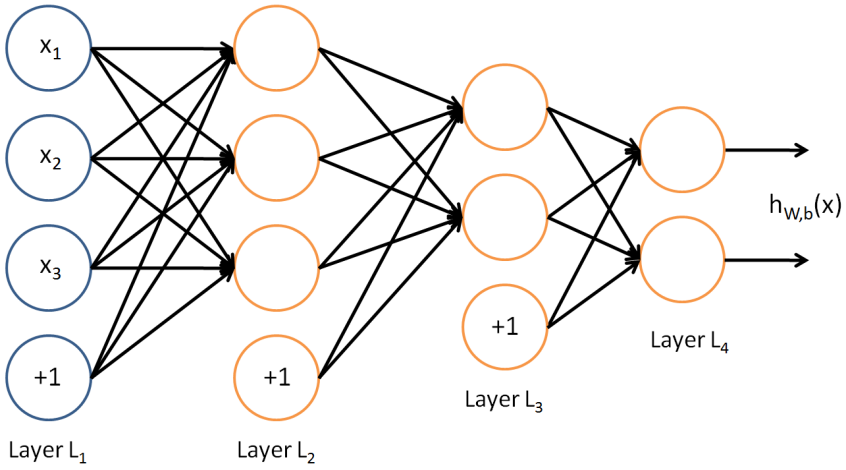


Figura 2.5: Rede Multicamada [37].

Em conclusão, as redes neuronais multicamada são mais poderosas, já que permitem a aprendizagem de representações complexas, além da regressão linear.

2.9 Aplicações das Tecnologias de Aprendizagem

A aplicação das três principais metodologias de aprendizagem resultam em vários exemplos de utilização: o reconhecimento de voz, usando *Convolutional Neural Network* (CNN), deteção de rosto, condução autónoma, deteção de objetos, deteção de defeitos, cancro gastro intestinal, COVID-19 através de radiografias pulmonares e classificação de sinais, utilizando redes *Long Short-Term Memory* (LSTM).

2.9.1 Processamento de Linguagem Natural

O processamento de linguagem natural é um ramo da IA que se foca em permitir que os computadores entendam e processem a expressão verbal humana. Esta área de estudo combina três especialidades: a linguística, a ciência da computação e a estatística, de forma a permitir uma melhor implementação dos modelos de processamento. O seu principal objetivo é permitir que os computadores façam o

processamento e análise de grandes volumes de dados, como texto e voz, para extrair informações úteis.

No contexto de DL, o processamento de linguagem natural envolve a criação de modelos que aprendam a partir de grandes conjuntos de dados rotulados, para aprender a mapear entradas (texto ou voz) e gerar saídas. Uma vez treinados, os modelos podem ser usados para classificar novos dados ou gerar novas saídas [38].

Existem muitas técnicas diferentes usadas no processamento de linguagem natural, nomeadamente, a análise sintática, a análise semântica e a análise pragmática.

A análise sintática compreende a identificação da estrutura gramatical das frases, enquanto a análise semântica engloba a extração do significado das mesmas. Já a análise pragmática envolve a interpretação de frases no contexto em que estão inseridas [39].

2.9.2 Robótica

A robótica é uma área da engenharia que trata o projeto, construção e operação de robôs. Existem várias aplicações para a robótica, tais como: manufatura, exploração espacial, medicina e entretenimento.

Aquando da interligação com DL, envolve-se a criação e treino de modelos para mapear entradas, como imagens ou sensores, de forma a gerar saídas, como movimentos ou decisões. Uma vez treinados, esses modelos servem para controlar robôs em tempo real.

Algumas das técnicas utilizadas na robótica são a visão computacional, o processamento de sinal e o controlo de movimento [40].

2.10 Sensores de Movimento

Os sensores são dispositivos eletrónicos que têm o papel de converter grandezas físicas em sinais elétricos, que podem ser processados por um sistema computacional.

No contexto de reconhecimento de movimentos através de técnicas de ML ou DL, a utilização de sensores constitui um tema fulcral. Por esse motivo, é imperativo categorizá-los tendo em conta alguns aspetos, tais como a posição de leitura ideal de cada um e a forma sobre a qual os dados serão tratados posteriormente. A categorização dos sensores permite ter uma perspetiva mais organizada acerca das diferentes metodologias de tratamento a aplicar a cada um deles, na fase de implementação de um sistema.

Para criar um sistema de reconhecimento de movimentos, assumindo previamente que é efetuada a aquisição de dados para treino da rede neuronal, é necessário tratar esses dados e categorizá-los. De seguida, é efetuado o treino de um modelo de aprendizagem, para reconhecer padrões nos dados recolhidos.

Existem várias técnicas de DL para criar um sistema de reconhecimento de movimentos. Duas das técnicas mais comuns são as redes neuronais to tipo *Feedforward Neural Network* (FNN) e as redes neuronais recorrentes (ou RNN), capazes de extrair características importantes dos dados provenientes dos sensores e de tratar sequências temporais de dados [41].

Tendo por base o artigo intitulado "Energy Efficient Smartphone-Based Activity Recognition using Fixed-Point Arithmetic", dependendo dos critérios a considerar, são definidos vários modos de agrupar sensores, tais como [5]:

- **Por tipo de sensor:** que se baseia nos sinais obtidos por sensores inerciais, baseados em visão computacional e monitores psicológicos;
- **Por localização:** os sensores são considerados externos quando são colocados em posições fixas num determinado ambiente ou *wearable*, e internos quando presos a um corpo em movimento;
- **Por princípio de modelização:** que pode ser por tipo de dados ou por organização de conhecimento. A escolha depende de como os modelos *Human Activity Recognition* (HAR) são arquitetados;
- **Por aprendizagem:** que pode ser supervisionada, semi-supervisionada ou não supervisionada [42].

Os detalhes relacionados com a arquitetura dos sensores para obtenção de um *dataset* são aprofundados no Capítulo 4.

2.11 Aplicações de Reconhecimento de Movimentos

Atualmente, os sistemas de reconhecimento de movimentos têm, essencialmente, três focos de aplicação prática: a saúde, a segurança e a indústria do entretenimento. Incidindo nestas áreas com maior detalhe, pode-se listar as seguintes aplicações [43, 44]:

Reabilitação física Os sistemas de reconhecimento de movimentos podem ser úteis na área da reabilitação física, permitindo que os pacientes realizem exercícios com precisão, graças à monitorização em tempo real, característica deste tipo de sistema computacional.

Entretenimento Os sistemas de reconhecimento de movimentos têm um papel relevante em muitos jogos, pois permitem que os jogadores controlem personagens e objetos com gestos, servindo como base para outros conceitos em desenvolvimento na atualidade, tais como a realidade virtual.

Segurança Os sistemas de reconhecimento de movimentos podem ser usados para monitorizar vários espaços, com a finalidade de detetar atividades suspeitas ou perigosas em tempo real.

Robótica Este tipo de sistemas tem também aplicabilidade na robótica, na medida em que permitem que os robôs interajam com o ambiente envolvente, realizando tarefas complexas.

Saúde Os sistemas de reconhecimento de movimentos podem ser usados para monitorizar a saúde dos pacientes, detetando mudanças subtis nos padrões de movimento que podem ser indicativos de problemas de saúde.

Capítulo 3

Redes Neuronais

Neste capítulo, são apresentadas várias tipologias de redes neuronais, assim como as principais características (como o formato dos dados de entrada) e aplicações das mesmas. Também são descritas as principais métricas utilizadas na criação e treino dessas mesmas redes neuronais, assim como as principais técnicas de otimização.

3.1 Introdução

Como indicado anteriormente, as redes neuronais são sistemas computacionais que se inspiram no funcionamento do cérebro humano, usando camadas de neurónios para processar dados, compreender o discurso verbal e reconhecer objetos. A informação é passada através de cada camada, com a saída da camada anterior a fornecer a entrada para a camada seguinte [45]. Existem vários tipos de redes neuronais, tais como:

Perceptrão: Este tipo de rede é usado maioritariamente em tarefas simples, tais como classificação binária. Um perceptrão pode ser treinado para distinguir entre imagens de cães e gatos, por exemplo. No entanto, tem a limitação de apenas fazer uma classificação dual, ou seja, apenas distingue corretamente duas classses. Para que a rede possa distinguir um maior número de classes, necessita de mais perceptrões. A Figura 3.1 ilustra o formato de um perceptrão.

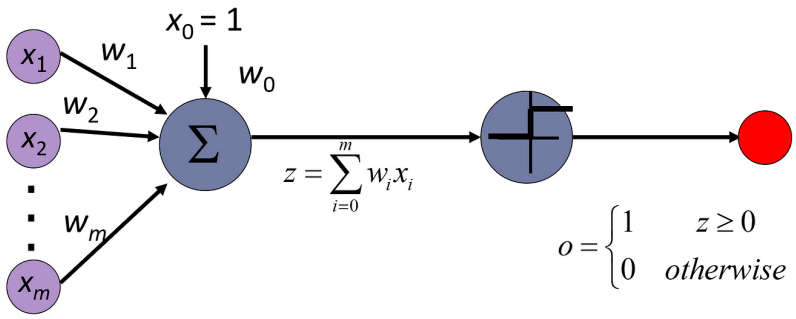


Figura 3.1: Perceptrão [46].

Redes Neuronais do tipo *Feedforward*: As redes FNN são consideradas as mais simples, dado que a informação é transportada num só sentido (da camada de entrada para a camada de saída, não sendo feita nenhuma realimentação da informação para a entrada ou retro-propagação). Nestas redes, os neurónios da mesma camada não interagem entre si.

Estas redes são usadas em tarefas como reconhecimento de discurso, reconhecimento de imagens e classificação de texto. Um exemplo de utilização desta rede é o *Google* Tradutor. A Figura 3.2 representa uma rede do tipo *FeedForward*.

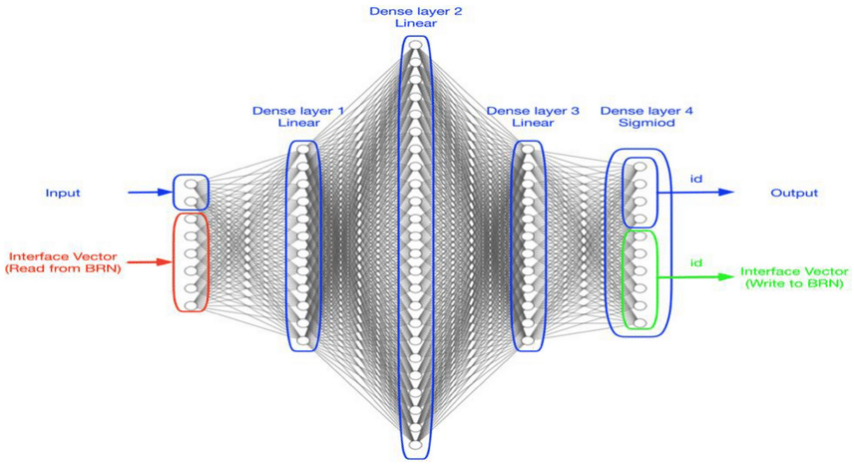


Figura 3.2: Rede Neuronal *Feedforward* [47].

Redes Neuronais Recorrentes: As redes RNN são utilizadas, essencialmente, em dados estáticos, tais como texto, pois têm a capacidade de prever a sequência do discurso com base no que fora escrito anteriormente. Esta é a base de muitas aplicações de *text completion*, como a presente no *Gmail*.

Uma das principais diferenças deste tipo de rede para uma rede do tipo *Feedforward* é a realimentação, ou seja, a utilização de previsões anteriores, segundo uma janela temporal, para efetuar previsões de saídas seguintes. Os dados são, por isso,

injetados novamente na entrada da rede, por forma a serem utilizados como contexto para a previsão subsequente.

Poderão também ser usadas em tarefas de previsão do estado do tempo e reconhecimento de voz. A Apple, por exemplo, utiliza este tipo de rede na sua assistente virtual, a *Siri* para entender e responder a comandos de voz. A Figura 3.3 evidencia o esquema de uma rede do tipo RNN.

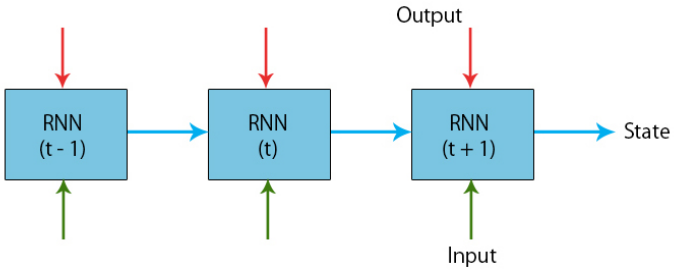


Figura 3.3: Rede Neuronal Recorrente [48].

Redes *Long Short-Term Memory* (LSTM): As redes neuronais LSTM são redes com a capacidade de detetar e aprender padrões em sequências de informação, tais como as fornecidas em ficheiros de texto. São, também capazes de processar informações sequenciais que dependem da memorização de cada elemento num conjunto, como é o caso de um vídeo curto [49]. As redes LSTM são mais adequadas para classificar, processar e prever séries temporais, com intervalos de tempo de duração desconhecida.

São um tipo de rede recorrente, pois recorrem a saídas anteriores para prever as entradas seguintes. A informação é guardada provisoriamente num elemento de memória designado de *Internal State*. Este elemento está presente em cada nó da rede, armazenando informações anteriores relevantes. Outro elemento constituinte deste tipo de rede são as *gates*, que permitem controlar o fluxo de informação da entrada e da saída, permitindo à rede escolher que informação é mais relevante.

Estas redes são utilizadas em aplicações como [50, 51]:

- **Tradução:** as redes LSTM são usadas em sistemas de tradução automática (como o *Google* Tradutor), permitindo traduzir frases de um idioma para outro.
- **Geração de texto:** estas redes podem ser usadas para gerar texto em linguagem natural, podendo criar notícias e artigos.
- ***Chatbots*:** No seguimento da geração de texto, estas redes podem também ser utilizadas em *chatbots*, para estabelecer uma conversa com o utilizador.

- **Transcrição de áudio:** as redes LSTM são usadas em sistemas *Text to Speech* (TTS) para transcrever áudio a partir de texto.
- **Geração de legendas em imagens**.
- **Reconhecimento de movimentos em vídeos**.

A Figura 3.4 apresenta o diagrama de blocos de uma rede do tipo LSTM.

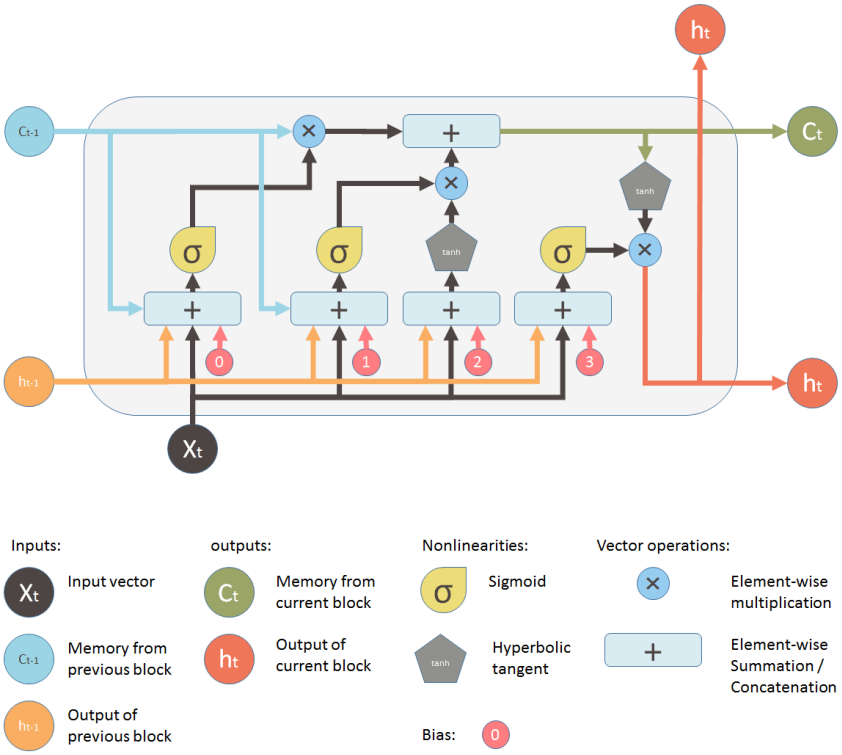


Figura 3.4: Rede LSTM [52].

Redes Neuronais Bidirecionais: Estas redes podem ser designadas por Bidirecionais ou Bi-LSTM. Como o próprio nome indica, são uma extensão da rede LSTM convencional, que consiste na utilização de duas camadas LSTM com sentidos de propagação inversos. Tomando como exemplo, a frase "Eu vou nadar hoje", a camada LSTM de propagação direta efetua a leitura da entrada da esquerda para a direita (sentido em que a frase é lida por humanos), enquanto que a LSTM de retro-propagação faz a mesma leitura no sentido inverso. Isto permite à rede efetuar um processamento mais rápido e eficaz quando comparado com uma simples rede LSTM.

Como este tipo de rede é utilizado somente para codificar a informação, é necessária a introdução de outras camadas para efetuar as previsões e testes.

A Figura 3.5 representa a estrutura de uma rede Bi-LSTM convencional.

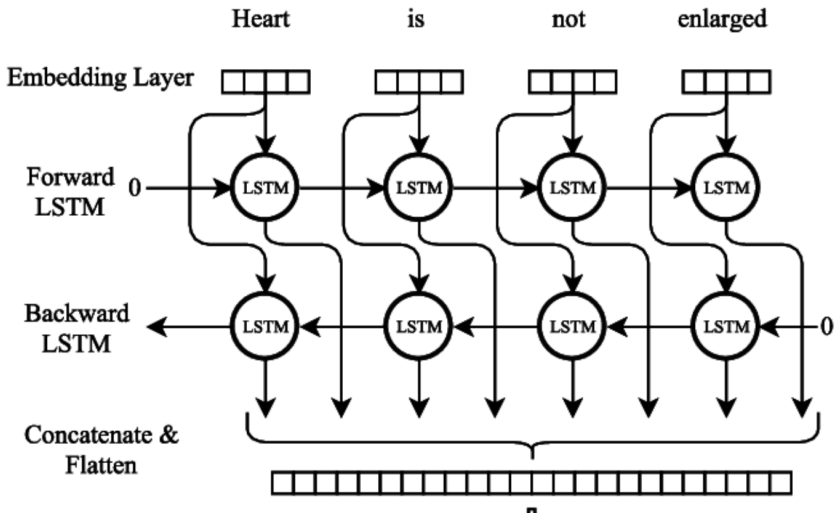


Figura 3.5: Estrutura de uma Rede Bi-LSTM [53].

Redes Neuronais *Gated Recurrent Unit* (GRU): Estas redes são outro exemplo de RNN, mais recentes do que as redes LSTM, cujo objetivo é colmatar a memória curta destas últimas, que nem sempre conseguem armazenar o contexto necessário para fazer uma boa previsão.

Para tal, as GRU combinam a memória de curto prazo (*short-term memory*) com a memória de longo prazo (*long-term memory*), aliado à utilização de 2 *gates*: *update gate*, que controla a quantidade de amostras passadas que devem ser armazenadas e *reset gate*, que controla a quantidade de amostras que podem ser esquecidas.

A Figura 3.6 representa a estrutura de uma rede GRU.

A Tabela 3.1 lista as principais diferenças entre a rede LSTM e a rede GRU.

Tabela 3.1: Diferenças Entre as Redes LSTM e GRU.

LSTM	**GRU**
Possui 3 *Gates* (Entrada, Saída e *Forget*)	Possui 2 *Gates* (*Update* e *Reset*)
Maior precisão em sequências de dados mais longas	Mais eficiente a nível computacional
Mais antiga	Mais recente e cada vez mais popular

Redes Neuronais Convolucionais: As redes CNN são usadas em reconhecimento facial, deteção de objetos e classificação de imagens. Esta rede é utilizada pelo *Facebook* para identificar rostos em fotos e fazer sugestões. A Figura 3.7 mostra uma rede CNN convencional.

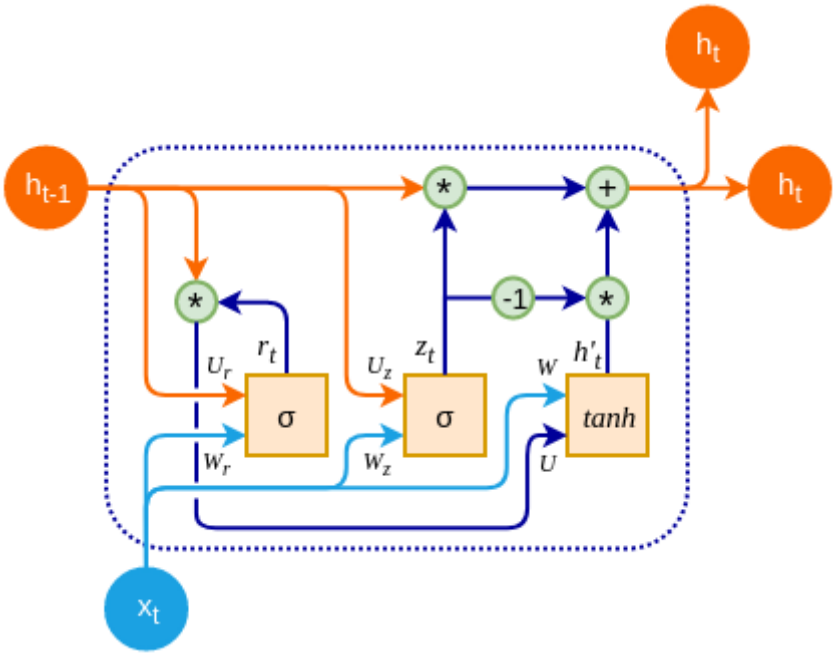


Figura 3.6: Estrutura de uma Rede GRU [54].

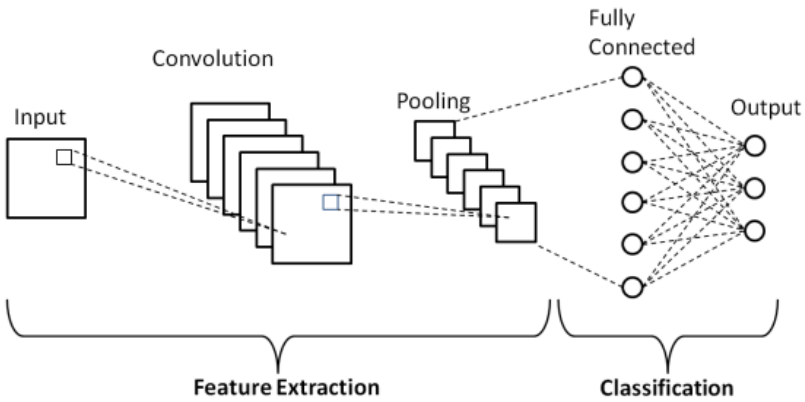


Figura 3.7: Rede Neuronal Convolucional [55].

Camada de *Dropout* As redes neuronais, em vários casos, atingem estruturas demasiado complexas, com várias não linearidades, podendo conter várias camadas ocultas. Nestes casos, a probabilidade de ocorrência de *overfitting* é crescente, principalmente quando os *datasets* de treino apresentam dimensões menores.

O aumento da complexidade em demasia numa rede neuronal que contém poucos dados de treino, resulta na complicação da análise desses dados, gerando aprendizagem de ruído. Para que tal não aconteça, foi criada a metodologia designada de *Dropout*, em que algumas ligações entre unidades é desfeita pela própria rede neuronal durante o treino.

A Figura 3.8 representa a mesma rede neuronal com e sem o efeito da camada de *Dropout*. À esquerda, é possível verificar que são estabelecidas várias conexões entre unidades, que poderá levar ao fenómeno de *overfitting*, enquanto que a aplicação da camada de *Dropout* simplifica essas ligações, detetando níveis de complexidade desnecessários para a interpretação dos dados de treino.

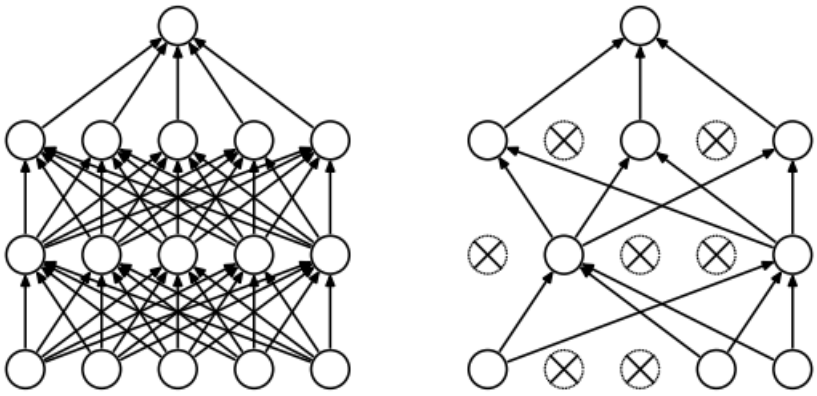


Figura 3.8: Comparação da Utilização de uma Camada de *Dropout* [56].

3.2 Métricas de Avaliação da Precisão de Redes Neuronais

As métricas de precisão são utilizadas na fase de treino e teste das redes neuronais para avaliar o seu desempenho. Essas medidas indicam a capacidade de a rede prever corretamente a saída. Algumas das métricas de precisão comuns são [57, 58]:

***Accuracy*:** calculada através do quociente entre os exemplos classificados corretamente (Verdadeiros Positivos - TP e Verdadeiros Negativos - TN), relativamente ao universo de classificações (Verdadeiros Positivos - TP, Verdadeiros Negativos - TN, Falsos Positivos - FP e Falsos Negativos - FN) (equação (3.1)).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

Usando como exemplo a classificação de cães e gatos, se, em 100 amostras, a rede previr corretamente 80 animais (TP e TN), pode-se concluir que a *accuracy* da rede é de $\frac{80}{100} = 80\,\%$.

***Precision*:** calculada através do quociente entre os exemplos classificados como positivos e os que são realmente positivos (equação (3.2)).

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

No exemplo de classificação de cães e gatos, assumindo que o valor positivo é cão e o negativo é gato, se uma rede neuronal identifica 100 cães, mas apenas 90 (TP) são realmente esse animal, a métrica *precision* é de $\frac{90}{90+10} = 90\,\%$.

***Recall*:** calculada através da razão entre a proporção de exemplos positivos que foram classificados corretamente (equação (3.3)).

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

Por exemplo, se uma rede identificar corretamente 70 cães (TP) e identificar incorretamente 30 gatos (FN), o *recall* é de $\frac{70}{70+30} = 70\,\%$.

***F1-Score*:** dado pelo valor médio entre a precisão (*precision*) e o *recall* (equação (3.4)). Por esse motivo, esta métrica combina as duas métricas anteriores, sendo mais utilizada quando as classes são desequilibradas, ou seja, quando uma classe tem muito mais amostras do que a outra.

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3.4}$$

Matriz de Confusão: Como verificado anteriormente, os valores que se podem obter acerca do desempenho de uma rede neuronal são sempre representados sob a forma de probabilidades (valor escalar). Um método de se obter dados mais concretos e efetivos acerca do desempenho da rede é através da Matriz de Confusão.

Uma Matriz de Confusão é uma representação que permite avaliar se determinada previsão é, de facto, verdadeira ou não. Mantendo o exemplo da classificação de cães e gatos, é possível gerar uma matriz 2×2 semelhante à da Figura 3.9. Analisando cada hipótese, pode-se afirmar que existem quatro combinações possíveis [59]:

- **Verdadeiro Positivo:** indica que a previsão foi positiva e verdadeira (ex: a rede identifica que a imagem contém um cão e, de facto, a imagem representa um cão).

- **Verdadeiro Negativo:** indica que a previsão foi negativa e verdadeira (ex: a rede identifica que a imagem contém um gato e, de facto, a imagem representa um gato).
- **Falso Positivo (Erro de Tipo 1):** indica que a previsão foi positiva e falsa (ex: a rede identifica um cão, mas a imagem contém um gato).
- **Falso Negativo (Erro de Tipo 2):** indica que a previsão foi negativa e falsa (ex: a rede identifica um gato, mas a imagem contém um cão).

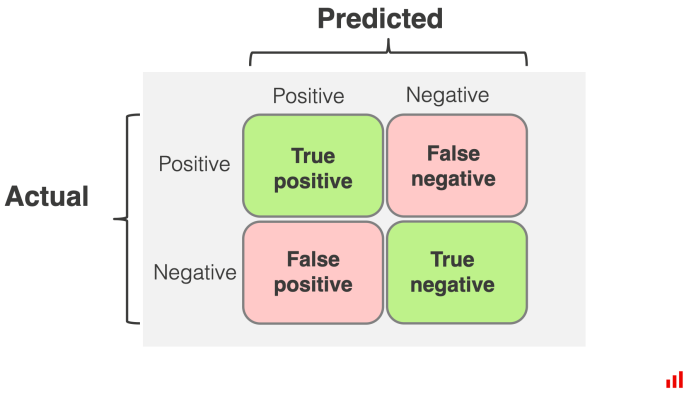


Figura 3.9: Matriz de Confusão Exemplificativa [60].

Resumindo, as métricas são usadas na avaliação do desempenho de uma rede neuronal em contexto de classificação. A *accuracy* é uma métrica geral, pois indica o quão correta é a classificação da rede. A *precision* e o *recall* são métricas mais específicas, que indicam o quão bem a rede neuronal identifica exemplos positivos. O *F1-Score* é uma métrica que combina *precision* e *recall* numa única medida. Além das métricas de desempenho baseadas em probabilidades, destaca-se a Matriz de Confusão que permite analisar dados concretos previstos pela rede neuronal e compará-los com os dados reais, através de uma representação gráfica.

3.3 Otimização de Redes Neuronais

As redes neuronais podem ser sujeitas a determinados algoritmos com o objetivo de as otimizar, tendo em conta os dados disponíveis para o treino e até, facilitando a criação de modelos estruturalmente mais complexos, se tal for necessário.

Esses algoritmos podem ser aplicados a redes pré-treinadas, alterando parte da estrutura da rede e dos próprios *datasets*, de forma a atingir mais facilmente, os objetivos de treino ótimos para determinados dados, mas também podem ser aplicados a redes neuronais treinadas de raiz, otimizando os hiperparâmetros das mesmas.

As técnicas de *fine tuning* permitem a adaptação de redes neuronais pré-treinadas para tarefas específicas ou para a leitura e interpretação mais correta de *datasets*.

Nesses casos, a técnica de *fine tuning* é considerada um mecanismo de *Transfer Learning*, aplicando pequenos ajustes às estruturas e parâmetros internos das redes neuronais [61].

Em alguns tipos de *fine tuning*, a estrutura das redes neuronais mantém-se, em grande parte, durante o processo. Nesses casos, a ideia passa somente por avaliar as *features* mais relevantes para o modelo, quando o *dataset* se torna muito denso e complexo.

As principais razões pelas quais há interesse em utilizar técnicas de *fine tuning* são as seguintes:

- **Eficiência:** o treino de uma rede neuronal de raiz poderá ser extremamente demorado e ineficiente do ponto de vista computacional. Em contrapartida, os algoritmos de *fine tuning* permitem reutilizar modelos pré-treinados, reduzindo significativamente o tempo e os recursos necessários para obter resultados satisfatórios.

- **Desempenho adaptado às necessidades:** os métodos de *fine tuning* permitem uma melhor adaptação das redes neuronais à problemática a resolver. Dessa forma, cria-se um modelo capaz de responder com mais especificidade aos dados apresentados nos *datasets*.

- **Eficiência dos dados:** dada a dificuldade de obter grandes quantidades de dados para criação de *datasets* consistentes para um treino mais eficaz, os *fine tuners* oferecem soluções para treinar os modelos, tendo em conta os dados existentes.

3.3.1 Otimização de Hiperparâmetros

No treino de redes neuronais, os hiperparâmetros assumem um papel crucial, pois é através deles que, partindo dos mesmos dados, se pode variar o desempenho de um modelo, aumentando a sua capacidade de resolver determinados problemas [62].

Entende-se por hiperparâmetros, elementos como:

- ***Learning Rate*:** controla a velocidade de convergência do modelo, sendo que um valor mais elevado contribui para uma convergência mais rápida, enquanto um valor mais baixo contribui para uma convergência mais lenta. As variações bruscas de *Learning Rate*, como o uso de valores muito elevados ou muito baixos contribuem para um *overshoot* da solução ótima para o treino e a convergência demasiado lenta, respetivamente. A Figura 3.10 mostra o efeito da variação do *Learning Rate* no treino de uma rede neuronal.

- **Quantidade de Camadas e Unidades Ocultas:** definem a arquitetura da rede neuronal. A escolha deste hiperparâmetro também tem um elevado

impacto, uma vez que pode influenciar o tempo de treino e a compreensão de padrões mais complexos.

- ***Batch Size*:** determina a quantidade de amostras usadas em cada iteração durante o treino do modelo. Geralmente, um *batch size* menor aumenta a capacidade de generalização da rede, mas também aumenta o tempo de treino.
- **Funções de Ativação:** estas funções impactam no modo como o modelo trata as não linearidades.

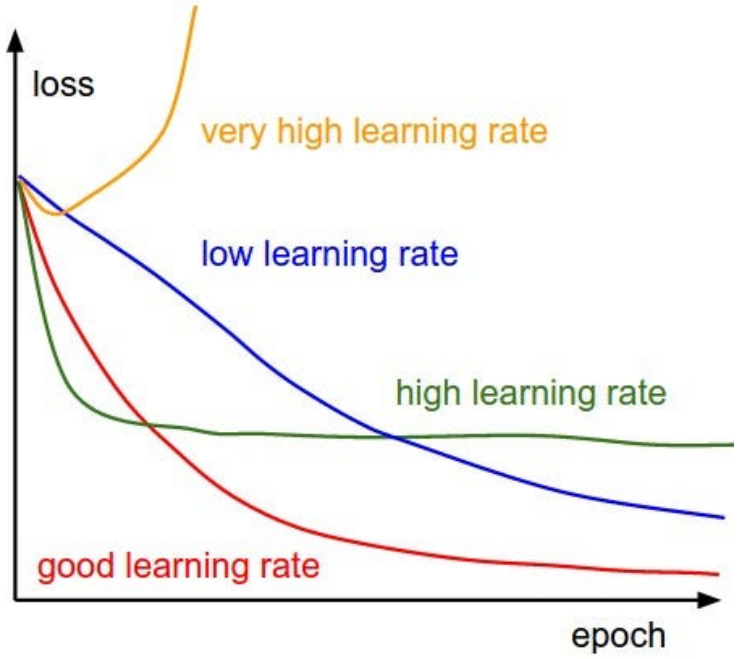


Figura 3.10: Efeitos da Variação do *Learning Rate* [63].

Para efetuar este tipo de otimização, foram criados vários tipos de otimizadores, dos quais se destacam:

- ***Grid Search*:** é um otimizador que envolve a especificação de intervalos de valores para cada hiperparâmetro e testa todas as combinações possíveis. Apesar de ter um elevado consumo computacional, garante que o melhor modelo é encontrado.
- ***Random Search*:** é um otimizador que escolhe os hiperparâmetros aleatoriamente, partindo de valores típicos. Este algoritmo é menos pesado a nível computacional, apesar de conseguir obter valores relativamente semelhantes ao *Grid Search*.
- **Otimização Bayesiana (*Bayesian Optimization*):** recorre a um modelo probabilístico para prever o desempenho de diferentes configurações dos hiperparâmetros. De seguida, seleciona as configurações que, possivelmente, trarão

valores melhores ao modelo. Este otimizador resulta numa técnica mais eficiente e efetiva.

- **Algoritmos Genéticos:** tendo por base o conceito de seleção natural, este otimizador faz evoluir várias populações constituídas por hiperparâmetros, ao longo de várias gerações, sendo que as melhores configurações para o modelo em estudo são as que "sobrevivem" e se reproduzem, passando às gerações seguintes.
- **Otimização de Hiperparâmetros Automatizada:** aplicado em ferramentas da Google (como o AutoML), que oferecem algoritmos de automatização para definir os melhores hiperparâmetros, sem necessidade de configurações manuais.

Capítulo 4

O *Dataset*

Neste capítulo, é detalhada a criação do *dataset* utilizado neste trabalho, denominado de *UCI HAR Dataset*. São também mostradas as imagens relativas à variação de cada sinal obtido dos sensores utilizados, de modo a fornecer uma perspetiva visual do formato desses mesmos dados à saída dos sensores.

4.1 Escolha do *Dataset* para o Sistema de Reconhecimento de Movimentos

Para a execução do projeto proposto, optou-se por efetuar uma pesquisa por *datasets* com informações obtidas por sensores de um telemóvel, de forma a aproveitar uma base de conhecimento existente para treinar as redes neuronais. Desta forma, foi utilizado o *dataset* denominado "*Human Activity Recognition Using Smartphones*" ou simplesmente "UCI HAR *Dataset*", que fora documentado em vários artigos, que transmitem a necessidade da criação de um *dataset* desta natureza, detalhando as etapas de desenvolvimento do mesmo e o respetivo material utilizado [5, 64, 65, 66].

4.2 Contextualização da Criação do *UCI HAR Dataset*

A maioria dos idosos tende a sofrer de doenças como a de *Parkinson*, assim como perda de capacidade visual, levando a que, com o avançar do tempo, se torne cada vez mais difícil para estas pessoas serem independentes e terem controlo sobre a mobilidade [5].

Existem várias aplicações para este tipo de sistemas, tais como a criação de casas inteligentes, através da assistência ao utilizador, outras áreas da saúde e segurança [67, 68, 69].

Apesar dos desenvolvimentos na área de sensorização ambiente, existem limitações quanto à utilização desta tecnologia, pois a precisão das leituras é inferior quando comparadas com sensores que acompanham os movimentos do corpo, além de se tornarem limitadoras, pelo simples facto de que os sensores se encontram parados no mesmo local da casa/espaço frequentado pelos pacientes, não sendo possível acompanharem movimentos fora desses espaços.

A *Human Activity Recognition* (HAR) é o campo de estudo que trata de identificar as ações realizadas por um ou mais sujeitos, através da aquisição de informação com contexto sobre o estado do utilizador e o ambiente envolvente. Para tal, são utilizadas arquiteturas distribuídas de sensores, integrados por recursos computacionais. A maioria dos métodos de sensorização utilizados baseiam-se na acelerometria que, tal como o nome indica, recorre a acelerómetros para adquirir dados relativos a atividades executadas pelos utilizadores [66].

A utilização de sensores próximos do corpo dos pacientes permite melhorar a capacidade de monitorização dos movimentos efetuados, possibilitando medir várias grandezas e sinais, tais como localização, micro movimentos e estado psicológico, mas também, permite uma supervisão mais portátil do paciente. Apesar destas vantagens, a tecnologia ainda não evoluiu o suficiente para deixar alguns pacientes menos desconfortáveis com a utilização deste tipo de sistemas [66].

Segundo o descrito no artigo "A Public Domain Dataset for Human Activity Recognition Using Smartphones" [66], os dados constituintes do *Dataset* foram obtidos a partir da leitura de dados por parte de sensores presentes no *Smartphone Samsung Galaxy S2*, colocado na cintura de 30 voluntários, com idades compreendidas entre os 19 e os 48 anos. As tarefas a monitorizar são:

- Estar de pé;
- Estar sentado;
- Estar deitado;
- Andar;
- Subir/Descer Escadas.

Para obter tais dados, os sensores usados são o acelerómetro e o giroscópio que, utilizando uma frequência de amostragem de 50 Hz, permitiram obter a aceleração triaxial e a velocidade angular.

4.2.1 Primeiros Testes ao *Dataset* Desenvolvido

Segundo informações presentes nos artigos [5, 65, 66], os testes efetuados com o *dataset* permitiram obter uma precisão de 96 %, partindo de 2947 padrões de teste. De forma a avaliar se o *dataset* poderia ter melhor precisão, o acelerómetro triaxial foi montado na cintura dos voluntários, mostrando que a precisão descia para 90.8 %. O mesmo foi também montado no pescoço dos utilizadores, mostrando que a precisão aumentava para 93.9 % estando, ainda assim, abaixo do valor inicial.

4.2.2 Conclusões Obtidas pela Equipa de Desenvolvimento do *Dataset*

No trabalho desenvolvido para a criação do *dataset*, o foco foi da aprendizagem supervisionada, baseada na utilização de sensores inerciais facilmente colocados no corpo do utilizador segundo referido no artigo "Human Activity Recognition on Smartphones using a Multiclass Hardware-Friendly Support Vector Machine" [64].

A utilização de sensores portáteis presentes nos *smartphones* permitem uma sensibilidade aprimorada, dada a proximidade dos sensores embebidos e o corpo, aliado ao poder de processamento dos dispositivos móveis nos dias que correm.

As *Support Vector Machines* (SVM) foram escolhidas para esta aplicação pois promovem uma boa precisão para um tempo de treino menor, aliado à disponibilidade de várias *frameworks* gratuitas para trabalhar este tipo de dados, como a LIBSVM [70]. A desvantagem do uso desta tecnologia prende-se com a fraca capacidade de generalização em certos casos, não se prevendo tal evento nesta aplicação [5, 64, 65, 66].

Um problema que poderá surgir da utilização de SVM, é a elevada exigência aplicada à bateria do *smartphone* para efetuar cálculos de vírgula flutuante. Por exigir alguns recursos computacionais, pode drená-la.

Os dados provenientes dos sensores utilizados têm o aspeto da Figura 4.1.

4.3 Formato do *Dataset*

Os ficheiros de *dataset* fornecidos são designados por `X_train.txt`, `X_test.txt`, `y_train.txt`, `y_test.txt`, representando os dados de treino e teste das entradas e saídas, respetivamente. Todos os ficheiros estão num formato semelhante ao *Comma-separated Values* (CSV), sendo separado por espaços e sem incluir *headers*. Os mesmos são fornecidos em ficheiros separados (`features.txt`, que correspondem aos *headers* dos *datasets* de entrada (Secção A.1), e `activity_labels.txt`, que correspondem ao mapeamento entre as saídas e o respetivo movimento (Tabela 4.2)). Os ficheiros que compõem o *dataset* ocupam um espaço na memória próximo dos 80 MB.

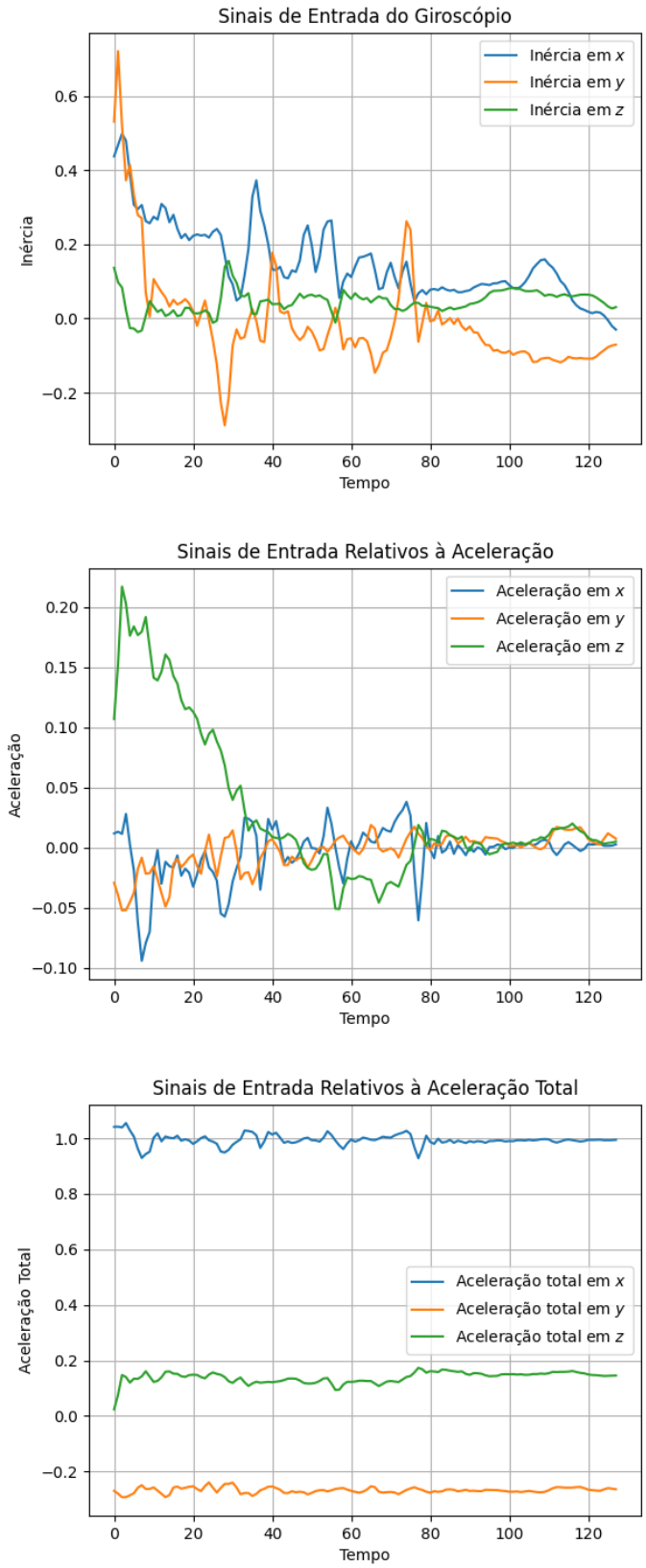


Figura 4.1: Sinais de Entrada Obtidos dos Sensores.

O *dataset* de entrada (Figura 4.2a) é constituído por um ficheiro com 561 colunas, que representam as *features* e, assim como os *datasets* de treino e validação, 7352 linhas. Já o *dataset* de saída (Figura 4.2b) é constituído por apenas uma coluna, que indica um de 6 movimentos possíveis de distinguir e a mesma quantidade de linhas que o *dataset* de entrada.

[9]:

	0	1	2	3	4	5	6	7	8	9	...	551	552	553	
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	-0.074323	-0.298676	-0.710304	-0.1
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	0.158075	-0.595051	-0.861499	0.0
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	0.414503	-0.390748	-0.760104	-0.1
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692	...	0.404573	-0.117290	-0.482845	-0.0
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469	...	0.087753	-0.351471	-0.699205	0.1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7347	0.299665	-0.057193	-0.181233	-0.195387	0.039905	0.077078	-0.282301	0.043616	0.060410	0.210795	...	-0.070157	-0.588433	-0.880324	-0.1
7348	0.273853	-0.007749	-0.147468	-0.235309	0.004816	0.059280	-0.322552	-0.029456	0.080585	0.117440	...	0.165259	-0.390738	-0.680744	0.0
7349	0.273387	-0.017011	-0.045022	-0.218218	-0.103822	0.274533	-0.304515	-0.098913	0.332584	0.043999	...	0.195034	0.025145	-0.304029	0.0
7350	0.289654	-0.018843	-0.158281	-0.219139	-0.111412	0.268893	-0.310487	-0.068200	0.319473	0.101702	...	0.013865	0.063907	-0.344314	-0.1
7351	0.351503	-0.012423	-0.203867	-0.269270	-0.087212	0.177404	-0.377404	-0.038678	0.229430	0.269013	...	-0.058402	-0.387052	-0.740738	-0.2

7352 rows × 561 columns

(a) *Dataset* de Entrada.

[10]:

	0
0	5
1	5
2	5
3	5
4	5
...	...
7347	2
7348	2
7349	2
7350	2
7351	2

7352 rows × 1 columns

(b) *Dataset* de Saída.

Figura 4.2: *Datasets* de Entrada e de Saída.

4.3.1 *Features* do *Dataset* de Entrada

Os dados obtidos para a criação do *dataset* de entrada são provenientes de sinais em bruto de um acelerómetro (`tAcc-XYZ`) e giroscópio (`tGyro-XYZ`) triaxiais. Estes dados são apresentados no domínio do tempo, sendo, por isso, utilizado o prefixo `t` e foram capturados a uma taxa constante de 50 Hz.

De seguida, estes dados foram subdivididos, dando origem a acelerações corporais (`tBodyAcc-XYZ`) e a acelerações gravíticas (`tGravityAcc-XYZ`).

Por sua vez, os dados corporais anteriores passaram por operações de derivação no tempo, de forma a obter sinais *Jerk*, que representam as taxas de variação das acelerações - `tBodyAccJerk-XYZ` a `tBodyGyroJerk-XYZ`.

Também foram obtidos os valores de magnitude de cada sinal tridimensional, recorrendo à norma Euclidiana, dando origem a mais cinco *features*:

1. `tBodyAccMag`;
2. `tGravityAccMag`;
3. `tBodyAccJerkMag`;
4. `tBodyAccGyroMag`;
5. `tBodyGyroJerkMag`.

Recorrendo à *Fast Fourier Transform* (FFT), foram obtidas as *features* indicadas com o prefixo `f`, tais como:

1. `fBodyAcc-XYZ`;
2. `fBodyAccJerk-XYZ`;
3. `BodyGyro-XYZ`;
4. `fBodyAccJerkMag`;
5. `fBodyGyroMag`;
6. `fBodyGyroJerkMag`.

As restantes *features* foram obtidas através de processos matemáticos aplicados às grandezas anteriores, tais como os que são apresentados na Tabela 4.1.

A Secção A.1 mostra o conjunto de todas as *features* obtidas através dos sinais dos sensores do projeto que criou o *HAR Dataset*.

4.3.2 Possíveis Saídas

A Tabela 4.2 mostra o respetivo mapeamento entre as possíveis saídas de cada rede neuronal. Por uma questão de desempenho computacional, cada saída é representada por um número que, posteriormente, é mapeado ao respetivo movimento.

De forma a avaliar quantas amostras diferentes existem de cada movimento no *dataset* de treino, foi traçado o gráfico da Figura 4.3 e, a partir daí, pode-se afirmar que os movimentos `LAYING`, `STANDING` e `WALKING` são os que mais amostras apresentam, ao contrário de `WALKING_DOWNSTAIRS` e `WALKING_UPSTAIRS`, que têm uma menor quantidade de amostras.

Tabela 4.1: Processos Matemáticos Aplicados aos Dados.

Função	**Descrição**
`mean()`	Valor Médio
`std()`	Desvio Padrão
`mad()`	Desvio Absoluto Mediano
`max()`	Valor Máximo
`min()`	Valor Mínimo
`sma()`	Área de Magnitude do Sinal
`energy()`	Soma dos Quadrados das Energias, Dividido pelo Total de Amostras
`iqr()`	Amplitude Interquartil
`entropy()`	Entropia do Sinal
`arCoeff()`	Coeficientes de autorregressão com ordem de Burg igual a 4
`correlation()`	Coeficiente de Correlação entre Dois Sinais
`maxInds()`	Índice da Componente de Frequência com a Maior Magnitude
`meanFreq()`	Média Pesada das Componentes de Frequência
`skewness()`	Assimetria do Sinal no Domínio das Frequências.
`kurtosis()`	Achatamento da Curva de Distribuição do Sinal no Domínio das Frequências
`bandsEnergy()`	Energia de um Intervalo de Frequência da FFT
`angle()`	Ângulo Entre Dois Vetores

Tabela 4.2: Movimentos de Possível Distinção.

Identificador	**Movimento**
1	WALKING (Andar)
2	WALKING_UPSTAIRS (Subir Escadas)
3	WALKING_DOWNSTAIRS (Descer Escadas)
4	SITTING (Sentar)
5	STANDING (De Pé)
6	LAYING (Deitar)

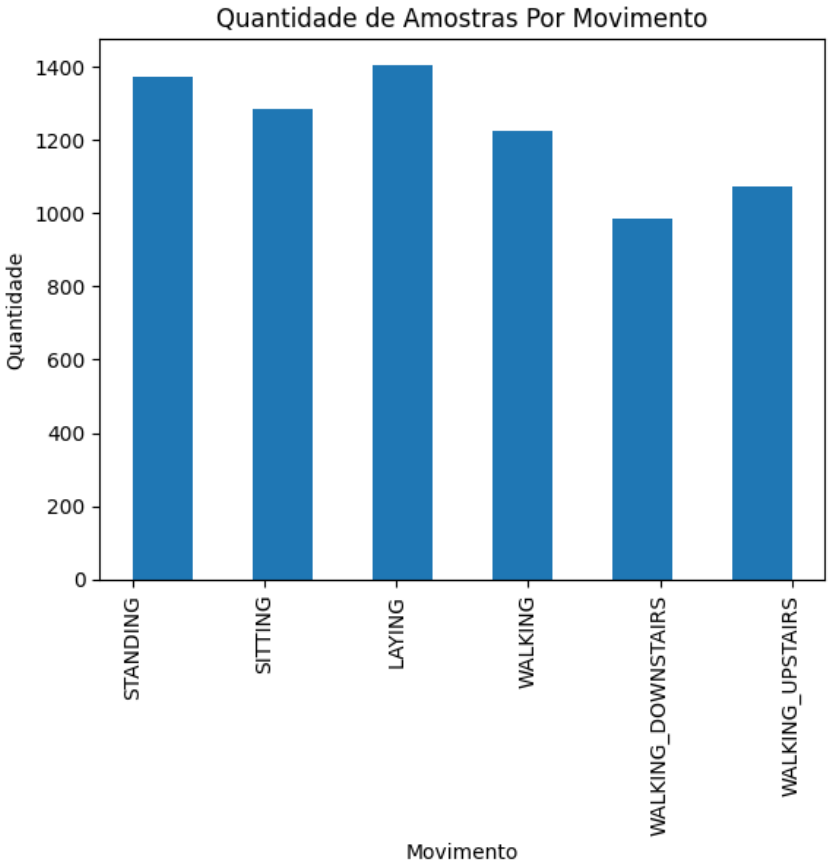


Figura 4.3: Quantidade de Amostras Obtidas por Cada Tipo de Movimento.

Capítulo 5

Implementação

Neste capítulo, é descrito o procedimento adotado para o estudo do treino de quatro redes neuronais com o objetivo de fazer um reconhecimento de movimentos com elevados níveis de precisão e eficácia [1].

5.1 Introdução

Com vista à resolução do problema de reconhecimento de movimentos, foram criadas quatro redes neuronais distintas: uma *Feedforward Neural Network* (FNN), uma rede do tipo *Long Short-Term Memory* (LSTM), uma rede *Bidirectional Long Short-Term Memory* (Bi-LSTM) e a uma rede do tipo *Gated Recurrent Unit* (GRU). Os modelos criados tiveram por base estruturas típicas utilizadas em cada um dos tipos de rede estudados.

Todas as redes foram treinadas através da linguagem *Python*, utilizando as bibliotecas *Keras*, *Pandas*, *Scikit-Learn*, *Numpy* e *Matplotlib* com recurso ao *dataset* criado por Jorge Reyes-Ortiz, Davide Anguita, Alessandro Ghio, Luca Oneto e Xavier Parra, comummente designado por "*UCI HAR Dataset*" [71]. Este *dataset* já apresenta dados tratados, não sendo necessário pré-processamento dos mesmos.

Uma vez que o *dataset* já apresenta os dados de treino e de teste divididos, a única etapa adicional de tratamento prendeu-se com a divisão do *dataset* de teste para obter dados de validação. Para esse segundo, foram utilizados 10 % dos dados

[1] O código desenvolvido encontra-se disponível em `https://github.com/danilmour/Sistema-de-Reconhecimento-de-Movimentos`.

presentes em `X_test.txt` e `y_test.txt`. A Listagem 5.1 mostra a divisão dos respetivos *datasets* via *Python*. De notar que a função `read_csv` da biblioteca *Pandas* é responsável por carregar os dados do ficheiro como um *Dataframe*, que permite uma manipulação semelhante à de uma tabela de uma base de dados. Dessa forma, é possível executar *queries* para manipular os dados.

```
1  X_train = pd.read_csv("../Dataset/UCI HAR Dataset/UCI HAR Dataset/
       train/X_train.txt", header=None, sep="\s+")
2  y_train = pd.read_csv("../Dataset/UCI HAR Dataset/UCI HAR Dataset/
       train/y_train.txt", header=None)
3
4  X_test_val = pd.read_csv("../Dataset/UCI HAR Dataset/UCI HAR
       Dataset/test/X_test.txt", sep="\s+", header=None)
5  y_test_val = pd.read_csv("../Dataset/UCI HAR Dataset/UCI HAR
       Dataset/test/y_test.txt", header=None)
6
7  X_test = X_test_val[0:round(0.9*X_test_val.shape[0])]
8  X_val = X_test_val[round(0.9*X_test_val.shape[0]):]
9  y_test = y_test_val[0:round(0.9*y_test_val.shape[0])]
10 y_val = y_test_val[round(0.9*y_test_val.shape[0]):]
```

Listagem 5.1: Carregamento dos Ficheiros de *Dataset* e Criação de *Datasets* de Validação.

Para tornar a análise comparativa das redes um pouco mais simples, a técnica de treino foi, dentro do possível, reutilizada, de forma a manter alguma coerência entre os procedimentos. A Figura 5.1 representa um fluxograma geral do método utilizado na criação, treino e teste das redes neuronais.

No tópico relacionado com os hiperparâmetros utilizados no treino das redes, foi utilizado o otimizador `Adam`, que, por defeito, utiliza um *learning rate* fixo de 0.001, com um `beta_1` igual a 0.9 e um `beta_2` igual a 0.999.

5.2 Criação de uma Rede FNN

Utilizando o *dataset* explicitado acima, foi criada uma primeira implementação do reconhecimento de movimentos, com uma rede FNN.

Para esta implementação, foram utilizadas as bibliotecas *Tensorflow* e *Keras* para obter as ferramentas necessárias à criação da rede neuronal.

5.2.1 Criação do Modelo da Rede

Após a preparação dos *dataseFts* conforme descrito na Listagem 5.1, passou-se à criação do modelo correspondente à rede neuronal a estudar.

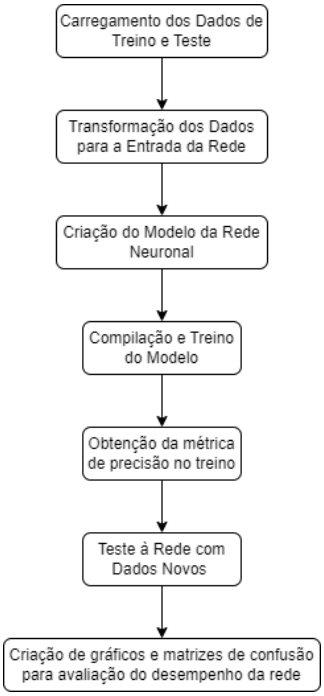


Figura 5.1: Fluxograma da Criação das Redes Neuronais.

Nesta fase, foi criado um modelo sequencial, constituído por três camadas de processamento do tipo *Dense*, confirme ilustra a Figura 5.2.

A primeira camada adicionada ao modelo apresenta 576 neurónios e é ativada pela função *ReLU*. A segunda camada é constituída por 9 neurónios, tendo a mesma função de ativação. Finalmente, a terceira camada apresenta 7 neurónios e a ativação é dada pela função *softmax*. Esta última camada é usada para a classificação das várias classes, dado que a saída será a probabilidade de cada um dos movimentos categorizados.

A Listagem 5.2 evidencia as linhas de código que permitiram criar a rede FNN.

```
1 modelo = Sequential()
2 modelo.add(Dense(576, activation="relu"))
3 modelo.add(Dense(9, activation="relu"))
4 modelo.add(Dense(7, activation="softmax"))
```

Listagem 5.2: Criação do Modelo da Rede FNN.

5.2.2 Compilação do Modelo

A compilação recorre à função de perda `binary_crossentropy` e ao otimizador `Adam`. Uma vez que podem ser avaliadas várias métricas ao longo do treino, as selecionadas para avaliar a precisão são as referidas na Secção 3.2. O código da Listagem 5.3 permitiu efetuar o treino da rede com os parâmetros indicados.

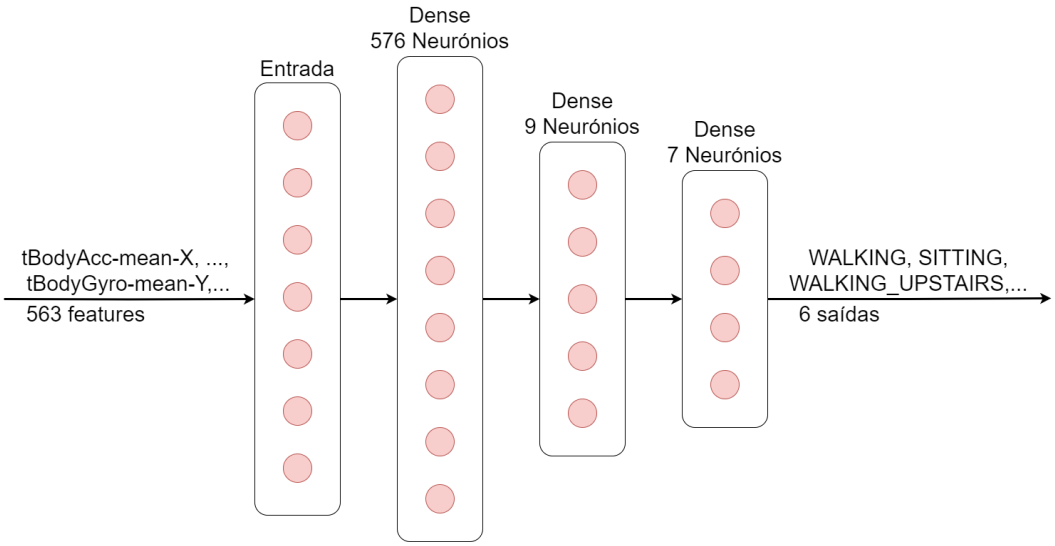


Figura 5.2: Formato da Rede FNN.

```
1 modelo.compile(loss="binary_crossentropy", optimizer="adam",
      metrics=["accuracy", "Precision", "Recall"])
```

Listagem 5.3: Compilação e Treino do Modelo da Rede FNN.

```
1 y_treino_cat = to_categorical(y_train)
2 y_teste_cat = to_categorical(y_test)
3 y_val_cat = to_categorical(y_val)
4
5 historico = modelo.fit(X_train, y_treino_cat, epochs=20,
      batch_size=10, validation_data=(X_val, y_val_cat))
6 modelo.save("ModeloFNN.keras")
```

Listagem 5.4: Formatação dos *Datasets* de Saída e Posterior Treino da Rede FNN.

O método `fit` é responsável por iniciar o treino da rede neuronal definida pelo modelo, sendo necessário passar como parâmetro os dados de treino (`X_train`) e as saídas correspondentes (`y_treino_cat`). Os *Dataframes* de saída inseridos neste método diferem no formato geral, dado que o treino necessita das saídas num formato de matriz binária, que é obtido através da função `to_categorical`.

O treino é realizado ao longo de 20 épocas, com um *batch size* de 10 amostras de cada vez, sendo feita a validação com os dados provenientes de `X_val` e `y_val_cat`, como é possível visualizar na Listagem 5.4. À medida que o treino é cumprido, os dados históricos vão sendo armazenados na variável `historico`. Por fim, o modelo é guardado num ficheiro designado `ModeloFNN.keras`.

5.3 Criação de uma Rede LSTM

Outra rede comummente utilizada neste tipo de dados é a rede LSTM. Para dados em formato de texto, como é o caso, este tipo de rede tende a ter um melhor desempenho que a rede FNN.

No que toca ao carregamento dos *datasets* de entrada, estes foram carregados num formato diferente do da rede FNN, para corresponder ao formato de entrada e saída específico para redes LSTM, que é uma matriz tridimensional constituída por amostras, *timesteps* e *features*. Para isso, foi utilizado o método `values.reshape` (Listagem 5.5).

Da mesma forma que na rede anterior, os *datasets* de saída foram transformados numa representação categórica (no formato de uma matriz binária). Isso é necessário para o treino da rede, na medida em que a camada de saída possui a função de ativação `softmax`.

```
1 X_treino = X_treino.values.reshape((X_treino.shape[0], 1, X_treino
      .shape[1]))
2 X_teste = X_teste.values.reshape((X_teste.shape[0], 1, X_teste.
      shape[1]))
3 X_val = X_val.values.reshape((X_val.shape[0], 1, X_val.shape[1]))
4
5 y_treino_cat = to_categorical(y_treino)
6 y_teste_cat = to_categorical(y_teste)
7 y_val = to_categorical(y_val)
```

Listagem 5.5: Transformação dos *Datasets* para Redes Neuronais Recorrentes.

5.3.1 Criação do Modelo da Rede

À semelhança da rede FNN, a arquitetura da rede foi definida utilizando o objeto `Sequential`, que indica que a rede em causa é constituída por camadas sequenciais.

```
1 modelo = Sequential()
2 modelo.add(LSTM(561, input_shape=(X_treino.shape[1], X_treino.
      shape[2])))
3 modelo.add(Dense(9, activation="relu"))
4 modelo.add(Dense(7, activation="softmax"))
```

Listagem 5.6: Criação da Estrutura da Rede LSTM.

A primeira camada é do tipo LSTM, sendo constituída por 561 unidades, e o `input_shape` é definido para corresponder ao formato dos dados de entrada. As duas camadas seguintes são do tipo *Dense*, tendo a primeira 9 neurónios ativados pela

função `relu` e a segunda camada 7 neurónios ativados através da função `softmax`. A última é a camada de saída, que distingue os diferentes tipos de movimento.

A Listagem 5.6 e a Figura 5.3 mostram as linhas de código que permitiram criar o modelo da rede e a especificação da mesma, respetivamente.

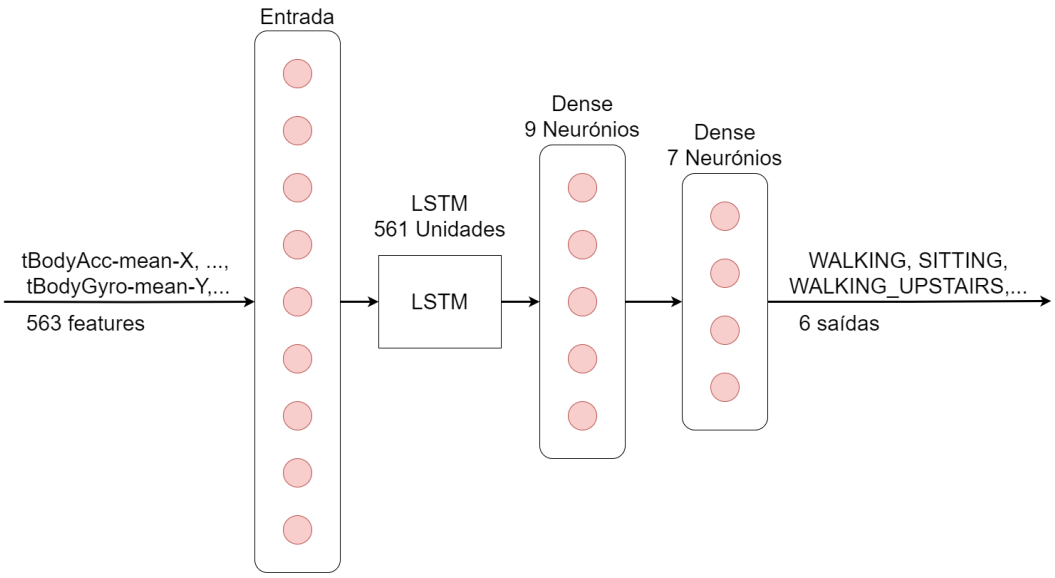


Figura 5.3: Formato da Rede LSTM.

5.3.2 Compilação do Modelo

À semelhança da rede FNN, a compilação foi efetuada através da função de perda `binary_crossentropy` e o otimizador `Adam`. Foram também monitorizadas as métricas da Secção 3.2 e os dados de validação são `X_val` e `y_val`.

```
1 modelo.compile(loss="binary_crossentropy", optimizer="adam",
      metrics=["accuracy", "Precision", "Recall"])
2
3 historico = modelo.fit(X_treino, y_treino_cat, epochs=20,
      batch_size=10, validation_data=(X_val, y_val))
4 modelo.save("ModeloLSTM.keras")
```

Listagem 5.7: Compilação e Treino do Modelo da Rede LSTM.

Após a compilação, iniciou-se o treino da rede LSTM. O modelo é treinado ao longo de 20 épocas, com um *batch size* igual a 10 amostras de cada vez. O histórico de treino e validação é armazenado na variável `historico` e a rede treinada é guardada no ficheiro `ModeloLSTM.keras`.

A Listagem 5.7 reflete as fases de compilação e treino da rede LSTM.

5.4 Criação de uma Rede Bi-LSTM

A terceira rede implementada trata-se de uma Bi-LSTM. Como foi referido anteriormente, este é um tipo de rede particularmente interessante de implementar, dado que se trata de um *redesign* da rede LSTM, acrescendo de camadas bidirecionais, podendo resultar num melhor desempenho a nível geral.

Sendo esta rede uma implementação da comum LSTM, o formato dos *datasets* teve de ser alterado da mesma forma que na Listagem 5.5.

5.4.1 Criação do Modelo da Rede

O modelo da rede Bi-LSTM é constituído por cinco camadas: a primeira camada é do tipo *Bidirectional*, que implementa uma rede LSTM, sendo constituída por 64 unidades, e com o redirecionamento das saídas anteriores para a entrada seguinte (`return_sequences=True`); de seguida, é adicionada uma camada de *Dropout*, com um valor de 0,5, que garante que não haja um decaimento da precisão superior a 50 % durante o treino; a camada subsequente é também uma *Bidirectional*, que implementa uma LSTM que, desta vez, não retorna as saídas anteriores para a entrada seguinte; segue-se outra camada de *Dropout* com um valor de 0.5, finalizando com uma camada *Dense* com 7 neurónios, que indica a saída da rede neuronal. Neste caso, a camada de saída é ativada pela função `sigmoid`. A Listagem 5.8 e a Figura 5.4 ilustram a estrutura descrita.

```
1 modelo = Sequential()
2 modelo.add(Bidirectional(LSTM(64, return_sequences=True)))
3 modelo.add(Dropout(0.5))
4 modelo.add(Bidirectional(LSTM(64)))
5 modelo.add(Dropout(0.5))
6 modelo.add(Dense(7, activation="sigmoid"))
```

Listagem 5.8: Criação da Estrutura da Rede Bi-LSTM.

5.4.2 Compilação do Modelo

Seguindo a mesma estratégia das redes anteriores, a compilação foi efetuada através da função de perda `binary_crossentropy` e o otimizador `Adam`. As métricas selecionadas são também as referidas na Secção 3.2.

O treino da rede Bi-LSTM foi configurado para 20 épocas, com um *batch size* igual a 10 amostras de cada vez e com os dados de validação referentes às variáveis `X_val` e `y_val`. Mais uma vez, o histórico foi guardado na variável `historico` e o modelo treinado, guardado com o nome `ModeloBiLSTM.keras`.

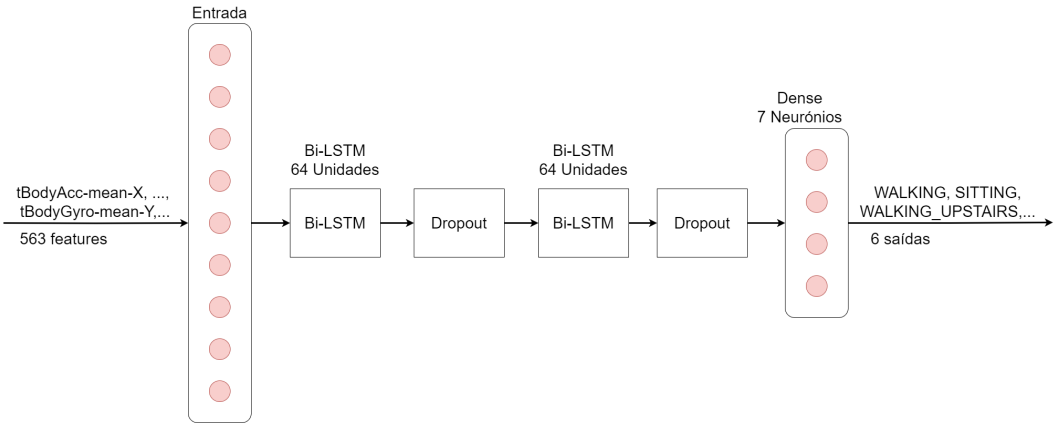


Figura 5.4: Formato da Rede Bi-LSTM.

A Listagem 5.9 representa o código utilizado na compilação e treino da rede Bi-LSTM.

```
1 modelo.compile(loss="binary_crossentropy", optimizer="adam",
      metrics=["accuracy", "Precision", "Recall"])
2
3 historico = modelo.fit(X_train, y_train, epochs=20, batch_size=10,
       validation_data=(X_val, y_val))
4 modelo.save("ModeloBiLSTM.keras")
```

Listagem 5.9: Compilação e Treino do Modelo da Rede Bi-LSTM.

5.5 Criação de uma Rede GRU

A quarta implementação é relativa à implementação de uma rede GRU. A possível vantagem teórica da utilização desta rede prende-se com o facto de ser mais recente do que a rede LSTM, tendo sido sujeita a critérios mais atuais, no que toca ao consumo de recursos computacionais e à otimização do processo de treino.

Tendo uma estrutura interna semelhante à rede LSTM no que toca ao formato de dados de entrada e de saída, os mesmos tiveram que sofrer a transformação evidenciada na Listagem 5.5.

5.5.1 Criação do Modelo da Rede

A rede GRU criada é constituída por três camadas: a primeira, que é uma camada do tipo GRU, constituída por 128 unidades; uma segunda camada de *Batch Normalization*, que permite normalizar os valores de entrada da camada seguinte, aumentando consideravelmente a velocidade de treino, sem impactar na precisão final; a última

camada é do tipo *Dense* e é composta por 7 neurónios, sendo ativada pela função `softmax`. A Listagem 5.10 e a Figura 5.5 ilustram a descrição do modelo da rede.

```
1 modelo = Sequential()
2 modelo.add(GRU(128))
3 modelo.add(BatchNormalization())
4 modelo.add(Dense(7, activation="softmax"))
```

Listagem 5.10: Criação da Estrutura da Rede GRU.

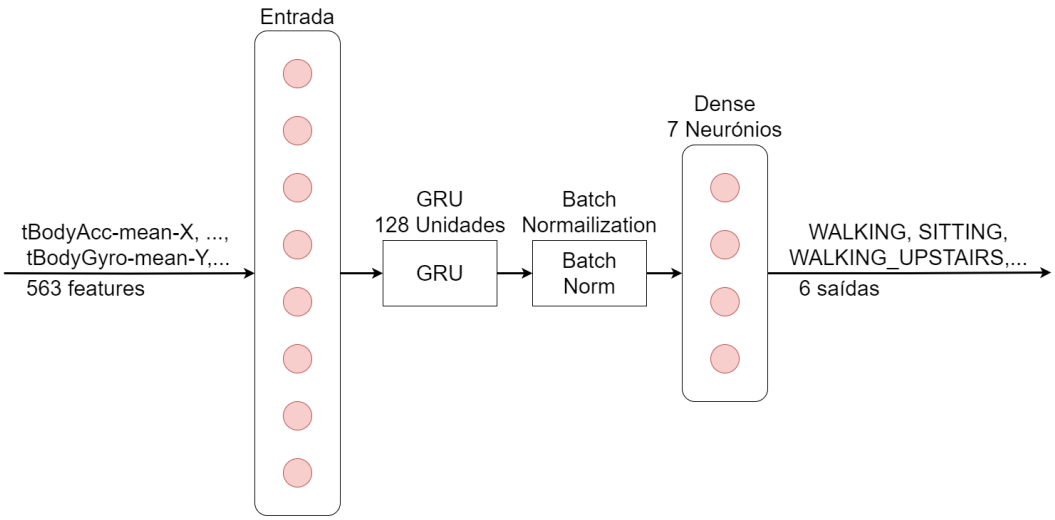


Figura 5.5: Formato da Rede GRU.

5.5.2 Compilação do Modelo

Finalizando a etapa de criação da rede GRU, inicia-se a fase de compilação e treino do modelo. Para tal, à semelhança das restantes redes, foi feita a compilação com base na função de perda `binary_crossentropy` e no otimizador `Adam`. As métricas monitorizadas são as da Secção 3.2.

```
1 modelo.compile(loss="binary_crossentropy", optimizer="adam",
      metrics=["accuracy", "Precision", "Recall"])
2
3 historico = modelo.fit(X_train, y_train, batch_size=10, epochs=20,
       validation_data=(X_val, y_val))
4 modelo.save("ModeloGruLSTM.keras")
```

Listagem 5.11: Compilação e Treino do Modelo da Rede GRU.

O treino foi efetuado ao longo de 20 épocas, com um *batch size* igual a 10 amostras de cada vez. Os dados de validação são os das variáveis `X_val` e `y_val`. O histórico

de treino foi armazenado na variável `historico` e, após o mesmo, o modelo foi guardado no ficheiro `ModeloGruLSTM.keras`, como indica a Listagem 5.11.

5.6 Melhorias dos Modelos

Após concluir a fase de criação e treino das redes neuronais apresentadas, decidiu-se aplicar melhorias a cada rede, de forma a otimizar o desempenho das mesmas, no que toca à precisão. Para tal, foram aplicadas duas estratégias:

- Aumento do número de camadas e de neurónios (ou unidades) associados às redes neuronais;
- Otimização Bayesiana, utilizado como método mais eficiente para obter melhores valores de precisão de um modo mais automatizado.

5.6.1 Aumento de Camadas e Neurónios

Nesta fase, o principal objetivo era avaliar se o aumento do número de camadas e de neurónios (ou unidades) de uma rede neuronal contribui significativamente para uma melhoria da precisão. Para tal, as redes neuronais foram alteradas segundo a Tabela 5.1.

Tabela 5.1: Alterações Efetuadas às Redes Neuronais.

Rede	**Estrutura Incial**	**Estrutura Alterada**
FNN	Dense (576 Neurónios), Dense (9 Neurónios) e Dense (7 Neurónios)	Dense (736 Neurónios), Dense (368 Neurónios), Dense(123 Neurónios), Dense(31 Neurónios) e Dense (7 Neurónios)
LSTM	LSTM (561 Unidades), Dense (9 Neurónios) e Dense (7 Neurónios)	LSTM (128 Unidades), Dropout (50 %), LSTM (128 Unidades), Dropout (50 %), Flatten, Dense (64 Neurónios) e Dense (7 Neurónios)
Bi-LSTM	Bi-LSTM (64 Unidades), Dropout (50 %), Bi-LSTM (64 Unidades), Dropout (50 %) e Dense (7 Neurónios)	Bi-LSTM (563 Unidades), Dropout (50 %), Bi-LSTM (563 Unidades), Dropout (50 %), Batch Normalization, Dropout (50 %) e Dense (7 Neurónios)
GRU	GRU (128 Unidades), Batch Normalization e Dense (7 Neurónios)	GRU (128 Unidades), GRU (64 Unidades), Batch Normalization e Dense (7 Neurónios)

A fase de treino recorreu aos mesmos parâmetros previamente utilizados, pelo que o código utilizado é semelhante ao mostrado nas secções anteriores.

5.6.2 Otimização Bayesiana

Tendo em conta os resultados obtidos com o aumento de camadas e de neurónios, decidiu-se executar a otimização Bayesiana em todas as redes neuronais, permitindo avaliar a relevância que alguns hiperparâmetros têm para a melhoria de precisão e até do tempo de treino das redes neuronais.

Uma vez que, como referido no Capítulo 6, a rede GRU obteve melhores resultados após o aumento de camadas e unidades, a otimização Bayesiana foi aplicada à segunda versão da rede neuronal.

Uma vantagem da utilização deste método é a indicação de quais os parâmetros mais ajustados para obter uma estrutura e um treino adequados para o tipo de dados que o *dataset* contém, assim como o valor mais elevado de precisão obtido nas diferentes iterações, que corresponde ao modelo proposto.

Com vista a obter o código necessário para a otimização, foi consultada a documentação do *Keras* [72], tendo sido posteriormente, desenvolvida a solução, que é descrita abaixo (Listagens 5.12, 5.13, 5.14 e 5.15).

Inicialmente, é necessário definir os *datasets* a utilizar para o treino e teste das redes neuronais. O código utilizado é semelhante ao utilizado na Listagem 5.1.

De seguida, deve ser definido um método que tenha como parâmetro, a variável `hp`, que representa o conjunto de hiperparâmetros possíveis de gerar durante as otimizações. Visualizando a Listagem 5.12, verifica-se que esta variável é utilizada na definição dos neurónios das camadas densas, no caso da rede FNN.

```
1 def build_model(hp):
2     modelo = Sequential()
3     modelo.add(Dense(hp.Int("neurons_dense_1", min_value=1,
          max_value=600), activation=hp.Choice("activation_dense_1",
          ["relu", "sigmoid"])))
4     modelo.add(Dense(hp.Int("neurons_dense_2", min_value=1,
          max_value=600), activation=hp.Choice("activation_dense_2",
          ["relu", "sigmoid"])))
5     modelo.add(Dense(7, activation="softmax"))
6     modelo.compile(loss=hp.Choice("loss", ["binary_crossentropy",
          "sparse_categorical_crossentropy"]), optimizer=Adam(
          learning_rate=hp.Choice("learning_rate", [0.0001, 0.001,
          0.01])), metrics=["accuracy"])
7     return modelo
```

Listagem 5.12: Método de Definição do Modelo e dos Hiperparâmetros.

A utilização desta variável tem o seguinte intuito:

- `hp.Int(x, min_value=y, max_value=z)` permite definir um hiperparâmetro `x`, compreendido entre os valores inteiros `y` e `z`;

- `hp.Float(x, min_value=y, max_value=z)` permite definir um hiperparâmetro `x`, compreendido entre os valores com vírgula flutuante `y` e `z`;
- `hp.Choice(x, [y, z])` permite definir um hiperparâmetro `x`, que pode tomar os valores `y` ou `z`.

Este tipo de definição permite que o *tuner* faça o treino do mesmo modelo, variando os valores definidos nos hiperparâmetros em cada iteração.

Para todas as otimizações, foram feitas as configurações da Tabela 5.2.

Tabela 5.2: Configurações dos Hiperparâmetros.

Configurações	**Intervalos**
Neurónios/Unidades	[1, 600]
Valor de *Dropout*	[0.001, 0.5]
Funções de Perda	`binary_crossentropy` e `sparse_categorical_crossentropy`
Funções de Ativação *Dense*	`relu` e `sigmoid`
Learning Rate	0.0001, 0.001, e 0.01

Após a definição dos parâmetros variáveis durante a otimização, o método retorna o modelo criado.

Após a etapa de criação do modelo, é inicializada a otimização, como se pode ver na Listagem 5.13.

```
1 tuner = BayesianOptimization(hypermodel=build_model, objective='
    accuracy')
2 tuner.search(X_train, y_train, validation_data=(X_test, y_test),
    epochs=20, batch_size=10, callbacks=[EarlyStopping(patience=2)
    ])
```

Listagem 5.13: Inicialização da Otimização Bayesiana.

É, portanto, criado um `tuner`, cuja natureza é Bayesiana, e onde é definido o modelo a otimizar e o objetivo principal de otimização. No caso, são passados por parâmetro, o método de construção do modelo e a *accuracy*, uma vez que é esta a principal métrica a melhorar. Um parâmetro que foi omitido é o número de iterações que o *tuner* vai fazer e que, por omissão, é igual a 10.

Para iniciar a otimização, é utilizado o método `search`, onde são definidos os parâmetros de treino, que, no caso, são os mesmos utilizados nos treinos iniciais. A única diferença encontra-se na definição da *callback* `EarlyStopping()`, que permite ao otimizador, interromper um treino caso a rede tenha problemas de convergência. Isto permite poupar algum tempo, caso haja uma combinação de parâmetros menos eficaz.

Após as 10 iterações, são visualizados os hiperparâmetros ideais encontrados pelo *tuner*, assim como o modelo que obteve a melhor precisão. Para tal, foi desenvolvido o código das Listagens 5.14 e 5.15, respetivamente.

```
hyperparameters = tuner.get_best_hyperparameters()[0]
print(f"Units LSTM 1: {hyperparameters.get('units_lstm_1')}")
print(f"Dropout 1: {hyperparameters.get('dropout_1')}")
print(f"Units LSTM 2: {hyperparameters.get('units_lstm_2')}")
print(f"Dropout 2: {hyperparameters.get('dropout_2')}")
print(f"Learning Rate: {hyperparameters.get('learning_rate')}")
```

Listagem 5.14: Obtenção dos Hiperparâmetros Ideais.

```
best_model = tuner.get_best_models()[0]
best_model.summary()
```

Listagem 5.15: Obtenção do Melhor Modelo.

Capítulo 6

Resultados

Neste capítulo, são apresentados os resultados relacionados com os treinos realizados às redes neuronais e descritos no Capítulo 5. O objetivo será abordar os pontos fulcrais para a escolha do modelo com melhor capacidade de adaptação ao *dataset* de reconhecimento de movimentos.

6.1 Introdução

Após o treino de cada rede neuronal, a variável `historico` armazena vários dados temporais, que permitem inferir como decorreu o treino e a validação com os dados fornecidos. A partir desses dados, e recorrendo à biblioteca *Matplotlib*, é possível traçar um gráfico que mostra a evolução dos parâmetros de cada rede. A Listagem 6.1 ilustra como foram criados os gráficos da evolução da precisão e validação presentes neste capítulo.

No que toca às métricas referidas na Secção 3.2, o seu valor é obtido através da chamada ao método `evaluate`, associado ao objeto do modelo (Listagem 6.2). A este método, são passados como parâmetro os *datasets* de teste.

Finalmente, para obter as matrizes de confusão para cada modelo, foi usado o código presente na Listagem 6.3. Para tal, são usadas as bibliotecas *Scikit-learn*, *Matplotlib* e *Numpy*.

Primeiramente, são feitas todas as previsões associadas ao *dataset* de saída, sendo guardados esses valores num vetor (`y_previsto`). De forma a obter apenas as classes com maior probabilidade, que indicam o movimento previsto pela rede, é percorrido

esse vetor, selecionando os elementos que possuem a probabilidade mais elevada (`np.argmax`). O mesmo é feito para as saídas reais, obtendo dois vetores com os movimentos previstos e reais para cada caso de teste.

```
plt.plot(historico.history["accuracy"])
plt.plot(historico.history["val_accuracy"])
plt.title("Evolucao da Precisao/Validacao")
plt.ylabel("Precisao")
plt.xlabel("Epoca")
plt.legend(["Precisao", "Validacao"])
plt.grid()
plt.savefig("Evolucao Precisao Validacao", bbox_inches="tight")
plt.show()
```

Listagem 6.1: Código de Geração de Gráficos de Evolução da Precisão e Validação ao Longo do Treino.

```
_, precisao, precision, recall = modelo.evaluate(X_test,
    y_teste_cat)
f1_score = (2 * precision * recall) / (precision + recall)
```

Listagem 6.2: Código para Obtenção das Métricas.

De seguida, é criado um objeto que representa a matriz de confusão, onde são passados os valores reais e os valores previstos pela rede em causa. Finalmente, recorrendo à função `ConfusionMatrixDisplay`, é possível indicar qual o objeto da matriz de confusão e quais os nomes a aplicar aos diferentes movimentos a classificar.

```
y_previsto = modelo.predict(X_test)
y_previsto = [np.argmax(y) for y in y_previsto]

y_real = [np.argmax(y) for y in y_teste_cat]

cm = confusion_matrix(y_real, y_previsto)
display = ConfusionMatrixDisplay(cm, display_labels=labels[1])

display.plot(xticks_rotation="vertical")
display.figure_.savefig("Matriz de Confusao FNN", bbox_inches="
    tight")
```

Listagem 6.3: Código para Obtenção das Matrizes de Confusão.

Através das métricas obtidas, é possível avaliar as diferentes redes neuronais, a fim de indicar a mais eficiente para resolver a problemática do reconhecimento de movimentos.

6.2 *Setup* Utilizado para os Treinos

A ferramenta que permitiu o treino das redes neuronais, baseou-se num computador portátil, da marca *MSI*, com um processador *Intel Core i7* de décima segunda geração (i7-12700H), 16 GB de memória RAM e uma placa gráfica *NVIDIA GeForce RTX 3060*.

Através deste *hardware*, foi possível efetuar os treinos em tempos mais reduzidos, dada a elevada capacidade de processamento da placa gráfica. Dessa forma, a Tabela 6.1 evidencia os tempos de treino para cada rede neuronal.

Tabela 6.1: Tempos de Treino por Rede Neuronal.

Rede Neuronal	**Tempo de Treino**
FNN	5 min
LSTM	5 min
Bi-LSTM	1 h
GRU	3 min

6.3 Rede FNN

A avaliação do desempenho da rede foi feita utilizando os *datasets* de teste - `X_teste` e `y_teste_cat`. A Figura 6.1 representa a evolução da precisão ao longo das 20 épocas de treino. Pode-se constatar que as curvas de precisão e validação aumentaram consideravelmente durante as primeiras duas épocas, tendo atingido um *steady-state* nas restantes. O valor final da precisão situa-se perto dos 93 %. Já o da validação, ronda os 98 %.

De forma a avaliar a capacidade de previsão da rede, foi criada a matriz de confusão, que permite visualizar a quantidade de falhas ou acertos, relativamente aos dados de teste que são injetados na sua entrada. Para a rede FNN, a maioria das previsões foi correta, tendo havido um maior erro de previsão nos casos em que o utilizador estava em pé ("STANDING") ou sentado ("SITTING"), estando a confusão associada ao inverso. A Figura 6.2 evidencia essas mesmas previsões, comparativamente aos valores reais.

6.4 Rede LSTM

A rede LSTM, foi sujeita à mesma avaliação que a rede FNN. Dessa forma, foi obtido o gráfico de variação da precisão e validação da Figura 6.3. A partir desses dados, é possível verificar que a curva de precisão teve um aumento significativo nas primeiras 2 épocas, à semelhança do que ocorreu com a rede FNN, no entanto, a curva de validação situou-se em valores próximos de 98 % na maioria do tempo de

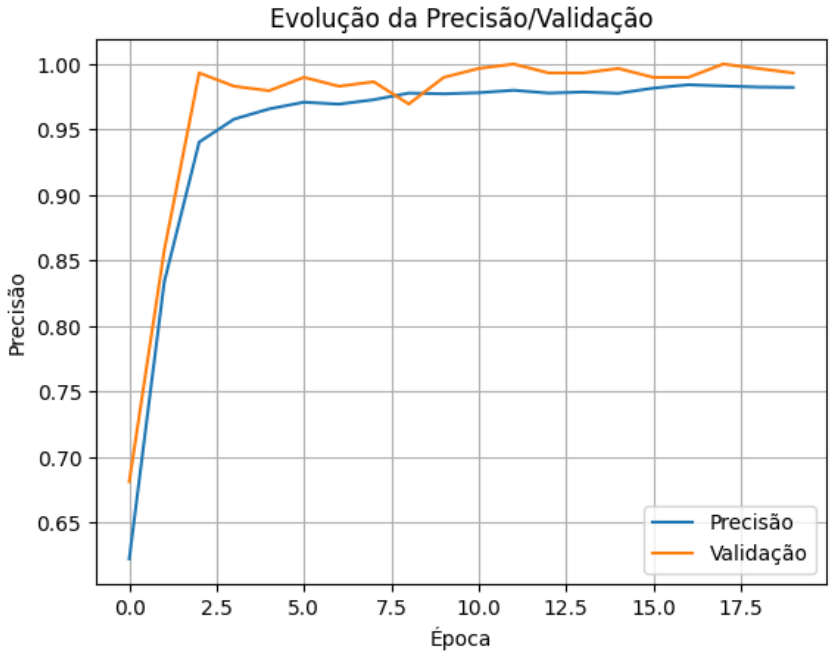


Figura 6.1: Evolução da Precisão e da Validação ao Longo do Treino da Rede FNN.

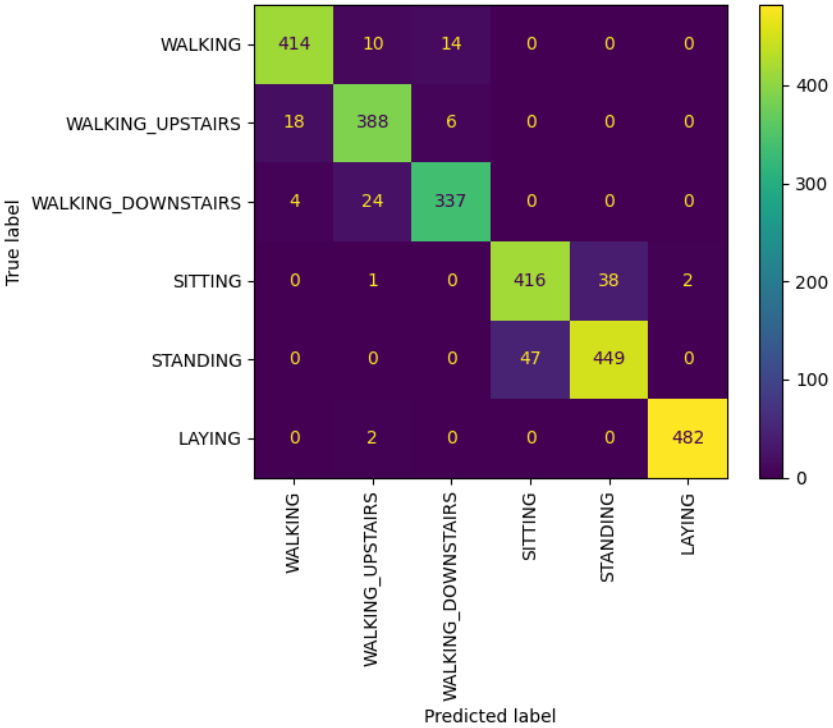


Figura 6.2: Matriz de Confusão da Rede FNN.

treino, tendo sofrido algumas oscilações. O valor final da precisão para esta rede situa-se nos 93 % e o de validação, perto dos 97 %.

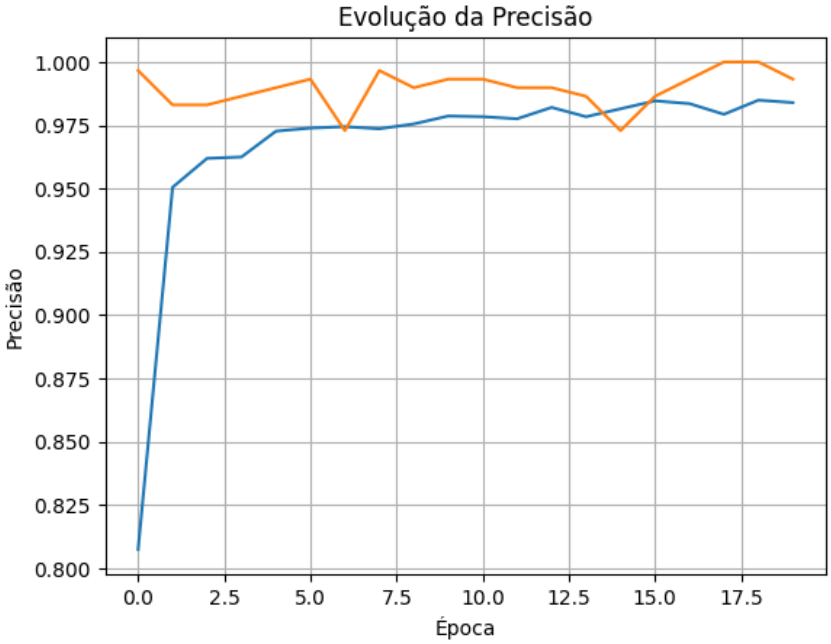


Figura 6.3: Evolução da Precisão e da Validação ao Longo do Treino da Rede LSTM.

Do mesmo modo que na rede FNN, a Figura 6.4 mostra a matriz de confusão obtida partindo dos dados previstos, quando comparados aos dados reais. À semelhança da rede FNN, a confusão mais notável deu-se na previsão do "STANDING" com o "SITTING", sendo também evidente um ligeiro aumento do erro no caso de "WALKING UPSTAIRS", em que a rede previa o movimento "WALKING".

6.5 Rede Bi-LSTM

Para a rede Bi-LSTM, foi criado o gráfico da Figura 6.5, permitindo inferir que a curva de precisão teve um crescimento mais gradual ao longo do treino, atingindo o *steady-state* nas proximidades da 13ª época. A curva de validação apresenta, também, um aumento gradual, tendo oscilado na proximidade da 4ª época. O valor final da precisão situa-se na proximidade dos 95 % e o da validação, nos 96 %.

Já a Figura 6.6 evidencia os resultados das previsões efetuadas pela rede, quando comparadas com os valores reais. Esta rede apresenta maior erro nas previsões, tendo aumentado significativamente os erros de identificação de "STANDING", de "WALKING" e de "WALKING UPSTAIRS".

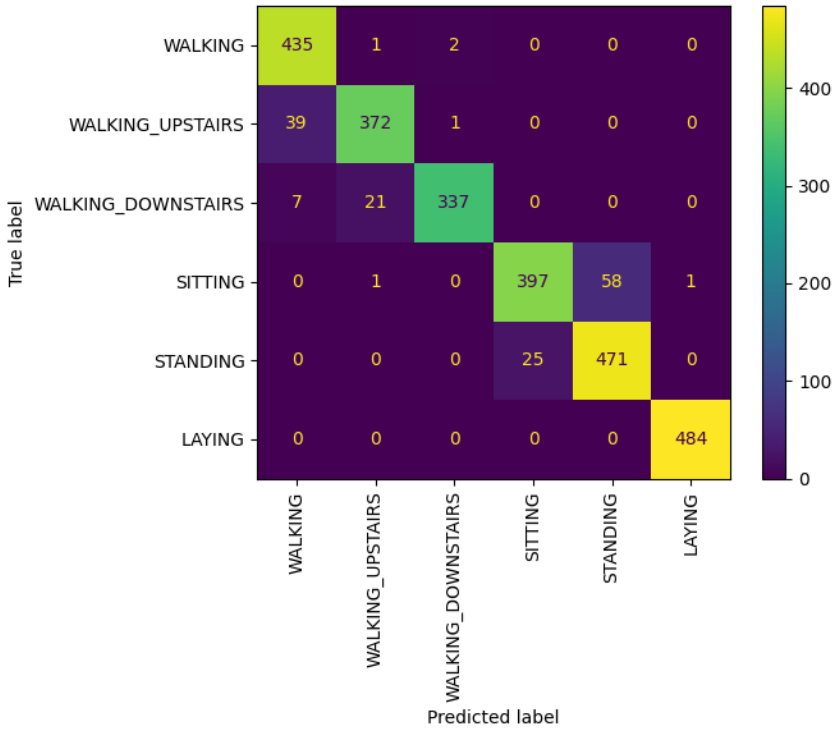


Figura 6.4: Matriz de Confusão da Rede LSTM.

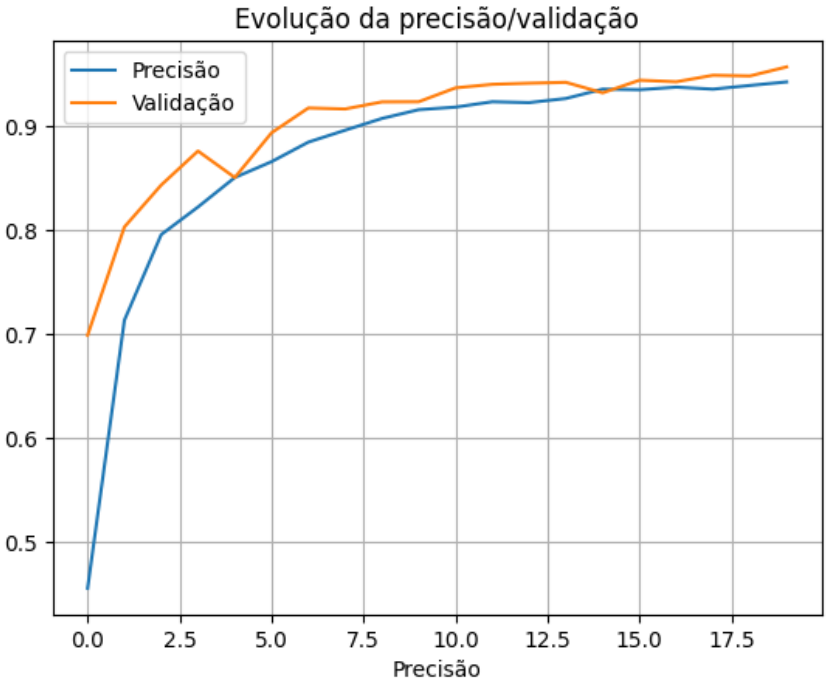


Figura 6.5: Evolução da Precisão e da Validação ao Longo do Treino da Rede Bi-LSTM.

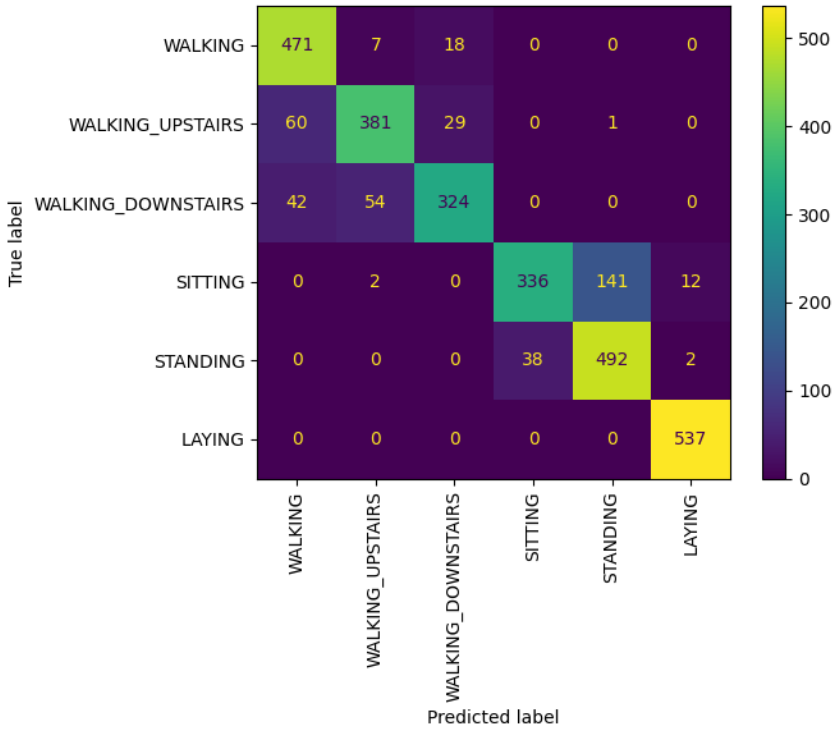


Figura 6.6: Matriz de Confusão da Rede Bi-LSTM.

6.6 Rede GRU

Finalmente, a rede GRU foi sujeita à mesma avaliação, resultando no gráfico da Figura 6.7. No mesmo, verifica-se que a curva de precisão teve um aumento semelhante ao da rede FNN, tendo atingido o *steady-state* na 10ª época, mas a curva de validação apresentou imensa oscilação ao longo de todo o treino. O valor final da precisão situa-se perto dos 97 % e o da validação, nas proximidades desse mesmo valor.

Passando à matriz de confusão, a Figura 6.8 ilustra a precisão do teste para cada previsão *vs.* o movimento real. É visível um aumento da confusão no "SITTING" relativamente às redes FNN e LSTM, tendo melhorado na previsão de "STANDING" quando comparada com a rede Bi-LSTM. De salientar a dificuldade da rede em prever o movimento "WALKING DOWNSTAIRS" quando comparado com o "WALKING UPSTAIRS".

6.7 Comparação dos Desempenhos Obtidos

É possível verificar que, em todas as redes, a precisão não foi afetada pelo treino em demasia, pois teve uma evolução crescente. Nas redes FNN e LSTM, o aumento da precisão foi muito significativo nas primeiras épocas do treino, indicando que, para os dados em causa, a aprendizagem foi mais eficaz nestas redes, quando comparadas

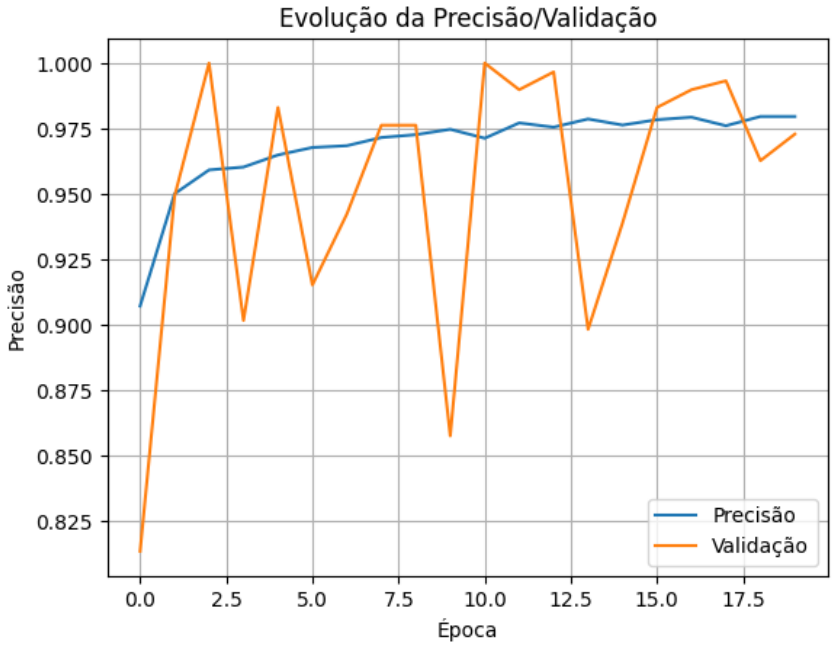


Figura 6.7: Evolução da Precisão e da Validação ao Longo do Treino da Rede GRU.

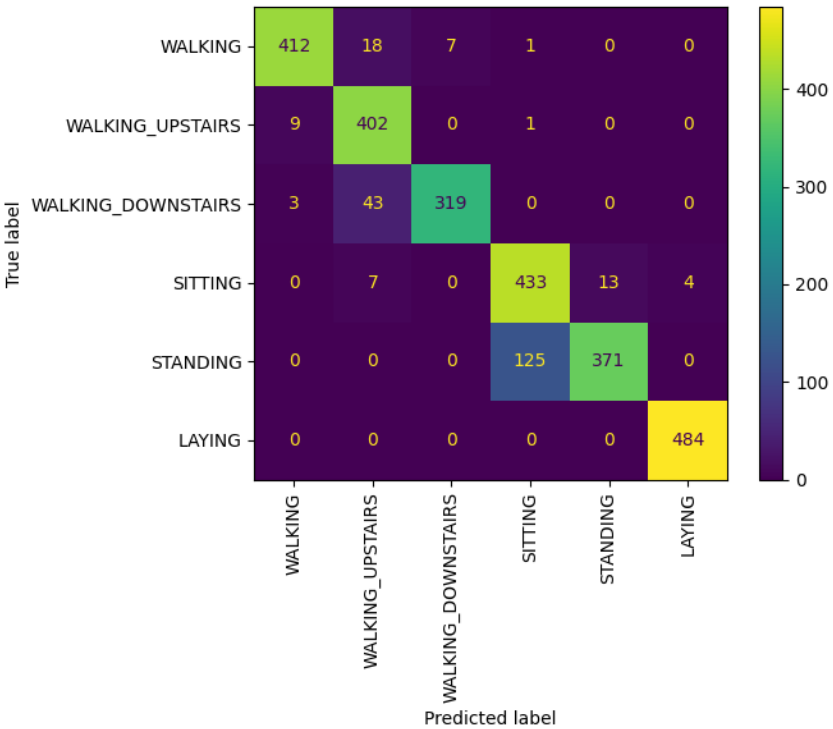


Figura 6.8: Matriz de Confusão da Rede GRU.

com as restantes. Desse modo, infere-se que estas redes não necessitariam de tantas épocas para atingir um resultado satisfatório.

Analisando a evolução da validação, é possível perceber que, na rede GRU houve uma grande oscilação ao longo do treino, o que poderá significar que a rede não foi bem projetada para os dados utilizados. Neste caso, poderá ter ocorrido *overfitting*, tendo a rede "aprendido algum ruído" dada a complexidade do problema e a reduzida quantidade de dados disponíveis. Nas restantes redes, a validação aparentou ter uma evolução normal.

Ao nível do desempenho nas previsões efetuadas, as redes apresentam respostas semelhantes, sendo que as falhas ocorreram maioritariamente nos movimentos "WALKING UPSTAIRS", "SITTING" e "STANDING". Tal facto pode dever-se à menor quantidade de amostras dessas classes nos *datasets*, como se pode verificar através da análise da Figura 4.3. Segundo referido em "A Public Domain Dataset for Human Activity Recognition Using Smartphones" [66], a confusão gerada nos movimentos "SITTING" e "STANDING" pode dever-se à menor dinâmica desses movimentos. Tendo em conta que não há significativas variações de aceleração nestes movimentos, os dados gerados pelo acelerómetro apresentam uma menor quantidade de informação.

No geral, e corroborando os valores de precisão, as redes conseguiram prever corretamente a maioria dos movimentos, retirando essa conclusão do facto de a diagonal principal das matrizes - que indicam os valores previstos que, simultaneamente, são reais - concentrarem a maioria das previsões efetuadas.

Após a análise dos gráficos obtidos e das matrizes de confusão, resta apenas analisar as métricas calculadas após o treino. Na Tabela 6.2 são apresentados os valores obtidos de cada uma dessas métricas.

Tabela 6.2: Métricas Obtidas Após o Treino de Cada Rede Neuronal.

Rede Neuronal	***Accuracy***	***Precision***	***Recall***	***F1-Score***
FNN	91.3 %	93.2 %	91.3 %	91.3 %
LSTM	94.6 %	94.0 %	94.6 %	94.6 %
Bi-LSTM	83.1 %	83.0 %	83.2 %	98.1 %
GRU	91.3 %	91.4 %	91.1 %	99.6 %

A partir destes dados, pode-se concluir que, num nível mais geral, todas as redes conseguiram obter valores de precisão e desempenho elevados, sendo que a rede Bi-LSTM foi a que se manteve em valores mais baixos (abaixo dos 90 %).

A rede LSTM foi a que obteve um valor de *accuracy* mais elevado, demonstrando ser a rede que mais vezes fez previsões corretas. A mesma também obteve o melhor resultado na métrica *precision*, permitindo concluir que foi a rede que fez as previsões positivas com mais qualidade. A mesma rede também se destacou no *recall*, significando isso que foi a rede que identificou corretamente uma maior quantidade

de amostras positivas, quando comparado com o total de amostras do *dataset*. Por fim, e como referido no Capítulo 3 (Secção 3.2), o *F1-Score* é a amálgama das métricas *precision* e *recall* e, para essa métrica, a rede GRU obteve o valor mais elevado do conjunto de redes neuronais.

6.8 Otimização das Redes Neuronais

Como forma de melhorar os resultados obtidos anteriormente, foram efetuadas duas intervenções às redes criadas.

A primeira intervenção tratou-se de uma mera tentativa de aumentar a quantidade de camadas e de neurónios/unidades de cada rede neuronal, para verificar se, de facto, é evidente alguma melhoria.

A segunda intervenção teve como objetivo, utilizar uma técnica de *fine tuning* para obter a estrutura ideal de rede neuronal, de forma a maximizar a precisão final do treino.

Os resultados obtidos dessas duas intervenções são mostrados de seguida.

6.8.1 Aumento do Número de Camadas e Neurónios

Na primeira fase de otimização das redes neuronais, foram obtidos os resultados da Tabela 6.3.

Tabela 6.3: Resultados de Precisão e Tempo de Treino.

Rede	**Duração do Treino**	***Accuracy* Final (Treino)**
FNN	5 min	94.2 %
LSTM	5 min	93.9 %
Bi-LSTM	73 h	46.7 %
GRU	3 min	95.6 %

No geral, os tempos de treino não sofreram variação, à exceção da rede Bi-LSTM, cujo tempo de treino teve um aumento significativo. No que concerne aos valores de *accuracy* obtidos no treino, as redes FNN e GRU tiveram melhorias significativas, enquanto que as redes LSTM e Bi-LSTM atingiram valores inferiores.

Deste modo, é possível concluir que nem todas as redes melhoram o seu desempenho com o aumento da sua complexidade. O caso mais evidente desta afirmação verificou-se na Bi-LSTM, que apesar de ter um treino mais longo e uma maior complexidade relativamente à proposta inicial, teve uma convergência pior.

6.8.2 Otimização Bayesiana

A segunda fase de otimização recorreu à técnica de *fine tuning* designada por otimização Bayesiana. Os resultados obtidos após 10 iterações do *tuner* foram os da Tabela 6.4.

Tabela 6.4: Resultados Obtidos na Otimização Bayesiana.

Rede	**Duração Total da Otimização**	**Melhor *Accuracy* (Treino)**
FNN	14 min	98.5 %
LSTM	23 min	98.6 %
Bi-LSTM	48 h	93.4 %
GRU	31 min	98.3 %

No geral, os tempos totais de otimização das 4 redes neuronais não foram muito elevados, à exceção da rede Bi-LSTM, cujos tempos de treino sempre foram substancialmente maiores. No que respeita à precisão mais elevada de cada rede, todos os valores foram superiores aos obtidos anteriormente, o que demonstra que a técnica de otimização permitiu uma melhoria significativa do desempenho de todas as redes neuronais.

Desta forma, as redes neuronais otimizadas têm a constituição da Tabela 6.5.

Tabela 6.5: Estruturas e Parâmetros de Treino Otimizados para as Redes Neuronais.

Rede	**Estrutura**	**Parâmetros de Treino**
FNN	*Dense* (113 Neurónios e ativação `relu`), Dense (421 Neurónios e ativação `sigmoid`), *Dense* (7 Neurónios e ativação `softmax`)	Função de Perda `binary_crossentropy` e otimizador `adam` com um *Learning Rate* de 0.001
LSTM	LSTM (117 Unidades), *Dense* (269 Neurónios e ativação `relu`) e *Dense* (7 Neurónios e ativação `softmax`)	Função de perda `binary_crossentropy` e otimizador `adam` com um *Learning Rate* de 0.0001
Bi-LSTM	Bi-LSTM (355 Unidades), *Dropout* (0.2592), Bi-LSTM (347 Unidades), *Dropout* (0.3632) e *Dense* (7 Neurónios e ativação `sigmoid`)	Função de perda `binary_crossentropy` e otimizador `adam` com um *Learning Rate* de 0.001
GRU	GRU (268 Unidades), GRU (286 Unidades), *BatchNormalization* e *Dense* (7 Neurónios e ativação `softmax`)	Função de perda `binary_crossentropy` e otimizador `adam` com um *Learning Rate* de 0.0001

Treino dos Modelos da Otimização Bayesiana

De forma a testar os modelos obtidos através da otimização Bayesiana, foi executado o treino e teste a cada modelo, tendo sido executados os mesmos passos que nas Secções 6.3, 6.4, 6.5 e 6.6.

Os tempos de treino para cada rede foram semelhantes aos apresentados na Tabela 6.1.

Os resultados obtidos das métricas estão presentes na Tabela 6.6, sendo possível compará-los com os valores obtidos inicialmente. Do mesmo modo, são apresentados os valores obtidos nas matrizes de confusão (Tabelas 6.7, 6.8, 6.9 e 6.10). De forma a simplificar a escrita de cada movimento nas matrizes de confusão, os movimentos são identificados da seguinte forma:

- W - WALKING
- WU - WALKING UPSTAIRS
- WD - WALKING DOWNSTAIRS
- SI - SITTING
- ST - STANDING
- L - LAYING

Tabela 6.6: Métricas Obtidas Após o Treino das Redes Inicial/Otimizada.

Rede	***Accuracy***	***Precision***	***Recall***	***F1-Score***
FNN	91.3 %/93.2 %	97.2 %/94.5 %	91.3 %/94.3 %	91.3 %/94.2 %
LSTM	94.6 %/94.3 %	93.96 %/94.3 %	94.6 %/94.3 %	94.6 %/93.1 %
Bi-LSTM	83.1 %/79.3 %	83.0 %/79.9 %	83.2 %/79.0 %	98.1 %/97.4 %
GRU	91.3 %/93.3 %	91.4 %/93.3 %	91.1 %/93.1 %	99.6 %/95.6 %

Quando comparados, os valores das métricas obtidos através dos dados de teste (Tabela 6.6) apresentam, na generalidade, valores ligeiramente superiores aos obtidos inicialmente, salientando a perda de desempenho da rede Bi-LSTM, cujas métricas desceram sem exceção. Destacam-se as redes FNN e LSTM, que se encontram na proximidade dos mesmos valores.

Tabela 6.7: Comparação das Matrizes de Confusão das Redes FNN Inicial/Otimizada.

	W	WU	WD	SI	ST	L
W	436/424	2/1	0/13	0/0	0/0	0/0
WU	22/28	390/377	0/7	0/0	0/0	0/0
WD	16/3	86/8	262/354	0/0	0/0	1/0
SI	0/0	3/3	0/0	412/415	42/39	0/0
ST	0/1	0/0	0/0	33/41	463/454	0/0
L	0/0	0/0	0/0	0/0	25/0	459/484

Analisando a comparação das matrizes de confusão inicial e pós otimização para a rede FNN (Tabela 6.7), é possível verificar um aumento do erro associado aos movimentos “WALKING”, “WALKING UPSTAIRS” e “STANDING”. Contudo, houve

uma ligeira diminuição do erro associado a "WALKING DOWNSTAIRS", "SITTING" e "LAYING".

Tabela 6.8: Comparação das Matrizes de Confusão das Redes LSTM Inicial/Otimizada.

	W	WU	WD	SI	ST	L
W	431/436	4/0	3/2	0/0	0/0	0/0
WU	18/41	392/370	2/1	0/0	0/0	0/0
WD	5/5	21/13	339/347	0/0	0/0	0/0
SI	0/0	1/2	0/0	426/385	30/70	0/0
ST	0/0	0/0	0/0	58/16	438/480	0/0
L	0/0	0/0	0/0	0/0	0/0	484/484

No caso da rede LSTM (Tabela 6.8), o modelo otimizado apresenta melhorias no caso "WALKING", assim como em "WALKING DOWNSTAIRS" e "STANDING". Os movimentos "WALKING UPSTAIRS" e "SITTING" apresentam uma redução da qualidade das previsões. O movimento "LAYING" manteve-se igual.

Tabela 6.9: Comparação das Matrizes de Confusão das Redes Bi-LSTM Inicial/Otimizada.

	W	WU	WD	SI	ST	L
W	471/299	7/60	18/79	0/0	0/0	0/0
WU	60/50	381/334	29/28	0/0	1/0	0/0
WD	42/41	54/88	324/236	0/0	0/0	0/0
SI	0/1	2/4	0/0	336/321	141/129	12/2
ST	0/1	0/0	0/0	38/44	492/451	2/0
L	0/0	0/0	0/0	0/0	0/23	537/461

Para a rede Bi-LSTM (Tabela 6.9), a otimização resultou numa perda em todas as previsões, tendo sido o único caso em que não foram registadas melhorias.

Tabela 6.10: Comparação das Matrizes de Confusão das Redes GRU Inicial/Otimizada.

	W	WU	WD	SI	ST	L
W	412/437	18/1	7/0	1/0	0/0	0/0
WU	9/48	402/364	0/0	1/0	0/0	0/0
WD	3/4	43/8	319/352	0/1	0/0	0/0
SI	0/0	7/1	0/0	433/435	13/21	4/0
ST	0/0	0/0	0/0	125/95	371/401	0/0
L	0/0	0/0	0/0	0/0	0/0	484/484

Por fim, a rede GRU otimizada apresentou melhorias na previsão dos movimentos "WALKING", "WALKING DOWNSTAIRS", "SITTING" e "STANDING", mantendo o mesmo valor em "LAYING" e não conseguindo um melhor resultado em "WALKING UPSTAIRS".

Capítulo 7

Conclusões

Após a realização do trabalho descrito neste documento, é possível enumerar algumas conclusões retiradas. A principal conclusão é a de que a IA e todas as suas componentes (ML e DL, nomeadamente) têm vindo a crescer, de forma a automatizar vários processos rotineiros do dia a dia, aumentando consideravelmente, o tempo criativo dos utilizadores. Além disso, pode ser útil em vários campos de aplicação, fazendo com que se torne uma tecnologia altamente adaptável às necessidades de várias áreas.

Focando particularmente na resolução do problema do reconhecimento de movimentos, os diferentes algoritmos de ML e DL existentes permitem uma monitorização constante dos movimentos de vários indivíduos com dificuldades de mobilidade. Esta problemática aplica-se, acima de tudo, à população mais idosa que, com o avançar dos anos, se tem vindo a tornar cada vez maior, dado o aumento da esperança média de vida. Futuramente, muitos processos rotineiros associados a esta temática poderão tornar-se menos cansativos, exigindo menos dos cuidadores que, através de um simples *Smartphone*, poderão avaliar o estado atual de cada paciente, não precisando de estar necessariamente no mesmo local que a pessoa monitorizada.

Na temática de implementação, é possível efetuar uma apreciação geral com base na execução da proposta, nas aprendizagens efetuadas através da pesquisa de informação e nas dificuldades encontradas durante o desenvolvimento do sistema de reconhecimento de movimentos.

No início do desenvolvimento das redes neuronais, foi utilizada a distribuição *Anaconda*, dada a facilidade de criar ambientes com versões específicas do *Python*.

No entanto, a instalação de alguns pacotes trouxe algum atraso aos desenvolvimentos (como foi o caso da biblioteca *Tensorflow*), uma vez que, não tendo conhecimento da possibilidade de selecionar outras versões da linguagem, a instalação estava a ser feita sobre a versão 12 do *Python*, não havendo compatibilidade entre os dois *softwares* ao momento da execução do projeto.

Outra dificuldade sentida durante o desenvolvimento prende-se com a perceção do funcionamento de cada rede/camada das redes neuronais desenvolvidas, nomeadamente, a arquitetura das redes LSTM.

Apesar das dificuldades evidenciadas, também se pode salientar que o objetivo proposto foi cumprido, tendo sido possível efetuar a análise do desempenho de várias redes neuronais, que procuraram solucionar o mesmo problema, partindo dos mesmos dados e usando parâmetros de treino semelhantes.

Também foi possível colocar em prática as técnicas de *fine tuning*, utilizadas para aumentar o desempenho das redes neuronais, através de um conjunto de otimizações (tanto ao nível das estruturas das redes, como dos parâmetros de treino mais eficazes em cada caso), que, de facto, se verificaram úteis para a melhoria significativa das redes LSTM e GRU. A rede FNN, apesar de apresentar algumas melhorias, também sofreu algumas falhas, mantendo-se equilibrada quanto a movimentos com previsão correta e incorreta. A rede Bi-LSTM mostrou ser a mais complexa de implementar, evidenciando que a estrutura utilizada não seria a ideal ou os dados disponíveis não foram suficientes para obter um resultado satisfatório em todas as propostas de simulação.

Para concluir, tanto a rede LSTM como a rede GRU obtiveram resultados finais semelhantes, numa ótica de análise das matrizes de confusão, mas, com o auxílio das métricas escalares, é possível indicar que a rede LSTM se adaptou com mais facilidade, à problemática do reconhecimento de movimentos e aos dados presentes no *UCI HAR Dataset*.

A principal vantagem retirada da implementação das redes neuronais foi o aprofundamento dos conhecimentos obtidos em Unidades Curriculares do Ramo de Automação e Sistemas do Mestrado em Engenharia Eletrotécnica e de Computadores, onde o tema fora abordado em jeito de introdução à aprendizagem máquina.

7.1 Trabalho Futuro

Numa perspetiva de melhoria do trabalho já implementado, a criação de redes de outra natureza seria um ponto de grande importância. Ao gerar gráficos a partir dos dados dos sensores, poderia ser utilizada, por exemplo, uma rede CNN e, através desses dados, efetuar o treino que utilizasse como *features*, os padrões presentes em todas as amostras gráficas conhecidas pela rede.

De forma a avaliar a possibilidade de se melhorar ainda mais os tempos de treino e o desempenho das redes, poder-se-iam implementar outras tipologias de *tuners*, tais como o *Random Search* ou o *SelectKBest*. Por exemplo, para o caso do *SelectKBest*, seria possível utilizar diferentes *score functions* para efetuar uma seleção das *k features* mais relevantes no *dataset*.

Também a utilização de um *dataset* com menos *features* e que representasse os dados obtidos de cada sensor ao longo do tempo (ao invés de valores obtidos pela análise matemática desses sinais), poderiam ter como consequência, um aumento significativo dos valores de desempenho em cada um dos quatro casos estudados.

Por outro lado, e com um enfoque maior na componente de testes com dados em tempo real, a utilização das redes neuronais criadas, num contexto prático de reconhecimento de movimentos teria, também, uma grande relevância. Tal objetivo poderia ser atingido através da criação de uma aplicação *Web* que, através da comunicação com um *Smartphone*, permitisse obter os dados dos sensores desse equipamento e, em tempo real, fizesse o tratamento desses dados para a entrada da rede neuronal selecionada. O *feedback* visual em termos de interface com o utilizador seria o movimento detetado pela respetiva rede aquando da análise do sinal obtido.

Referências

[1] "OMS publica primeiro relatório global sobre inteligência artificial na saúde e seis princípios orientadores para sua concepção e uso - OPAS/OMS | Organização Pan-Americana da Saúde." https://www.paho.org/pt/noticias/28-6-2021-oms-publica-primeiro-relatorio-global-sobre-inteligencia-artificial-na-saude-e. [Citado na página 1]

[2] D. Yalalov, "Google anuncia um reconhecedor de gestos AI para interagir com a Web em tempo real." https://mpost.io/pt/google-announces-an-ai-gesture-recognizer-to-interact-with-the-web-in-real-time/, Mar. 2023. [Citado na página 2]

[3] "Aplicações e Soluções de Deep Learning: Entenda a Importância!." https://www.opencadd.com.br/blog/aplicacoes-de-deep-learning-para-seus-projetos. [Citado na página 3]

[4] D. Matos, "7 Casos de Uso de Deep Learning." https://www.cienciaedados.com/7-casos-de-uso-de-deep-learning/, June 2017. [Citado na página 3]

[5] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Energy efficient smartphone-based activity recognition using fixed-point arithmetic," *Energy ...*, 2013. [Citado nas páginas 3, 21, 35 e 37]

[6] R. M. Vicari, "Influências das Tecnologias da Inteligência Artificial no ensino," *Estudos Avançados*, vol. 35, pp. 73–84, Apr. 2021. [Citado na página 7]

[7] "O que é machine learning? | IBM." https://www.ibm.com/br-pt/topics/machine-learning. [Citado na página 7]

[8] "'Deep learning': um conceito-chave para levar a inteligência artificial a um nível superior." https://www.iberdrola.com/inovacao/deep-learning. [Citado na página 7]

[9] T. M. Mitchell, *Machine Learning*. McGraw-Hill series in computer science, McGraw-Hill Science/Engineering/Math, 1997. [Citado na página 8]

[10] "What is artificial intelligence (AI)?." https://www.mathworks.com/discovery/artificial-intelligence.html. [Citado nas páginas ix, 8 e 9]

[11] Dataconomy, "How to create an artificial intelligence?." https://dataconomy.com/2023/03/13/how-to-create-an-artificial-intelligence/. [Citado na página 9]

[12] PyTorch, "Pytorch." https://www.pytorch.org. [Citado na página 9]

[13] TensorFlow, "Tensorflow." https://www.tensorflow.org/?hl=pt-br. [Citado na página 9]

[14] MATLAB, "Matlab." https://www.mathworks.com/products/matlab.html. [Citado na página 9]

[15] martinekuan, "Arquitetura de inteligência artificial (IA) - Azure Architecture Center." https://learn.microsoft.com/pt-pt/azure/architecture/ai-ml/, Sept. 2023. [Citado na página 10]

[16] "Azure Data Factory – Serviço de Integração de Dados | Microsoft Azure." https://azure.microsoft.com/pt-pt/products/data-factory. [Citado na página 10]

[17] "Azure Databricks | Microsoft Azure." https://azure.microsoft.com/pt-pt/products/databricks. [Citado na página 10]

[18] "Machine Learning - Amazon Web Services." https://aws.amazon.com/pt/sagemaker/. [Citado na página 10]

[19] H. Demirkan, S. Earley, and R. R. Harmon, "Cognitive computing," *IT Professional*, vol. 19, no. 4, pp. 16–20, 2017. [Citado na página 12]

[20] Intellipaat, "Pros & cons of artificial intelligence - insights from ai experts." https://intellipaat.com/blog/pros-and-cons-of-ai/?US. [Citado na página 12]

[21] R. S. Barbosa, "Conceitos de inteligência artificial (ia),machine learning (ml) e deep learning (dl)." [Citado nas páginas 13, 14, 15, 16 e 18]

[22] Coursera, "Machine learning vs. ai: Differences, uses, and benefits." https://www.coursera.org/articles/machine-learning-vs-ai. [Citado na página 13]

[23] IBM, "What is computer vision? | ibm." https://www.ibm.com/topics/computer-vision. [Citado na página 13]

[24] FIA, "Machine Learning: como funciona, benefícios, tipos e exemplos." https://fia.com.br/blog/machine-learning/, Nov. 2021. [Citado na página 14]

[25] R. Khadka, "Machine Learning Types #2." https://towardsdatascience.com/machine-learning-types-2-c1291d4f04b1, Sept. 2017. [Citado nas páginas ix e 15]

[26] "Histologia do neurônio." https://www.kenhub.com/pt/library/anatomia/histologia-do-neuronio. [Citado na página 16]

[27] "Deep Learning (DL)." http://mriquestions.com/what-is-a-neural-network.html. [Citado nas páginas ix e 16]

[28] LogAp, "Funcionamento do deep learning: conceito, história e aplicações." https://logap.com.br/blog/deep-learning-2/, Dec. 2021. [Citado na página 17]

[29] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology press, 2005. [Citado na página 17]

[30] A. Dennanni, "Acelere seus projetos de Deep Learning com Redes Neurais pré-treinadas." https://medium.com/neuronio-br/acelere-seus-projetos-de-deep-learning-com-redes-neurais-pre-treinadas-b5c43bc590e1, Oct. 2018. [Citado na página 17]

[31] "Redes neurais - o que são e qual sua importância?." https://www.sas.com/pt_br/insights/analytics/neural-networks.html. [Citado na página 17]

[32] "VGG-16 convolutional neural network - MATLAB vgg16." https://www.mathworks.com/help/deeplearning/ref/vgg16.html. [Citado na página 17]

[33] "Guia avançado do Inception v3 | Cloud TPU." https://cloud.google.com/tpu/docs/inception-v3-advanced?hl=pt-br. [Citado na página 18]

[34] R. Horev, "BERT Explained: State of the art language model for NLP." https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270, Nov. 2018. [Citado na página 18]

[35] "GPT-2: 1.5B release." https://openai.com/research/gpt-2-1-5b-release. [Citado na página 18]

[36] "Monolayer neural network." https://www.researchgate.net/figure/Monolayer-neural-network_fig3_238747009. [Citado nas páginas ix e 18]

[37] "Unsupervised Feature Learning and Deep Learning Tutorial." http://deeplearning.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/. [Citado nas páginas ix e 19]

[38] A. Moreira, A. Marques, C. Modesto, N. Nobre, R. Silva, R. B. Silva, and W. Oliveira, "Processamento de linguagem natural aplicado à inteligência artificial," *Revista da UI_IPSantarém*, vol. 9, pp. 81–90, Dec. 2021. [Citado na página 20]

[39] "What is Natural Language Processing? | IBM." `https://www.ibm.com/topics/natural-language-processing`, abstract = Natural language processing enables machines to understand and respond to text or voice data. [Citado na página 20]

[40] C. Britto, G. Pinto, and G. Silva, "Um breve estudo sobre inteligência artificial aplicado à robótica em tempos de COVID-19." `https://intranet.cbt.ifsp.edu.br/qualif/volume08/relato02_ed_08.pdf`, 2021. [Citado na página 20]

[41] P. Wang, W. Li, P. Ogunbona, J. Wan, and S. Escalera, "RGB-D-based Human Motion Recognition with Deep Learning: A Survey." `http://arxiv.org/abs/1711.08362`, Apr. 2018. [Citado na página 21]

[42] J. R. Kwapisz, G. M. Weiss, and S. A. Moore, "Activity recognition using cell phone accelerometers," *ACM SIGKDD Explorations Newsletter*, vol. 12, no. 2, pp. 74–82, 2011-03. [Citado na página 21]

[43] M. H. Arshad, M. Bilal, and A. Gani, "Human Activity Recognition: Review, Taxonomy and Open Challenges," *Sensors*, vol. 22, p. 6463, Jan. 2022. [Citado na página 21]

[44] N. Mohamed, M. B. Mustafa, and N. Jomhari, "A Review of the Hand Gesture Recognition System: Current Progress and Future Directions," *IEEE Access*, vol. 9, pp. 157422–157436, 2021. [Citado na página 21]

[45] "O que são Redes Neurais? | IBM." `https://www.ibm.com/br-pt/topics/neural-networks`. [Citado na página 23]

[46] D. R. Yehoshua, "Perceptrons: The First Neural Network Model," June 2023. [Citado nas páginas ix e 24]

[47] "A four-layered feed-forward network (ffn) showing the input from brn (read vector in red) and the output to brn from the fourth layer." `https://www.researchgate.net/figure/FFN-model-A-four-layered-feed-forward-network-FFN-showing-the-input-from-BRN-read_fig2_356588469`. [Citado nas páginas ix e 24]

[48] "Recurrent Neural Networks (RNN) | Working | Steps | Advantages." https://www.educba.com/recurrent-neural-networks-rnn/, Aug. 2019. [Citado nas páginas ix e 25]

[49] "Long Short Term Memory: redes neurais artificiais que são capazes de ler, ouvir e ver." `https://ateliware.com/blog/long-short-term-memory`, abstract = Redes Neurais Long Short Term Memory (LSTMs) representam um tipo de rede neural artificial e são aplicadas principalmente em tarefas de processamento de linguagem natural, áudio e vídeo. Saiba mais. [Citado na página 25]

[50] "Memória longa de curto prazo (lstm) - Definirtec." `https://definirtec.com/memoria-longa-de-curto-prazo-lstm/`, Mar. 2021. [Citado na página 25]

[51] Paulo Victor Correia, Lucileide Dantasy, Luiz Affonso Guedes, and Marcelo Fernandes, "Análise de Desempenho de Redes Neurais LSTM com Técnicas de Pruning para Detecção de Falhas em Processos Industrias," 2021. [Citado na página 25]

[52] S. Yan, "Understanding LSTM and its diagrams." `https://blog.mlreview.com/understanding-lstm-and-its-diagrams-37e2f46f1714`, Nov. 2017. [Citado nas páginas ix e 26]

[53] "Papers with Code - BiLSTM Explained." `https://paperswithcode.com/method/bilstm`. [Citado nas páginas ix e 27]

[54] Anishnama, "Understanding Gated Recurrent Unit (GRU) in Deep Learning." `https://medium.com/@anishnama20/understanding-gated-recurrent-unit-gru-in-deep-learning-2e54923f3e2`, May 2023. [Citado nas páginas ix e 28]

[55] "Schematic diagram of a basic convolutional neural network (cnn) architecture." `https://www.researchgate.net/figure/Schematic-diagram-of-a-basic-convolutional-neural-network-CNN-architecture-26_fig1_336805909`. [Citado nas páginas ix e 28]

[56] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014. [Citado nas páginas ix e 29]

[57] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Citado na página 29]

[58] S. Shahinfar, P. Meek, and G. Falzon, ""How many images do I need?" Understanding how sample size per class affects deep learning model performance metrics for balanced designs in autonomous wildlife monitoring," *Ecological Informatics*, vol. 57, p. 101085, May 2020. [Citado na página 29]

[59] S. Narkhede, "Understanding confusion matrix." https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62. [Citado na página 30]

[60] "How to interpret a confusion matrix for a machine learning model." https://www.evidentlyai.com/classification-metrics/confusion-matrix. [Citado nas páginas ix e 31]

[61] Amanatullah, "Fine-Tuning the Model: What, Why, and How." https://medium.com/@amanatulla1606/fine-tuning-the-model-what-why-and-how-e7fa52bc8ddf, Sept. 2023. [Citado na página 32]

[62] A. AK, "A Guide to Hyperparameter Tuning: Enhancing Machine Learning Models." https://medium.com/@abelkuriakose/a-guide-to-hyperparameter-tuning-enhancing-machine-learning-models-69dc9e0f02ea, Oct. 2023. [Citado na página 32]

[63] A. Rakhecha, "Understanding Learning Rate." https://towardsdatascience.com/https-medium-com-dashingaditya-rakhecha-understanding-learning-rate-dd5da26bb6de, July 2019. [Citado nas páginas ix e 33]

[64] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *Lecture Notes in Computer Science* (J. Bravo, R. Hervás, and M. Rodríguez, eds.), vol. 7657, pp. 216–223, Springer Berlin Heidelberg, 2012. [Citado nas páginas 35 e 37]

[65] D. Anguita, A. Ghio, L. Oneto, F. L. Parra, and J. L. Ortiz, "Human activity recognition on smartphones for mobile context awareness," in *Advances in Neural Information Processing Systems 26: proceedings of the 2012 conference*, 2012. [Citado nas páginas 35 e 37]

[66] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones," *Computational Intelligence*, 2013. [Citado nas páginas 35, 36, 37 e 65]

[67] T. Choudhury, G. Borriello, S. Consolvo, D. Haehnel, B. Harrison, B. Hemingway, J. Hightower, P. P. Klasnja, K. Koscher, A. LaMarca, J. A. Landay, L. LeGrand, J. Lester, A. Rahimi, A. Rea, and D. Wyatt, "The mobile sensing platform: An embedded activity recognition system," *IEEE Pervasive Computing*, vol. 7, no. 2, pp. 32–41, 2008-04. [Citado na página 36]

[68] P. Turaga, R. Chellappa, V. S. Subrahmanian, and O. Udrea, "Machine recognition of human activities: A survey," *IEEE Transactions on Circuits and*

Systems for Video Technology, vol. 18, no. 11, pp. 1473–1488, 2008-11. [Citado na página 36]

[69] R. Poppe, "A survey on vision-based human action recognition," *Image and Vision Computing*, vol. 28, no. 6, pp. 976–990, 2010-06. [Citado na página 36]

[70] C.-C. Chang and C.-J. Lin, "Libsvm - a library for support vector machines." `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`. [Citado na página 37]

[71] "UCI machine learning repository: Human activity recognition using smartphones data set." `https://archive.ics.uci.edu/dataset/240/human+activity+recognition+using+smartphones`. [Citado na página 43]

[72] K. Team, "Keras documentation: BayesianOptimization Tuner." `https://keras.io/api/keras_tuner/tuners/bayesian/`. [Citado na página 53]

Anexo A

Features do *Dataset* de Entrada

A.1 Identificação das *Features* de Entrada

- tBodyAcc-mean()-X
- tBodyAcc-mean()-Y
- tBodyAcc-mean()-Z
- tBodyAcc-std()-X
- tBodyAcc-std()-Y
- tBodyAcc-std()-Z
- tBodyAcc-mad()-X
- tBodyAcc-mad()-Y
- tBodyAcc-mad()-Z
- tBodyAcc-max()-X
- tBodyAcc-max()-Y
- tBodyAcc-max()-Z
- tBodyAcc-min()-X
- tBodyAcc-min()-Y
- tBodyAcc-min()-Z
- tBodyAcc-sma()
- tBodyAcc-energy()-X
- tBodyAcc-energy()-Y
- tBodyAcc-energy()-Z
- tBodyAcc-iqr()-X
- tBodyAcc-iqr()-Y
- tBodyAcc-iqr()-Z
- tBodyAcc-entropy()-X
- tBodyAcc-entropy()-Y
- tBodyAcc-entropy()-Z
- tBodyAcc-arCoeff()-X,1
- tBodyAcc-arCoeff()-X,2

- tBodyAcc-arCoeff()-X,3
- tBodyAcc-arCoeff()-X,4
- tBodyAcc-arCoeff()-Y,1
- tBodyAcc-arCoeff()-Y,2
- tBodyAcc-arCoeff()-Y,3
- tBodyAcc-arCoeff()-Y,4
- tBodyAcc-arCoeff()-Z,1
- tBodyAcc-arCoeff()-Z,2
- tBodyAcc-arCoeff()-Z,3
- tBodyAcc-arCoeff()-Z,4
- tBodyAcc-correlation()-X,Y
- tBodyAcc-correlation()-X,Z
- tBodyAcc-correlation()-Y,Z
- tGravityAcc-mean()-X
- tGravityAcc-mean()-Y
- tGravityAcc-mean()-Z
- tGravityAcc-std()-X
- tGravityAcc-std()-Y
- tGravityAcc-std()-Z
- tGravityAcc-mad()-X
- tGravityAcc-mad()-Y
- tGravityAcc-mad()-Z
- tGravityAcc-max()-X
- tGravityAcc-max()-Y
- tGravityAcc-max()-Z
- tGravityAcc-min()-X
- tGravityAcc-min()-Y
- tGravityAcc-min()-Z
- tGravityAcc-sma()
- tGravityAcc-energy()-X
- tGravityAcc-energy()-Y
- tGravityAcc-energy()-Z
- tGravityAcc-iqr()-X
- tGravityAcc-iqr()-Y
- tGravityAcc-iqr()-Z
- tGravityAcc-entropy()-X
- tGravityAcc-entropy()-Y
- tGravityAcc-entropy()-Z
- tGravityAcc-arCoeff()-X,1
- tGravityAcc-arCoeff()-X,2
- tGravityAcc-arCoeff()-X,3
- tGravityAcc-arCoeff()-X,4
- tGravityAcc-arCoeff()-Y,1
- tGravityAcc-arCoeff()-Y,2
- tGravityAcc-arCoeff()-Y,3
- tGravityAcc-arCoeff()-Y,4
- tGravityAcc-arCoeff()-Z,1
- tGravityAcc-arCoeff()-Z,2
- tGravityAcc-arCoeff()-Z,3
- tGravityAcc-arCoeff()-Z,4
- tGravityAcc-correlation()-X,Y
- tGravityAcc-correlation()-X,Z
- tGravityAcc-correlation()-Y,Z
- tBodyAccJerk-mean()-X

- tBodyAccJerk-mean()-Y
- tBodyAccJerk-mean()-Z
- tBodyAccJerk-std()-X
- tBodyAccJerk-std()-Y
- tBodyAccJerk-std()-Z
- tBodyAccJerk-mad()-X
- tBodyAccJerk-mad()-Y
- tBodyAccJerk-mad()-Z
- tBodyAccJerk-max()-X
- tBodyAccJerk-max()-Y
- tBodyAccJerk-max()-Z
- tBodyAccJerk-min()-X
- tBodyAccJerk-min()-Y
- tBodyAccJerk-min()-Z
- tBodyAccJerk-sma()
- tBodyAccJerk-energy()-X
- tBodyAccJerk-energy()-Y
- tBodyAccJerk-energy()-Z
- tBodyAccJerk-iqr()-X
- tBodyAccJerk-iqr()-Y
- tBodyAccJerk-iqr()-Z
- tBodyAccJerk-entropy()-X
- tBodyAccJerk-entropy()-Y
- tBodyAccJerk-entropy()-Z
- tBodyAccJerk-arCoeff()-X,1
- tBodyAccJerk-arCoeff()-X,2
- tBodyAccJerk-arCoeff()-X,3
- tBodyAccJerk-arCoeff()-X,4
- tBodyAccJerk-arCoeff()-Y,1
- tBodyAccJerk-arCoeff()-Y,2
- tBodyAccJerk-arCoeff()-Y,3
- tBodyAccJerk-arCoeff()-Y,4
- tBodyAccJerk-arCoeff()-Z,1
- tBodyAccJerk-arCoeff()-Z,2
- tBodyAccJerk-arCoeff()-Z,3
- tBodyAccJerk-arCoeff()-Z,4
- tBodyAccJerk-correlation()-X,Y
- tBodyAccJerk-correlation()-X,Z
- tBodyAccJerk-correlation()-Y,Z
- tBodyGyro-mean()-X
- tBodyGyro-mean()-Y
- tBodyGyro-mean()-Z
- tBodyGyro-std()-X
- tBodyGyro-std()-Y
- tBodyGyro-std()-Z
- tBodyGyro-mad()-X
- tBodyGyro-mad()-Y
- tBodyGyro-mad()-Z
- tBodyGyro-max()-X
- tBodyGyro-max()-Y
- tBodyGyro-max()-Z
- tBodyGyro-min()-X
- tBodyGyro-min()-Y
- tBodyGyro-min()-Z

- tBodyGyro-sma()
- tBodyGyro-energy()-X
- tBodyGyro-energy()-Y
- tBodyGyro-energy()-Z
- tBodyGyro-iqr()-X
- tBodyGyro-iqr()-Y
- tBodyGyro-iqr()-Z
- tBodyGyro-entropy()-X
- tBodyGyro-entropy()-Y
- tBodyGyro-entropy()-Z
- tBodyGyro-arCoeff()-X,1
- tBodyGyro-arCoeff()-X,2
- tBodyGyro-arCoeff()-X,3
- tBodyGyro-arCoeff()-X,4
- tBodyGyro-arCoeff()-Y,1
- tBodyGyro-arCoeff()-Y,2
- tBodyGyro-arCoeff()-Y,3
- tBodyGyro-arCoeff()-Y,4
- tBodyGyro-arCoeff()-Z,1
- tBodyGyro-arCoeff()-Z,2
- tBodyGyro-arCoeff()-Z,3
- tBodyGyro-arCoeff()-Z,4
- tBodyGyro-correlation()-X,Y
- tBodyGyro-correlation()-X,Z
- tBodyGyro-correlation()-Y,Z
- tBodyGyroJerk-mean()-X
- tBodyGyroJerk-mean()-Y
- tBodyGyroJerk-mean()-Z
- tBodyGyroJerk-std()-X
- tBodyGyroJerk-std()-Y
- tBodyGyroJerk-std()-Z
- tBodyGyroJerk-mad()-X
- tBodyGyroJerk-mad()-Y
- tBodyGyroJerk-mad()-Z
- tBodyGyroJerk-max()-X
- tBodyGyroJerk-max()-Y
- tBodyGyroJerk-max()-Z
- tBodyGyroJerk-min()-X
- tBodyGyroJerk-min()-Y
- tBodyGyroJerk-min()-Z
- tBodyGyroJerk-sma()
- tBodyGyroJerk-energy()-X
- tBodyGyroJerk-energy()-Y
- tBodyGyroJerk-energy()-Z
- tBodyGyroJerk-iqr()-X
- tBodyGyroJerk-iqr()-Y
- tBodyGyroJerk-iqr()-Z
- tBodyGyroJerk-entropy()-X
- tBodyGyroJerk-entropy()-Y
- tBodyGyroJerk-entropy()-Z
- tBodyGyroJerk-arCoeff()-X,1
- tBodyGyroJerk-arCoeff()-X,2
- tBodyGyroJerk-arCoeff()-X,3
- tBodyGyroJerk-arCoeff()-X,4

- tBodyGyroJerk-arCoeff()-Y,1
- tBodyGyroJerk-arCoeff()-Y,2
- tBodyGyroJerk-arCoeff()-Y,3
- tBodyGyroJerk-arCoeff()-Y,4
- tBodyGyroJerk-arCoeff()-Z,1
- tBodyGyroJerk-arCoeff()-Z,2
- tBodyGyroJerk-arCoeff()-Z,3
- tBodyGyroJerk-arCoeff()-Z,4
- tBodyGyroJerk-correlation()-X,Y
- tBodyGyroJerk-correlation()-X,Z
- tBodyGyroJerk-correlation()-Y,Z
- tBodyAccMag-mean()
- tBodyAccMag-std()
- tBodyAccMag-mad()
- tBodyAccMag-max()
- tBodyAccMag-min()
- tBodyAccMag-sma()
- tBodyAccMag-energy()
- tBodyAccMag-iqr()
- tBodyAccMag-entropy()
- tBodyAccMag-arCoeff()1
- tBodyAccMag-arCoeff()2
- tBodyAccMag-arCoeff()3
- tBodyAccMag-arCoeff()4
- tGravityAccMag-mean()
- tGravityAccMag-std()
- tGravityAccMag-mad()
- tGravityAccMag-max()
- tGravityAccMag-min()
- tGravityAccMag-sma()
- tGravityAccMag-energy()
- tGravityAccMag-iqr()
- tGravityAccMag-entropy()
- tGravityAccMag-arCoeff()1
- tGravityAccMag-arCoeff()2
- tGravityAccMag-arCoeff()3
- tGravityAccMag-arCoeff()4
- tBodyAccJerkMag-mean()
- tBodyAccJerkMag-std()
- tBodyAccJerkMag-mad()
- tBodyAccJerkMag-max()
- tBodyAccJerkMag-min()
- tBodyAccJerkMag-sma()
- tBodyAccJerkMag-energy()
- tBodyAccJerkMag-iqr()
- tBodyAccJerkMag-entropy()
- tBodyAccJerkMag-arCoeff()1
- tBodyAccJerkMag-arCoeff()2
- tBodyAccJerkMag-arCoeff()3
- tBodyAccJerkMag-arCoeff()4
- tBodyGyroMag-mean()
- tBodyGyroMag-std()
- tBodyGyroMag-mad()
- tBodyGyroMag-max()

- tBodyGyroMag-min()
- tBodyGyroMag-sma()
- tBodyGyroMag-energy()
- tBodyGyroMag-iqr()
- tBodyGyroMag-entropy()
- tBodyGyroMag-arCoeff()1
- tBodyGyroMag-arCoeff()2
- tBodyGyroMag-arCoeff()3
- tBodyGyroMag-arCoeff()4
- tBodyGyroJerkMag-mean()
- tBodyGyroJerkMag-std()
- tBodyGyroJerkMag-mad()
- tBodyGyroJerkMag-max()
- tBodyGyroJerkMag-min()
- tBodyGyroJerkMag-sma()
- tBodyGyroJerkMag-energy()
- tBodyGyroJerkMag-iqr()
- tBodyGyroJerkMag-entropy()
- tBodyGyroJerkMag-arCoeff()1
- tBodyGyroJerkMag-arCoeff()2
- tBodyGyroJerkMag-arCoeff()3
- tBodyGyroJerkMag-arCoeff()4
- fBodyAcc-mean()-X
- fBodyAcc-mean()-Y
- fBodyAcc-mean()-Z
- fBodyAcc-std()-X
- fBodyAcc-std()-Y
- fBodyAcc-std()-Z
- fBodyAcc-mad()-X
- fBodyAcc-mad()-Y
- fBodyAcc-mad()-Z
- fBodyAcc-max()-X
- fBodyAcc-max()-Y
- fBodyAcc-max()-Z
- fBodyAcc-min()-X
- fBodyAcc-min()-Y
- fBodyAcc-min()-Z
- fBodyAcc-sma()
- fBodyAcc-energy()-X
- fBodyAcc-energy()-Y
- fBodyAcc-energy()-Z
- fBodyAcc-iqr()-X
- fBodyAcc-iqr()-Y
- fBodyAcc-iqr()-Z
- fBodyAcc-entropy()-X
- fBodyAcc-entropy()-Y
- fBodyAcc-entropy()-Z
- fBodyAcc-maxInds-X
- fBodyAcc-maxInds-Y
- fBodyAcc-maxInds-Z
- fBodyAcc-meanFreq()-X
- fBodyAcc-meanFreq()-Y
- fBodyAcc-meanFreq()-Z
- fBodyAcc-skewness()-X

- fBodyAcc-kurtosis()-X
- fBodyAcc-skewness()-Y
- fBodyAcc-kurtosis()-Y
- fBodyAcc-skewness()-Z
- fBodyAcc-kurtosis()-Z
- fBodyAcc-bandsEnergy()-1,8
- fBodyAcc-bandsEnergy()-9,16
- fBodyAcc-bandsEnergy()-17,24
- fBodyAcc-bandsEnergy()-25,32
- fBodyAcc-bandsEnergy()-33,40
- fBodyAcc-bandsEnergy()-41,48
- fBodyAcc-bandsEnergy()-49,56
- fBodyAcc-bandsEnergy()-57,64
- fBodyAcc-bandsEnergy()-1,16
- fBodyAcc-bandsEnergy()-17,32
- fBodyAcc-bandsEnergy()-33,48
- fBodyAcc-bandsEnergy()-49,64
- fBodyAcc-bandsEnergy()-1,24
- fBodyAcc-bandsEnergy()-25,48
- fBodyAcc-bandsEnergy()-1,8
- fBodyAcc-bandsEnergy()-9,16
- fBodyAcc-bandsEnergy()-17,24
- fBodyAcc-bandsEnergy()-25,32
- fBodyAcc-bandsEnergy()-33,40
- fBodyAcc-bandsEnergy()-41,48
- fBodyAcc-bandsEnergy()-49,56
- fBodyAcc-bandsEnergy()-57,64
- fBodyAcc-bandsEnergy()-1,16
- fBodyAcc-bandsEnergy()-17,32
- fBodyAcc-bandsEnergy()-33,48
- fBodyAcc-bandsEnergy()-49,64
- fBodyAcc-bandsEnergy()-1,24
- fBodyAcc-bandsEnergy()-25,48
- fBodyAcc-bandsEnergy()-1,8
- fBodyAcc-bandsEnergy()-9,16
- fBodyAcc-bandsEnergy()-17,24
- fBodyAcc-bandsEnergy()-25,32
- fBodyAcc-bandsEnergy()-33,40
- fBodyAcc-bandsEnergy()-41,48
- fBodyAcc-bandsEnergy()-49,56
- fBodyAcc-bandsEnergy()-57,64
- fBodyAcc-bandsEnergy()-1,16
- fBodyAcc-bandsEnergy()-17,32
- fBodyAcc-bandsEnergy()-33,48
- fBodyAcc-bandsEnergy()-49,64
- fBodyAcc-bandsEnergy()-1,24
- fBodyAcc-bandsEnergy()-25,48
- fBodyAccJerk-mean()-X
- fBodyAccJerk-mean()-Y
- fBodyAccJerk-mean()-Z
- fBodyAccJerk-std()-X
- fBodyAccJerk-std()-Y
- fBodyAccJerk-std()-Z
- fBodyAccJerk-mad()-X

- fBodyAccJerk-mad()-Y
- fBodyAccJerk-mad()-Z
- fBodyAccJerk-max()-X
- fBodyAccJerk-max()-Y
- fBodyAccJerk-max()-Z
- fBodyAccJerk-min()-X
- fBodyAccJerk-min()-Y
- fBodyAccJerk-min()-Z
- fBodyAccJerk-sma()
- fBodyAccJerk-energy()-X
- fBodyAccJerk-energy()-Y
- fBodyAccJerk-energy()-Z
- fBodyAccJerk-iqr()-X
- fBodyAccJerk-iqr()-Y
- fBodyAccJerk-iqr()-Z
- fBodyAccJerk-entropy()-X
- fBodyAccJerk-entropy()-Y
- fBodyAccJerk-entropy()-Z
- fBodyAccJerk-maxInds-X
- fBodyAccJerk-maxInds-Y
- fBodyAccJerk-maxInds-Z
- fBodyAccJerk-meanFreq()-X
- fBodyAccJerk-meanFreq()-Y
- fBodyAccJerk-meanFreq()-Z
- fBodyAccJerk-skewness()-X
- fBodyAccJerk-kurtosis()-X
- fBodyAccJerk-skewness()-Y
- fBodyAccJerk-kurtosis()-Y
- fBodyAccJerk-skewness()-Z
- fBodyAccJerk-kurtosis()-Z
- fBodyAccJerk-bandsEnergy()-1,8
- fBodyAccJerk-bandsEnergy()-9,16
- fBodyAccJerk-bandsEnergy()-17,24
- fBodyAccJerk-bandsEnergy()-25,32
- fBodyAccJerk-bandsEnergy()-33,40
- fBodyAccJerk-bandsEnergy()-41,48
- fBodyAccJerk-bandsEnergy()-49,56
- fBodyAccJerk-bandsEnergy()-57,64
- fBodyAccJerk-bandsEnergy()-1,16
- fBodyAccJerk-bandsEnergy()-17,32
- fBodyAccJerk-bandsEnergy()-33,48
- fBodyAccJerk-bandsEnergy()-49,64
- fBodyAccJerk-bandsEnergy()-1,24
- fBodyAccJerk-bandsEnergy()-25,48
- fBodyAccJerk-bandsEnergy()-1,8
- fBodyAccJerk-bandsEnergy()-9,16

- fBodyAccJerk-bandsEnergy()-17,24
- fBodyAccJerk-bandsEnergy()-25,32
- fBodyAccJerk-bandsEnergy()-33,40
- fBodyAccJerk-bandsEnergy()-41,48
- fBodyAccJerk-bandsEnergy()-49,56
- fBodyAccJerk-bandsEnergy()-57,64
- fBodyAccJerk-bandsEnergy()-1,16
- fBodyAccJerk-bandsEnergy()-17,32
- fBodyAccJerk-bandsEnergy()-33,48
- fBodyAccJerk-bandsEnergy()-49,64
- fBodyAccJerk-bandsEnergy()-1,24
- fBodyAccJerk-bandsEnergy()-25,48
- fBodyAccJerk-bandsEnergy()-1,8
- fBodyAccJerk-bandsEnergy()-9,16
- fBodyAccJerk-bandsEnergy()-17,24
- fBodyAccJerk-bandsEnergy()-25,32
- fBodyAccJerk-bandsEnergy()-33,40
- fBodyAccJerk-bandsEnergy()-41,48
- fBodyAccJerk-bandsEnergy()-49,56
- fBodyAccJerk-bandsEnergy()-57,64
- fBodyAccJerk-bandsEnergy()-1,16
- fBodyAccJerk-bandsEnergy()-17,32
- fBodyAccJerk-bandsEnergy()-33,48
- fBodyAccJerk-bandsEnergy()-49,64
- fBodyAccJerk-bandsEnergy()-1,24
- fBodyAccJerk-bandsEnergy()-25,48
- fBodyGyro-mean()-X
- fBodyGyro-mean()-Y
- fBodyGyro-mean()-Z
- fBodyGyro-std()-X
- fBodyGyro-std()-Y
- fBodyGyro-std()-Z
- fBodyGyro-mad()-X
- fBodyGyro-mad()-Y
- fBodyGyro-mad()-Z
- fBodyGyro-max()-X
- fBodyGyro-max()-Y
- fBodyGyro-max()-Z
- fBodyGyro-min()-X
- fBodyGyro-min()-Y
- fBodyGyro-min()-Z

- fBodyGyro-sma()
- fBodyGyro-energy()-X
- fBodyGyro-energy()-Y
- fBodyGyro-energy()-Z
- fBodyGyro-iqr()-X
- fBodyGyro-iqr()-Y
- fBodyGyro-iqr()-Z
- fBodyGyro-entropy()-X
- fBodyGyro-entropy()-Y
- fBodyGyro-entropy()-Z
- fBodyGyro-maxInds-X
- fBodyGyro-maxInds-Y
- fBodyGyro-maxInds-Z
- fBodyGyro-meanFreq()-X
- fBodyGyro-meanFreq()-Y
- fBodyGyro-meanFreq()-Z
- fBodyGyro-skewness()-X
- fBodyGyro-kurtosis()-X
- fBodyGyro-skewness()-Y
- fBodyGyro-kurtosis()-Y
- fBodyGyro-skewness()-Z
- fBodyGyro-kurtosis()-Z
- fBodyGyro-bandsEnergy()-1,8
- fBodyGyro-bandsEnergy()-9,16
- fBodyGyro-bandsEnergy()-17,24
- fBodyGyro-bandsEnergy()-25,32
- fBodyGyro-bandsEnergy()-33,40
- fBodyGyro-bandsEnergy()-41,48
- fBodyGyro-bandsEnergy()-49,56
- fBodyGyro-bandsEnergy()-57,64
- fBodyGyro-bandsEnergy()-1,16
- fBodyGyro-bandsEnergy()-17,32
- fBodyGyro-bandsEnergy()-33,48
- fBodyGyro-bandsEnergy()-49,64
- fBodyGyro-bandsEnergy()-1,24
- fBodyGyro-bandsEnergy()-25,48
- fBodyGyro-bandsEnergy()-1,8
- fBodyGyro-bandsEnergy()-9,16
- fBodyGyro-bandsEnergy()-17,24
- fBodyGyro-bandsEnergy()-25,32
- fBodyGyro-bandsEnergy()-33,40
- fBodyGyro-bandsEnergy()-41,48
- fBodyGyro-bandsEnergy()-49,56
- fBodyGyro-bandsEnergy()-57,64
- fBodyGyro-bandsEnergy()-1,16
- fBodyGyro-bandsEnergy()-17,32
- fBodyGyro-bandsEnergy()-33,48
- fBodyGyro-bandsEnergy()-49,64
- fBodyGyro-bandsEnergy()-1,24
- fBodyGyro-bandsEnergy()-25,48
- fBodyGyro-bandsEnergy()-1,8
- fBodyGyro-bandsEnergy()-9,16
- fBodyGyro-bandsEnergy()-17,24
- fBodyGyro-bandsEnergy()-25,32

- fBodyGyro-bandsEnergy()-33,40
- fBodyGyro-bandsEnergy()-41,48
- fBodyGyro-bandsEnergy()-49,56
- fBodyGyro-bandsEnergy()-57,64
- fBodyGyro-bandsEnergy()-1,16
- fBodyGyro-bandsEnergy()-17,32
- fBodyGyro-bandsEnergy()-33,48
- fBodyGyro-bandsEnergy()-49,64
- fBodyGyro-bandsEnergy()-1,24
- fBodyGyro-bandsEnergy()-25,48
- fBodyAccMag-mean()
- fBodyAccMag-std()
- fBodyAccMag-mad()
- fBodyAccMag-max()
- fBodyAccMag-min()
- fBodyAccMag-sma()
- fBodyAccMag-energy()
- fBodyAccMag-iqr()
- fBodyAccMag-entropy()
- fBodyAccMag-maxInds
- fBodyAccMag-meanFreq()
- fBodyAccMag-skewness()
- fBodyAccMag-kurtosis()
- fBodyBodyAccJerkMag-mean()
- fBodyBodyAccJerkMag-std()
- fBodyBodyAccJerkMag-mad()
- fBodyBodyAccJerkMag-max()
- fBodyBodyAccJerkMag-min()
- fBodyBodyAccJerkMag-sma()
- fBodyBodyAccJerkMag-energy()
- fBodyBodyAccJerkMag-iqr()
- fBodyBodyAccJerkMag-entropy()
- fBodyBodyAccJerkMag-maxInds
- fBodyBodyAccJerkMag-meanFreq()
- fBodyBodyAccJerkMag-skewness()
- fBodyBodyAccJerkMag-kurtosis()
- fBodyBodyGyroMag-mean()
- fBodyBodyGyroMag-std()
- fBodyBodyGyroMag-mad()
- fBodyBodyGyroMag-max()
- fBodyBodyGyroMag-min()
- fBodyBodyGyroMag-sma()
- fBodyBodyGyroMag-energy()
- fBodyBodyGyroMag-iqr()
- fBodyBodyGyroMag-entropy()
- fBodyBodyGyroMag-maxInds
- fBodyBodyGyroMag-meanFreq()
- fBodyBodyGyroMag-skewness()
- fBodyBodyGyroMag-kurtosis()
- fBodyBodyGyroJerkMag-mean()
- fBodyBodyGyroJerkMag-std()
- fBodyBodyGyroJerkMag-mad()
- fBodyBodyGyroJerkMag-max()
- fBodyBodyGyroJerkMag-min()

- fBodyBodyGyroJerkMag-sma()
- fBodyBodyGyroJerkMag-energy()
- fBodyBodyGyroJerkMag-iqr()
- fBodyBodyGyroJerkMag-entropy()
- fBodyBodyGyroJerkMag-maxInds
- fBodyBodyGyroJerkMag-meanFreq()
- fBodyBodyGyroJerkMag-skewness()
- fBodyBodyGyroJerkMag-kurtosis()
- angle(tBodyAccMean,gravity)
- angle(tBodyAccJerkMean),gravityMean)
- angle(tBodyGyroMean,gravityMean)
- angle(tBodyGyroJerkMean,gravityMean)
- angle(X,gravityMean)
- angle(Y,gravityMean)
- angle(Z,gravityMean)