POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

DroneGest – Interface Gestual para Controlo de Drones

Licenciatura em Engenharia Informática

Diogo Alexandre Ferreira Leonardo

João Pedro Sá da Silva Custódio

Leiria, julho de 2024

POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

DroneGest – Interface Gestual para Controlo de Drones

Licenciatura em Engenharia Informática

Diogo Alexandre Ferreira Leonardo

João Pedro Sá da Silva Custódio

Trabalho de Projeto da unidade curricular de Projeto Informático realizado sob a orientação do Professor Doutor Roberto Ribeiro, do Professor Doutor Nuno Rodrigues, do Professor Doutor João Ramos e do Professor Doutor António Pereira

Leiria, julho de 2024

Dedicatória

Este trabalho é dedicado a todos aqueles que, de forma incansável e carinhosa, nos apoiaram ao longo desta jornada.

Às nossas famílias, pela força, paciência e amor incondicionais demonstrados durante os momentos mais exigentes. Sem o vosso apoio constante, este projeto não teria sido possível. Vocês são a nossa fonte de inspiração, a base sólida sobre a qual construímos os nossos sonhos e a nossa maior motivação para continuar a lutar pelos nossos objetivos.

Aos nossos professores e orientadores, pelo conhecimento valioso que partilharam connosco e pela orientação dedicada e paciente que nos ofereceram. A vossa sabedoria e apoio foi fundamental para o desenvolvimento deste projeto. Agradecemos pela vossa disponibilidade, pelas palavras de encorajamento e pelos desafios que nos lançaram, que nos ajudaram a crescer e a superar as nossas expectativas.

Aos nossos colegas e amigos, pela colaboração incansável e pelas intermináveis horas de debate, brainstorming e partilha de ideias. A vossa camaradagem, espírito de equipa e apoio mútuo foram cruciais para tornar este percurso não só produtivo, mas também memorável e agradável. Juntos, enfrentámos desafios e celebrámos conquistas, e cada um de vocês teve um papel fundamental nesta jornada.

E, por fim, aos pioneiros e entusiastas da tecnologia, cuja paixão por inovação nos inspira a sonhar mais alto e a criar com mais determinação.

Deixamos aqui o nosso profundo agradecimento a todos por acreditarem em nós, por nos apoiarem incondicionalmente e por nos ajudarem a alcançar mais um marco significativo nas nossas vidas. A cada um de vocês, dedicamos este trabalho com imensa gratidão e reconhecimento.

Agradecimentos

Gostaríamos de expressar os nossos mais sinceros agradecimentos a todos aqueles que de alguma forma contribuíram para a realização deste projeto, "DroneGest - Interface Gestual para Controlo de Drones".

Em primeiro lugar, agradecemos aos nossos orientadores, Professor Roberto Aguiar Ribeiro, Professor Nuno Carlos Sousa Rodrigues, Professor João Pedro Ferreira Ramos e Professor António Manuel de Jesus Pereira, cujas orientações, paciência e conhecimentos foram fundamentais para o desenvolvimento deste trabalho. A sua dedicação e disponibilidade para esclarecer dúvidas, bem como as suas sugestões ao longo do processo, foram cruciais para o nosso sucesso. Agradecemos também ao corpo docente do Departamento de Engenharia Informática pela excelente formação e pelo suporte contínuo durante todo o curso.

Adicionalmente, gostaríamos de agradecer às nossas famílias e amigos pelo apoio incondicional e encorajamento ao longo deste período. O suporte emocional e a compreensão face às exigências do projeto foram inestimáveis. Agradecemos igualmente aos nossos colegas de curso, cujas colaborações e partilhas de conhecimento se mostraram valiosas em diversas fases do projeto.

Por fim, reconhecemos a importância das instituições e empresas que nos permitiram aceder a recursos e dados essenciais para a realização das nossas investigações e testes. A todos que, de alguma forma, contribuíram para a concretização deste projeto, deixamos o nosso profundo reconhecimento e gratidão.

Resumo

Este trabalho enquadra-se no âmbito da Inteligência Artificial e tem como objetivo o controlo de drones e respetivos payloads através da leitura de gestos executados pelo utilizador. Para isso, foi proposta uma biblioteca de 9 gestos dinâmicos e 1 movimento de descanso (noise), executados através de um único braço. Estes gestos são captados por sensores de movimento localizados nas costas do pulso, num dispositivo do tipo *wearable* e interpretados por um algoritmo de Machine Learning (ML), treinado com o conjunto de gestos propostos.

O projeto dividiu-se em 5 fases principais, sendo a primeira a pesquisa acerca da leitura e interpretação de gestos, com recurso a sensores, pesquisa sobre algoritmos de ML, especificamente as suas características assim como requisitos e pesquisa de *datasets* já criados com gestos provenientes de sensores. De seguida, na fase 2, procedemos a testes e experiências com recurso a algoritmos de ML existentes, encontrados na nossa pesquisa e utilizando um *dataset* retirado do website *Kaggle*. Na fase 3 iniciámos a criação e definição de um conjunto de gestos personalizado, que dariam origem ao nosso próprio *dataset* e que correspondem a ações especificas do drone. Já com este conjunto de dados como base, partimos para a fase 4 do projeto que seria o treino de modelos de ML com o algoritmo que havíamos desenvolvido baseado em redes neuronais convulsionais de uma dimensão para que aliado com o *dataset* criado anteriormente e alternando entre diversos parâmetros, encontrássemos a configuração mais precisa e otimizada possível para o problema. Por fim, na fase 5, desenvolvemos um protótipo no formato de uma interface web 3D, capaz de interpretar os gestos captados e simular os movimentos de um drone.

Resultados do modelo treinado:

Macro average:

- Precision: 0.9832
- Recall: 0.9845
- F1-score: 0.9837

Weighted average:

- Precision: 0.9841
- Recall: 0.9836
- F1-score: 0.9838

As conclusões do projeto indicam se é viável o controlo de drones de forma precisa e eficiente usando uma interface gestual baseada num algoritmo de Machine Learning. O projeto tem potencial para também ser aplicado em diversas áreas como tarefas industriais ou operações de resgate onde é imprescindível um controlo confiável e rápido de drones.

Palavras-chave: Drone, Inteligência Artificial, Gesto, Machine Learning, *dataset*

Abstract

This work falls within the scope of Artificial Intelligence and aims to control drones and their payloads by reading gestures made by the user. To this end, a library of 9 dynamic gestures and 1 resting movement (noise), executed by a single arm, has been proposed. These gestures are captured by motion sensors located on the back of the wrist in a wearable device and interpreted by a Machine Learning (ML) algorithm trained with the proposed set of gestures.

The project was divided into 5 main phases, the first being research into reading and interpreting gestures using sensors, research into ML algorithms, specifically their characteristics and requirements, and research into datasets already created with gestures from sensors. Next, in phase 2, we carried out tests and experiments using existing ML algorithms found in our research and using a dataset taken from the Kaggle website. In phase 3, we began to create and define a set of personalized gestures, which would give rise to our own dataset, and which correspond to specific drone actions. With this dataset as a base, we moved on to phase 4 of the project, which was to train ML models with the algorithm we had developed based on one-dimensional convulsive Neuronal Networks so that, combined with the dataset created earlier and alternating between various parameters, we could find the most accurate and optimized configuration possible for the problem. Finally, in phase 5, we developed a prototype in the form of a 3D web interface, capable of interpreting the gestures captured and simulating the movements of a drone.

Results of the trained model:

Macro average:

- Precision: 0.9832
- Recall: 0.9845
- F1-score: 0.9837

Weighted average:

- Precision: 0.9841
- Recall: 0.9836
- F1-score: 0.9838

The project's conclusions indicate whether it is feasible to control drones accurately and efficiently using a gesture interface based on a Machine Learning algorithm. The project also has the potential to be applied in various areas such as industrial tasks or rescue operations where reliable and fast drone control is essential.

Keywords: Drone, Artificial Intelligence, Gestures, Machine Learning, dataset

Índice

Dedicatória ..... **ii**

Agradecimentos ..... **iii**

Resumo ..... **v**

Abstract ..... **viii**

Lista de Figuras ..... **xv**

Lista de tabelas ..... **xviii**

Lista de siglas e acrónimos ..... **xix**

1. Introdução **1**

1.1. Contexto do Projeto **1**

1.2. Motivação e Objetivos **2**

1.3. Estrutura do relatório **3**

2. Pesquisa Preliminar **4**

2.1. Gestos Estáticos e Dinâmicos **4**

2.2. Machine Learning **5**

2.2.1. O que é e Como Treinar um Modelo de ML 6

2.2.2. Hiper parâmetros mais Importantes no Treino de um Modelo 6

2.2.3. *Neuronal Network*s 7

2.2.4. Avaliação do Modelo 9

2.2.5. Dificuldades Encontradas 11

2.3. Microcontrolador e Sensores **12**

2.3.1. Microcontrolador 12

2.3.2. Inertial Measurement Unit 13

2.4. Conjuntos de Dados (*datasets*) **15**

2.4.1. Definição de *dataset* 15

2.4.2. Criação de D*atasets* 16

2.4.3. Estrutura do *Dataset* 16

2.5. Tecnologia a Utilizar 16

2.5.1. TensorFlow e TensorFlow Micro 17

2.5.2. PyTorch 17

2.5.3. Redes Neuronais Convolucionais 18

2.6. Pré Processamento dos Dados 20

2.6.1. Coordenadas de *Quaternions* 21

2.6.2. Normalização dos *Quaternions* 21

2.6.3. Importância da Coordenada Z (Ângulo de Euler Z) 22

2.6.4. Conclusões sobre o Pré Processamento de Dados 23

3. Testes Preliminares 24

3.1. Dificuldades a Encontrar um *Dataset* Compatível 24

3.2. *Dataset* de Reconhecimento de Superfícies 24

3.3. Aprender, Testar e Validar 25

3.3.1. Análise do Algoritmo para Treino de Modelos 25

3.3.2. Análise do *Dataset* e da sua Estrutura 25

3.3.3. Treino do Modelo e Resultados Obtidos 25

4. Construção do *Dataset* 28

4.1. Definição dos Gestos Utilizados 28

4.2. Construção do Código Utilizado para ler os Dados dos Sensores 32

4.3. Construção do *Script* para Registar os Movimentos 33

4.3.1. Inicio da Construção do *Script* 33

4.3.2. Melhoria do *Script* 34

4.4. Qualidade e Gestão dos Dados do D*ataset* 39

4.4.1. Duplicação de Dados do *Dataset* 39

4.4.2. Técnicas de *Data Augmentation* 40

4.4.3. Movimento *Noise* (*idle*) 40

4.5. Expansão do *Dataset* **40**
4.5.1. *Script* para Combinar *Datasets* 41
4.5.2. Resultado da Expansão do *Dataset* 44

4.6. Erros Conhecidos **44**
4.6.1. Erro: *IndexError* 44
4.6.2. Erro de Resposta do Arduino 45

5. Treino de Modelos com o D*ataset* Próprio **46**

5.1. Primeira Ronda de Treino com *Dataset* Próprio **46**
5.1.1. Treino do Modelo 46
5.1.2. Treino de Modelos com Ajuste de Parâmetros 49

5.2. Parametrização e Funções em Algoritmos de ML **50**
5.2.1. Ajuste dos Principais Parâmetros 50
5.2.2. Definição dos Hiper Parâmetros 50
5.2.3. Funções de ML 51

5.3. Algoritmo Final e Treino em Bulk **51**

5.4. Treino de Modelos com o Novo Algoritmo **55**

6. Finalização da implementação, demonstração e Testes **57**

6.1. Conclusão do *Dataset* **57**

6.2. Treino e comparação de modelos **58**

6.3. Caracterização dos Hyperparameters **63**

6.4. Escolha do Modelo **64**
6.4.1. Resumo das Métricas 64
6.4.2. Análise das Métricas 65
6.4.3. Decisão 65

6.5. Desenvolvimento de Protótipo Demonstrativo **65**
6.5.1. Escolha do Protótipo 66
6.5.2. Desenvolvimento do Protótipo 66

6.5.3. *Server-side* – Codigo e Configurações 67
6.5.4. *Client-side* – Código e configurações 69

6.6. Testes Finais e Validação de Protótipo 80

7. Conclusão 83

7.1. Objetivos Alcançados 83

7.2. Impacto e Potencial Futuro 84

7.3. Trabalho Futuro 84

7.4. Considerações Finais 84

Bibliografia ou Referências Bibliográficas 85

Anexos 87

Glossário 89

Lista de Figuras

Figura 1 - Gestos dinâmicos 5

Figura 2 - Neural Network Scheme - geeksforgeeks.org 9

Figura 3 - Microcontrolador ESP32C6 - espressif-docs.readthedocs-hosted.com 13

Figura 4 - Setup microcontrolador e sensores 14

Figura 5 - Funcionamento de uma CNN - alura.com.br 20

Figura 6 - Surfaces Count Bar Graph 26

Figura 7 - Model Accuracy Plot (surfaces) 26

Figura 8 - Model Loss Plot (surfaces) 27

Figura 9 - Gesto Cima (class 1) 28

Figura 10 - Gesto Baixo (class 2) 29

Figura 11 - Gesto Esquerda (class 3) 29

Figura 12 - Gesto Direita (class 4) 30

Figura 13 - Gesto Trás (class 5) 30

Figura 14 - Gesto Frente (class 6) 31

Figura 15 - Gesto Girar (class 7) 31

Figura 16 - Gesto Palmas (class 8) 31

Figura 17 - Gesto Corte (class 9) 32

Figura 18 - Código arduino para definir intervalo de leitura 32

Figura 19 - Código Python para definir o header do *dataset* 33

Figura 20 - Código Python para leitura do ficheiro 34

Figura 21 - Código Python para leitura da porta serial 34

Figura 22 - Código Python para escrita no *dataset* 35

Figura 23 - Código Python para verificar a existência do ficheiro 35

Figura 24 - Código Python para inserção de informações 36

Figura 25 - Código Python de verificação do ficheiro de validação 36

Figura 26 - Código Python para criação do *backup* 36

Figura 27 - Código Python para manter movimentos do mesmo tamanho 38

Figura 28 - Código Python para pausar registo 39

Figura 29 - Output script generate_duplicated_data_from_csv.py 39
Figura 30 - Código Python para receber último *Group_id* 41
Figura 31 - Código Python para atualizar *Group_id* 42
Figura 32 - Output ficheiro combine*datasets*.py 43
Figura 33 - Exemplo de ficheiros de output 43
Figura 34 - Gesture Count (contagem de gestos por class) dataset 3300 gestos 44
Figura 35 - Erro *index* out of range 45
Figura 36 - Erro de o sensor parar de ler os dados 45
Figura 37 - Gesture Count Plot (contagem de gestos por classe) 46
Figura 38 - Ficheiro de dados CSV (treino) e ficheiro de dados CSV (validação) 47
Figura 39 - Ficheiro de dados CSV (treino) e ficheiro de dados CSV (validação) parte 2 47
Figura 40 - Output do treino do modelo 47
Figura 41 - Model accuracy plot do modelo treinado 48
Figura 42 - Model Loss plot do modelo treinado 48
Figura 43 - Código Python para criar um modelo 52
Figura 44 - Código Python para criação de ficheiro com dados do modelo treinado 53
Figura 45 - Output da versão final do ficheiro TrainAndEvaluateModel.py 54
Figura 46 - Imagens ilustrativas do ficheiro PDF gerado 55
Figura 47 - Grafico model accuracy com demonstração de overfitting 56
Figura 48 - Gesture Count (contagem de gestos por classe) do *dataset* final 57
Figura 49 - Gráfico das curvas ROC do pior modelo treinado 60
Figura 50 - Gráfico da precisão do pior modelo treinado 60
Figura 51 - Gráfico da curva de perda do pior modelo treinado 61
Figura 52 - Gráfico das curvas ROC do melhor modelo treinado 62
Figura 53 - Gráfico da precisão do melhor modelo treinado 62
Figura 54 - Gráfico da curva de perda do melhor modelo treinado 63
Figura 55 - Imagem do drone utilizado no desenvolvimento do protótipo 67
Figura 56 - Código Python de conexão entre servidor e cliente 68
Figura 57 - Código Python de previsão de gesto 69
Figura 58 - Código Javascript para criação da cena 70

Figura 59 - Código Javascript para importação do modelo 70
Figura 60 - Instanciação do *websocket* no cliente-side 72
Figura 61 - Definição de parâmetros 73
Figura 62 - Código Javascript com evento que lida com mensagens recebidas 74
Figura 63 - Código Javascript para identificação do gesto executado 76
Figura 64 - Código Javascript para pousar/levantar voo 77
Figura 65 - Código javascript para retornar à posição inicial 78
Figura 66 - Código Javascript do fecho do método *Switch* 79
Figura 67 - Código Javascript para execução do som do drone 80
Figura 68 - Drone a Levantar 80
Figura 69 - Drone a Ir Para Trás 81
Figura 70 - Drone a Pairar 81
Figura 71 - Drone a Descer 81
Figura 72 - Drone a Girar 82
Figura 73 - Drone a Pousar 82
Figura 74 - Código Python para duplicação de dados 88

Lista de tabelas

Tabela 1 - Resultados dos modelos com o *dataset* de 350 gestos 49

Tabela 2 - Resultados dos modelos treinados com o *dataset* de 3300 gestos..... 55

Tabela 3 - Resultados dos modelos treinados com o *dataset* final de 5500 gestos 58

Tabela 4 - Tabela dos HyperParameters do melhor modelo treinado..... 63

Tabela 5 - Resumo de métricas dos modelos 64

Tabela 6 - Resumo da precisão dos modelos..... 65

Lista de siglas e acrónimos

ADN	Ácido Desoxirribonucleico
ASR	Automatic Speech Recognition
CNN	Convolutional *Neuronal Network*
CSV	Comma Separated Values
ESTG	Escola Superior de Tecnologia e Gestão
IA	Inteligência Artificial
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
IoT	Internet of Things
ML	Machine Learning
NLG	Natural Language Generation
NLP	*Natural Language Processing*
NLU	Natural Language Understanding
RNA	Redes Neuronais Artificiais

1. Introdução

O projeto DroneGest foi desenvolvido durante o segundo semestre no âmbito da unidade curricular de Projeto Informático pertencente à Licenciatura em Engenharia Informática da Escola Superior de Tecnologia e Gestão, do Instituto Politécnico de Leiria e tem como objetivo aplicar os conhecimentos adquiridos ao longo da Licenciatura.

No atual capítulo pretende-se realizar uma breve introdução ao estudo da área, mais especificamente à área do reconhecimento de gestos com recurso a sensores através de algoritmos de *Machine Learning* (ML), permitindo assim uma contextualização do trabalho realizado. Também é do nosso interesse apresentar os objetivos deste projeto e a nossa motivação para a realização do mesmo.

1.1. Contexto do Projeto

De maneira a contextualizar os assuntos abordados ao longo deste relatório iremos abordar algumas explicações sobre as principais tecnologias.

Machine learning é uma parte da inteligência artificial que se dedica a ensinar os computadores a aprenderem com dados e a melhorarem com a experiência. Em ML, são treinados algoritmos para encontrarem padrões e correlações em grandes conjuntos de dados e para tomarem as melhores decisões e fazerem as melhores previsões possíveis com base nessa análise. As aplicações de ML melhoram e tornam-se mais exatas com a utilização e com a quantidade de dados a que têm acesso.

O ESP32-C6 é um microcontrolador avançado da Espressif, que inclui um núcleo de CPU RISC-V de 32 bits e suporte para Wi-Fi 6 e Bluetooth 5.0.

Um gesto é um movimento do corpo, principalmente da cabeça e dos braços, sendo estes normalmente utilizados para exprimir ideias ou sentimentos. Existem dois tipos de gestos sendo estes os gestos estáticos e os gestos dinâmicos.(Mohammed & Waleed, 2023)

Os gestos estáticos consistem numa postura ou forma das mãos constantes ao longo do tempo podendo ser compreendidos através de apenas uma imagem. Por outro lado, os gestos dinâmicos são representados de acordo com o movimento das mãos, sendo registados como

uma sequência de imagens que apenas podem ser identificados quando analisado o contexto temporal da informação.

Para o reconhecimento destes gestos utilizamos alguns sensores ligados a um arduino, sendo estes um acelerómetro e um giroscópio incorporados num MPU6050, permitindo assim medir a aceleração e a rotação nos três eixos coordenados (x, y, z). Dado isto decidimos utilizar a representação de gestos dinâmicos, sendo a que faz mais sentido para o controlo de um drone através dos sensores mencionados.

Para podermos reconhecer qual o gesto realizado pelo utilizador utilizou-se um algoritmo de ML treinado através de um *dataset* criado por diversas pessoas. Este algoritmo permite a leitura do gesto realizado e a resposta consoante os dados enviados pelos sensores.

1.2. Motivação e Objetivos

Atualmente os comandos de controlo remoto tradicionais são a principal alternativa para o controlo de drones, embora eles sejam precisos e robustos, são também complexos e pouco intuitivos, o que leva a afastar o controlo de drones a um utilizador comum.

Após percebermos este problema decidimos aprofundar mais este tema e após uma pesquisa na internet acerca do uso de gestos para controle de drones ou outros equipamentos, concluímos que normalmente são utilizados gestos dinâmicos (Hadri, 2018)(Hu et al., 2024), mas lidos através de uma câmara. Esta opção é bastante utilizada, mas tem diversos problemas associados, como a dependência constante da câmara para a captação dos movimentos, a variação na luminosidade do local, o custo associado e outros aspetos relevantes.

Foram estas razões que nos motivou a abraçar este projeto. Ao utilizarmos sensores no lugar de câmaras, obtemos mais liberdade nos movimentos, um custo reduzido e uma velocidade superior de interpretação. Acreditamos que essa abordagem inovadora não só resolverá os problemas atuais, mas também abrirá novas possibilidades na interação homem-máquina, tornando o controle de drones mais acessível, eficiente e versátil.

O objetivo principal deste projeto é tornar o controlo de drones através de gestos algo viável, com recurso a sensores utilizando os dados por eles lidos e interpretados pelo algoritmo de ML. De modo a salientar isso, destacamos os seguintes objetivos mais específicos:

- Interpretação de Gestos: Desenvolver um modelo que tenha a capacidade de interpretar gestos com precisão a partir dos dados lidos dos sensores.
- Desenvolvimento de Algoritmo: Treinar um modelo de ML para reconhecer diferentes gestos em tempo real.
- Integração de Sensores: Utilizar sensores, mais especificamente um acelerômetro e um giroscópio para a coleta dos dados dos gestos.
- Testes e Validações: Realizar testes para garantir que o sistema funciona bem em diversas condições e com diferentes utilizadores.
- Documentação: Documentar detalhadamente o processo de desenvolvimento, resultados e possíveis melhorias.

1.3. Estrutura do relatório

No percorrer deste relatório serão apresentadas as diversas fases do projeto, divididas em 5 capítulos.

O primeiro capítulo é referente a toda a pesquisa que antecedeu ao desenvolvimento do projeto, foi fundamental porque sem esta etapa não teríamos os conhecimentos necessários para concluir o desenvolvimento do projeto, foi também fundamental para aprofundar os nossos conhecimentos na área de Inteligência Artificial e tudo o que a engloba.

O segundo capítulo contém os testes e a aplicação dos conhecimentos obtidos na fase de pesquisa, utilizando estes conhecimentos com modelos e *datasets* disponibilizados online para treino.

O terceiro capítulo foi onde começamos a criação do nosso *dataset* de gestos, não só a criação como também definimos quais os gestos que iriamos utilizar durante o desenvolvimento do projeto.

O quarto capítulo é o que contém a fase de treino dos modelos, foram treinados diversos modelos com os dados do *dataset* criado no capítulo anterior, devido a termos treinados vários modelos conseguimos obter vários resultados diferentes.

O quinto e último capítulo é referente a finalização da etapa de implementação, escolhendo o melhor modelo e concluindo o *dataset*. Foi também nesta etapa que desenvolvemos o protótipo para demonstrar o funcionamento do drone através do reconhecimento de gestos.

2. Pesquisa Preliminar

Neste capítulo estão presentes as várias fases da pesquisa realizada antes de iniciar o desenvolvimento do projeto. Esta pesquisa foi bastante benéfica de modo a conseguirmos nos enquadrar melhor no âmbito do projeto. Também nos proporcionou uma base e ponto de partida naquilo que é a inteligência artificial num todo, mas principalmente algoritmos de ML e redes neuronais para interpretação de dados, neste caso gestos.

2.1. Gestos Estáticos e Dinâmicos

Os gestos podem ser classificados em estáticos e dinâmicos, podendo para isto ser usada qualquer parte do corpo, porém vamos apenas abordar o uso das mão e braços para o efeito, visto que é apenas nessa área que se situa o tema do nosso projeto.

Os gestos estáticos referem-se à postura ou forma das mãos, sem qualquer deslocação, movimento ou rotação da mão, enquanto os dinâmicos estão relacionados á deslocação das mãos ao longo do tempo, causando uma sensação de movimento, ambos os gestos com um significado.

Os gestos estáticos podem ser compreendidos instantaneamente por uma única representação, imagem, leitura de dados entre outros, enquanto os dinâmicos são uma sequência dos mesmos, que exigem uma análise do contexto temporal para serem compreendidos.

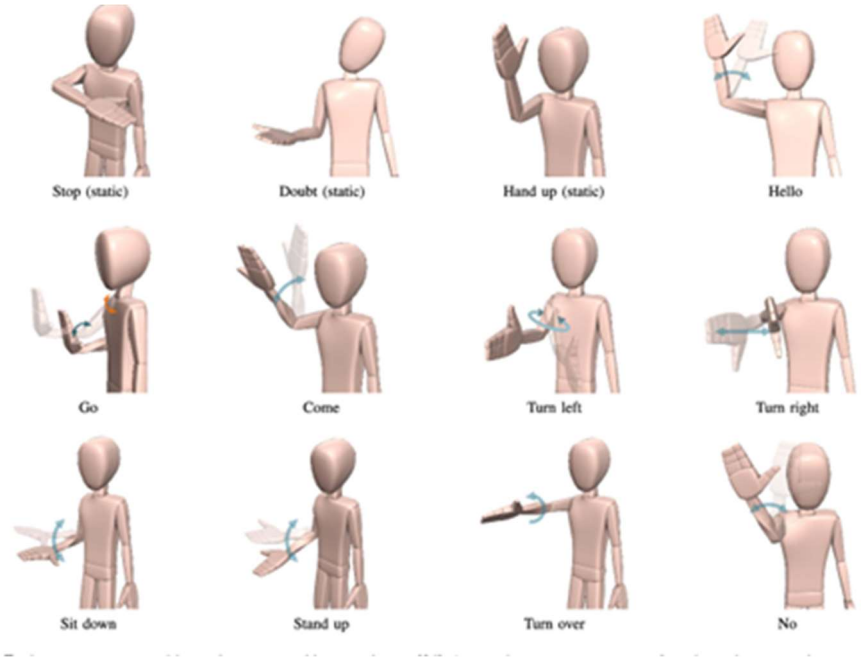

Figura 1 - Gestos dinâmicos

Assim, torna se muito difícil o reconhecimento de gestos estáticos numa abordagem como a nossa, em que apenas usamos sensores de posição, que capturariam, num cenário perfeito, sempre os mesmos dados pois a mão estaria sempre na mesma posição. Será então mais viável o uso de gestos dinâmicos, pois através da captura de dados ao longo do tempo, teremos a perceção de movimento e a variação da posição da mão, que poderão ser depois tratados e alimentados ao modelo como dados de entrada.

(Matheus Silva Pereira, 2022)

2.2.Machine Learning

Durante a pesquisa acerca de ML e o seu funcionamento obtivemos respostas que nos proporcionaram um melhor entendimento dos algoritmos de Inteligência Artificial.

Esta pesquisa foi dedicada a perceber os termos técnicos e estrutura correta de um *dataset* para a aplicação de um algoritmo de ML, o que nos iria facilitar tanto a nível de elaboração do *dataset*, como a nível de desenvolvimento dos algoritmos em questão.

2.2.1. O que é e Como Treinar um Modelo de ML

Um modelo de ML é um algoritmo que aprende padrões a partir de dados. Este modelo utiliza esses padrões para fazer previsões ou tomar decisões sem ser explicitamente programado para tal. O processo de aprendizagem pode ser supervisionado, não supervisionado ou semi-supervisionado, dependendo da disponibilidade de dados etiquetados.

O treino envolve várias etapas fundamentais. Primeiro, recolhe-se e prepara-se um conjunto de dados de qualidade, que é dividido em dados de treino e de teste. Em seguida, escolhe-se um algoritmo de aprendizagem apropriado, como regressão linear, árvores de decisão ou redes neuronais. O modelo é então treinado alimentado através do *dataset* anteriormente preparado e devidamente separado com os dados de treino, permitindo-lhe aprender padrões e relações entre os dados. Durante este processo, ajustam-se os Hiper parâmetros do modelo para otimizar e melhorar o seu desempenho. Após o treino, o modelo é validado e avaliado usando os dados do *dataset* anteriormente separados para teste, dados estes que o modelo nunca terá visto, para garantir que generaliza bem para novos dados. Se necessário, são feitos ajustes adicionais. Finalmente, o modelo treinado pode ser usado para realizar previsões ou classificações em novos dados.

2.2.2. Hiper parâmetros mais Importantes no Treino de um Modelo

No treino de um modelo de ML, os hiper parâmetros mais importantes incluem:

Taxa de aprendizagem (*learning rate*): controla o ajuste dos pesos do modelo ao longo do treino de modo a obter um nível de precisão mais elevado

Número de épocas (*epochs*), que define quantas vezes o algoritmo irá percorrer o conjunto de dados de treino até ser dado como concluído o treino

Tamanho do *batch* (*batch size*): define o número de amostras processadas antes da atualização do modelo

Função de ativação: impacta a não-linearidade do modelo.

Dropout: Técnica de regularização que desativa aleatoriamente uma fração das unidades de uma camada durante o treino para prevenir o *overfitting*.

Dense: Camada totalmente conectada onde cada neurónio está ligado a todos os neurónios da camada anterior, comum em redes neuronais artificiais.

Optimizer: Algoritmo que ajusta os pesos do modelo para minimizar a função de perda, como SGD, Adam ou RMSprop.

Reduce LR on Plateau: Técnica que reduz a taxa de aprendizagem quando a performance do modelo estagna, ajudando a encontrar um mínimo local.

Early Stopping: Método que interrompe o treino quando a performance no conjunto de validação não melhora após um número definido de épocas, evitando *overfitting*.

O ajuste de cada um destes parâmetros cuidadosamente é crucial para alcançar o melhor desempenho possível, equilibrando a capacidade de aprendizagem e a generalização do modelo.

2.2.3. *Neuronal Network*s

*Neuronal Network*s, ou em português, redes neuronais são sistemas de computação inspirados nas redes neuronais biológicas que constituem os cérebros dos animais. Tal como no cérebro, onde muitos neurónios se interligam e formam redes, uma rede neuronal artificial (RNA) é constituída por muitas camadas. Cada camada é um conjunto de uma série de neurónios. Uma RNA pode processar dados consecutivamente, o que significa que apenas a primeira camada está ligada aos dados de entrada. Quando as camadas se tornam muito grandes, o modelo torna-se um modelo de aprendizagem profunda. É difícil definir uma RNA com um determinado número de camadas. Há 10 anos, as RNA com apenas 3 camadas eram suficientemente profundas, atualmente são necessárias 20 camadas para maior parte dos problemas da atualidade.

As Redes Neuronais têm muitas variantes, as mais utilizadas são:

Rede Neuronal Convolucional - fez grandes avanços na visão computacional

Rede neuronal recorrente - criada para processar dados com características sequenciais, como texto e preços de ações.

Rede totalmente conectada - é o modelo mais fácil utilizado para processar dados estáticos/tabulares.

NLP (*Natural Language Processing*): O Processamento de Linguagem Natural, ou abreviadamente PNL, é um ramo da inteligência artificial que permite às máquinas compreender a linguagem humana e incorporá-la em todo o tipo de processos.

Algumas aplicações bem conhecidas da PNL incluem:

Classificação e ordenação de textos: Com a Internet, o problema crescente da sobrecarga de informação, os grandes volumes, a fraca estrutura e o ruído dos dados da Web tornam-nos passíveis de aplicação de técnicas de aprendizagem automática. É por isso que a classificação e ordenação de textos se torna cada vez mais relevante. Uma aplicação simples desta técnica é a seleção de correio eletrónico não solicitado através da análise do texto do correio. A nível empresarial, pode ser utilizada para identificar e extrair informações relacionadas com a concorrência, trabalho realizado normalmente pelo departamento de BI (*Business Inteligence*).

Análise de sentimentos: A análise de sentimentos, conhecida como extração de opiniões ou IA de emoções, permite a um computador decifrar sentimentos como a raiva, a tristeza e a alegria através da análise de cadeias de texto. A análise de sentimentos é amplamente aplicada a materiais de voz do cliente, tais como críticas e respostas a inquéritos, meios de comunicação social e online e materiais de cuidados de saúde para aplicações que vão do marketing ao serviço ao cliente e à medicina clínica.

Extração de informação: A extração de informação (IE) é a tarefa de extrair automaticamente informação estruturada de fontes textuais não estruturadas ou semiestruturadas. Funciona como um processo para resumir um parágrafo longo num texto curto, tal como criar um resumo.

Reconhecimento do discurso: O reconhecimento do discurso, também conhecido como reconhecimento automático do discurso (ASR), reconhecimento do discurso por computador ou conversão em texto, é uma capacidade que permite a um programa processar o discurso humano num formato escrito. Não deve ser confundido com o reconhecimento de voz, que apenas procura identificar a voz de um utilizador individual. Um ótimo exemplo disto é o Siri da Apple.

Compreensão e produção de linguagem natural (NLU & NLG): Existem três conceitos de processamento da linguagem natural e já os mencionámos todos. A um

nível elevado, NLU e NLG são apenas componentes da PNL. Dado o modo como se intersectam, são normalmente confundidos nas conversas. Para definir os termos individualmente, a NLU é a utilização da análise sintática e semântica do texto e da fala para determinar o significado de uma frase, enquanto a NLG é o processo de produção de uma resposta de texto em linguagem humana com base em alguns dados introduzidos. Esta tecnologia é muito utilizada para a comunicação humana com robôs.

(Ansel Barret, 2022)

Tradução automática: A tradução automática é o processo de utilização da inteligência artificial para traduzir automaticamente conteúdos de uma língua para outra sem qualquer intervenção humana.

(SAP, n.d.)

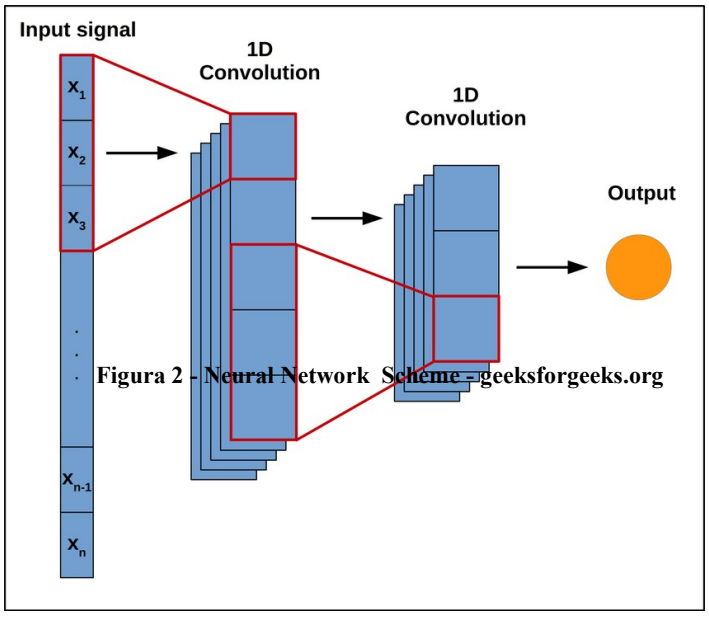

Figura 2 - Neural Network Scheme - geeksforgeeks.org

Figura 2 - Ilustração de uma rede neuronal de 1 dimensão

2.2.4. Avaliação do Modelo

Para a avaliação do modelo utilizamos várias métricas que nos permitem medir a qualidade e o desempenho do mesmo. Assim, utilizaremos as seguintes:

Precision (Precisão) é a proporção de verdadeiros positivos em relação ao total de previsões positivas. Mede a qualidade das previsões positivas do modelo.

Accuracy (Acurácia) é a proporção de previsões corretas (verdadeiros positivos e verdadeiros negativos) em relação ao total de previsões. Indica a precisão geral do modelo.

Recall (Sensibilidade) é a proporção de verdadeiros positivos em relação ao total de amostras positivas reais. Mede a capacidade do modelo de identificar corretamente as amostras positivas.

F1-score é a média harmónica entre a precisão e o *recall*, oferecendo uma métrica única que equilibra ambos. É útil quando desejamos um equilíbrio entre precisão e *recall*.

Macro Average (Média *Macro*) calcula a média da precisão, *recall* e *F1-score* de todas as classes, tratando todas as classes de forma igual, independentemente da sua frequência.

Weighted Average (Média Ponderada) calcula a média da precisão, *recall* e *F1-score* de todas as classes, ponderando cada classe pela sua frequência no conjunto de dados.

Curvas ROC (Receiver Operating Characteristic) são gráficos que avaliam o desempenho de classificadores binários mostrando a relação entre a taxa de verdadeiros positivos e a taxa de falsos positivos para diferentes limiares de decisão. O eixo Y representa a proporção de amostras positivas corretamente identificadas, enquanto o eixo X representa a proporção de amostras negativas incorretamente classificadas. A área sob a curva ROC (AUC) é uma métrica que resume a *performance* global do modelo, onde um valor mais próximo de 1 indica um melhor desempenho.

Loss Function (Função de Perda)

A função de perda mede a diferença entre as previsões do modelo e os valores reais dos dados. O objetivo do treino do modelo é minimizar esta perda. Existem diferentes funções de perda dependendo do tipo de problema (ex: MSE para regressão, Cross-Entropy para classificação).

Estes conceitos são fundamentais para avaliar e comparar a qualidade e desempenho de modelos de ML de forma eficaz e precisa.

2.2.5. Dificuldades Encontradas

Durante esta etapa viemos a encontrar algumas dificuldades na pesquisa, após termos definido a tecnologia a utilizar, procurámos diversos exemplos pela web em busca de *datasets* similares aquele que seria, mais tarde, o nosso próprio *dataset* para o problema. Serviria este *dataset* para que pudéssemos aplicar o conhecimento adquirido na prática e começar a ganhar alguma noção e sensibilidade naquilo que é o básico e o fundamental na área da inteligência artificial. O grande problema é que quase todos os resultados que obtemos e encontramos nas nossas pesquisas eram de reconhecimento de gestos através de câmaras ou de sensores de movimento afixados numa plataforma, direcionados para um centro de modo a reconhecer os gestos através de frequências. Ambos estes *datasets* não nos iriam ser úteis, apesar da ideia ser semelhante o desenvolvimento e a execução de todas as etapas ia variar consideravelmente pelo que não se aplicava ao desenvolvimento pretendido.

Em primeiro lugar, os *datasets* baseados em câmaras exigem um processamento totalmente diferente, no caso processamento de imagem e algoritmos de *Computer Vision* específicos, estes diferem bastante dos algoritmos para trabalhar com os sensores de movimento. A preparação e pré processamento dos dados visuais envolveriam etapas como segmentação de imagem, extração de características visuais e o uso de CNN's adaptados para deteção de padrões em imagens, no caso, CNN 2D.

Por outro lado, os *datasets* que utilizam os sensores fixados numa plataforma apresentam desafios diferentes. De certa forma incluíam sensores como o acelerômetro e o giroscópio que iremos usar, mas para além destes referidos utilizam sensores de ultrassom, exigindo assim um processamento de dados diferente e com restrição de amplitude e de campo de captura de dados, o que faz com que, neste cenário, apenas nos pudéssemos expressar no campo de visão projetado para o efeito, em vez de livremente. O que acontece nestes casos é a captura controlada de dados, em cenários específicos e previamente configurados.

Assim sendo, não se enquadrava novamente naquilo que procurávamos, a nossa necessidade era encontrar *datasets* adequados ao nosso contexto, envolvendo uma combinação de tecnologias e sensores, mas sensores estes que não estariam fixados, mas sim sendo amovíveis e de livre movimentação. A adaptação dos *datasets* anteriormente mencionados iria ser trabalhosa e ineficaz, uma vez que nos dados recolhidos, o método de treino e a validação do modelo ia diferir substancialmente.

Por fim, após bastante pesquisa e em discussão com os professores orientadores, conseguimos chegar á pesquisa certa, que ainda não sendo a ideal, era um exemplo similar ao que pretendíamos, com o uso de sensores amovíveis e algoritmo CNN 1D, onde podemos aplicar e praticar os conhecimentos aprendidos.

(SAP, n.d.)

2.3. Microcontrolador e Sensores

Esta secção foca-se na apresentação do microcontrolador e sensores utilizados no âmbito do projeto.

2.3.1. Microcontrolador

O ESP32-C6 é um microcontrolador avançado da Espressif, que inclui um núcleo de CPU RISC-V de 32 bits e suporte para Wi-Fi 6 e Bluetooth 5.0. Os principais componentes do ESP32-C6 incluem um módulo Wi-Fi e Bluetooth dual-band, um subsistema de processamento digital de sinais, memória RAM interna, memória Flash embutida, múltiplas interfaces de comunicação (como SPI, I2C, UART e CAN), um conversor analógico-digital (ADC), e GPIOs (General-Purpose Input/Output) configuráveis. Também possui mecanismos de segurança, como criptografia de hardware e um gerador de números aleatórios verdadeiros (TRNG), tornando-o ideal para aplicações de IoT que exigem conectividade robusta, eficiência energética e segurança.

Para que o microcontrolador possa começar a ler os dados e enviar de volta para o nosso computador, precisamos de seguir alguns passos.

Configurar o ambiente de desenvolvimento: Utilizando ferramentas como a IDE Arduino, com suporte ao ESP32C6 DEV MODULE, que será usada neste projeto, podemos fazer a transferência e instalação da biblioteca MPU6050, da Expressif e definir a porta COM a usar, que deverá estar entre a COM1 e a COM6.

Escrever o código: Para inicializar o sensor, ler os dados e enviar esses dados para a *serial console*, precisamos de usar algumas bibliotecas que deverão ser instaladas para controlo de leds e sensores, assim como escrito o código necessário para fazer uso das mesmas e recolher os dados, assim como apresentá-los.

Compilar o código e carregar para o ESP32: Através da IDE, é necessário definir ainda a velocidade de comunicação e outras configurações relacionadas com a comunicação entre o computador e o microcontrolador. Após a correta configuração destes parâmetros fazemos o carregamento para o equipamento através da IDE e os dados deverão começar a aparecer na consola serial.

Para a conexão do sensor não iremos utilizar o Bluetooth por duas razões, estas são, instabilidade e falta de alinhamento com o foco do projeto. Primeiramente o Bluetooth ainda está instável no ESP32-C6, o que poderia resultar em conexões frágeis e inconsistentes, confiar nesta tecnologia poderia comprometer a eficiência e fiabilidade do projeto. Em segundo o foco do projeto é em conexão wireless, que é a mais adequada para as nossas necessidades. O Wi-Fi oferece um maior alcance e velocidade, o que seria ideal para atender aos requisitos do nosso projeto. Apesar da conexão wireless ser a ideal para o projeto, iremos manter a conexão via cabo, isto porque as conexões wireless estão no início e com diversos problemas, coisa que iria prejudicar o nosso trabalho, então de modo a amenizar esse problema e a ter um projeto consolidado optámos pelo uso do cabo.

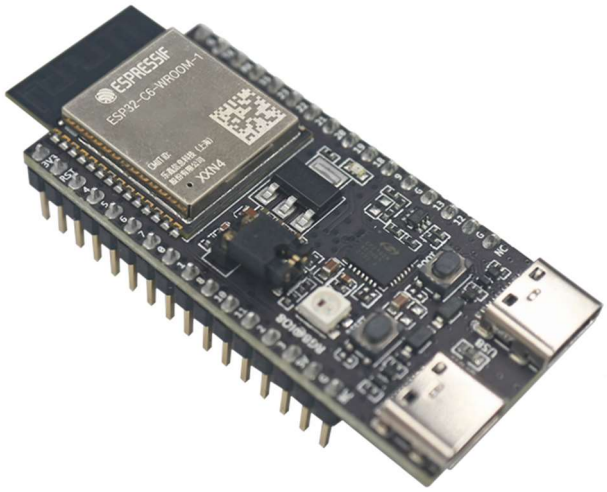

Figura 3 - Microcontrolador ESP32C6 - espressif-docs.readthedocs-hosted.com

2.3.2. Inertial Measurement Unit

Uma inertial measurement unit (en: IMU) ou, unidade de medição inercial é um componente eletrónico pertencente à família dos sensores. Utiliza uma combinação de acelerómetros, giroscópios e magnetómetros para medir a aceleração, a velocidade angular e a orientação do sensor.

Estes sensores devolvem os dados *orientation_X, orientation_Y, orientation_Z* e *orientation_W* do *Magnetometro, angular_velocity_X, angular_velocity_Y* e *angular_velocity_Z* do *Giroscopio* e *linear_acceleration_X, linear_acceleration_Y* e *linear_acceleration_Z* do *Acelerometro*.

Dois exemplos de IMUs são os mpu6050 e mpu9250, que tivemos ao nosso dispor, disponibilizados pelos professores orientadores do projeto.

O mpu9250 é composto pelos 3 sensores acelerómetro, giroscopio e magnetometro, que tivemos oportunidade de testar, já o mpu6050 é composto apenas pelo acelerómetro e giroscópio.

Optamos por prosseguir com o mpu6050 por uma questão de simplicidade e otimização pois não seria necessário saber a orientação do utilizador para o problema em questão, dado este obtido através do magnetómetro. Deste modo, seguimos com o sensor que acopla o acelerómetro e giroscópio juntamento com o ESP32C6 previamente abordado.

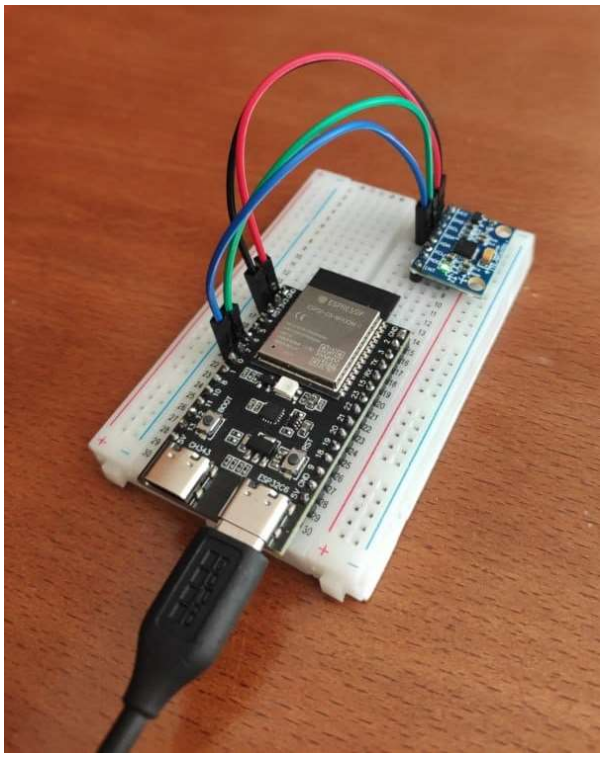

Figura 4 - Setup microcontrolador e sensores

(Fábio Guimarães, 2018),(Leandro Castro, 2021)

2.4.Conjuntos de Dados (*datasets*)

2.4.1. Definição de *dataset*

Um *dataset*, ou conjunto de dados, é uma coleção organizada de informações, geralmente estruturada em formato tabular, onde cada coluna representa uma variável específica e cada linha corresponde a uma observação ou amostra distinta. Estes dados podem ser utilizados para análises estatísticas, treino de modelos de inteligência artificial, ou para diversos outros tipos de pesquisas e exploração de dados. Os *datasets* podem variar em tamanho e complexidade, desde pequenas tabelas com poucas entradas até grandes coleções de dados compostas por milhões de registos, frequentemente armazenadas em bases de dados ou ficheiros CSV. A qualidade e a organização dos dados num *dataset* são cruciais para a precisão e a validade dos modelos e analises subsequentes.

Training dataset: Os Training *dataset* são um conjunto de exemplos usados para ajustar os parâmetros do modelo de ML na etapa de treino, o modelo será capaz de reconhecer as características importantes do conjunto de dados.

Validation dataset: O *validation dataset* é usado para cortar os coeficientes dos modelos e comparar modelos para escolher o ideal. O conjunto de dados de validação é diferente do Training *dataset* e não pode ser usado na seção de treinamento, deste modo, o modelo será sempre validade através de novos dados. Caso contrário, poderá ocorrer *overfitting* e afetar adversamente a geração de novos dados.

Test dataset: Assim que o modelo for finalizado, o *Test dataset* é usado para testar e avaliar o desempenho do modelo quando proposto a prever um novo conjunto de dados nunca visto. Este *dataset* pode ser desde dados previamente capturados como dados introduzidos ao modelo em *real time.*

Estes *datasets* são aliados a diversas técnicas aquando elaboração do algoritmo para treino do modelo, cujo normalmente, tendem a chamar ao conjunto de treino x_train e y_train e ao conjunto de teste/validação x_val e y_val, onde x serão os exemplos e y as *labels*/respostas a esses exemplos.

2.4.2. Criação de D*atasets*

Para criar um *dataset* próprio, é essencial seguir algumas técnicas e cuidados para garantir a qualidade dos dados. Primeiramente, é recomendável armazenar uma quantidade significativa de dados em diferentes condições e contextos de uso do sensor, garantindo diversidade e consistência. Durante a recolha, é importante verificar e corrigir eventuais *outliers* ou dados inconsistentes que possam comprometer a precisão do *dataset*. Os dados devem representar todos os cenários possíveis, ideais e pessimistas incluindo uma versão de *noise*, que é o equivalente a não fazer nada, para que o modelo possa perceber que não estamos em constante execução de movimentos.

2.4.3. Estrutura do *Dataset*

Quanto á estrutura do *dataset*, encontramos vários tipos de construção do mesmo em que aplicado ao projeto que pretendemos desenvolver e com consulta de outros *datasets* existentes na internet similares ao que pretendíamos, decidimos optar pelo *dataset* em ficheiro no formato CSV (*Comma-separated values file*), onde cada valor é delimitado por uma virgula.

Assim, a primeira linha representa o *header*, contendo os nomes dos atributos e as seguintes linhas os valores para esses atributos, á semelhança de colunas numa base de dados, mas separados por virgulas.

2.5.Tecnologia a Utilizar

Após iniciarmos a pesquisa sobre as tecnologias atuais e termos realizado uma série de pesquisas, deparámo-nos com 2 das maiores tecnologias na área do ML, para o reconhecimento de padrões. São elas o PyTorch e o TensorFlow.

Para as nossas necessidades entendemos que precisávamos de uma tecnologia com uma linha de aprendizagem moderada, com bastante documentação e que pudesse, ao mesmo tempo, ser leve para ser executada a partir de microcontroladores.

Isto traduz se nos seguintes pontos:

- Complexidade do Modelo
- Compatibilidade com o Hardware

- Facilidade de Uso e Documentação
- Tamanho e Velocidade de Execução

2.5.1. TensorFlow e TensorFlow Micro

Para além do mencionado anteriormente, a biblioteca TensorFlow oferece uma vasta documentação e uma comunidade ativa, tornando-se mais fácil o desenvolvimento e a resolução de possíveis problemas/erros. O TensorFlow tem também uma incrível capacidade de escalabilidade e performance, sendo possível o processamento eficaz de grandes volumes de dados e a execução de modelos criados e treinados. Outra vantagem é a compatibilidade que o TensorFlow tem com as diversas plataformas e linguagens de programação, fazendo com que seja uma ferramenta adaptada às nossas necessidades.

TensorFlow Micro é uma versão do TensorFlow Lite otimizada para dispositivos microcontroladores, como o Arduino e alternativas como o ESP32. Ele permite executar modelos de Inteligência Artificial diretamente nestes dispositivos, o que é ideal para aplicações de reconhecimento de gestos. Funciona de forma que o modelo para reconhecimento de gestos é construído usando TensorFlow e, em seguida, convertemo-lo para o formato TensorFlow Lite Micro, fase em que já pode ser executado nos microcontroladores. (*TensorFlow*, n.d.)(*TensorFlow Lite Para Microcontroladores*, n.d.)

2.5.2. PyTorch

O PyTorch é conhecido pela sua flexibilidade e expressividade, características úteis para quem experimenta diferentes arquiteturas de redes neuronais ou técnicas de treinamento. A popularidade do PyTorch em ambientes de pesquisa deve-se à sua interface mais amigável e flexível, tornando-o a escolha preferida para explorar técnicas de última geração em aprendizagem máquina. Embora o ecossistema do PyTorch para microcontroladores possa não ser tão completo quanto o de TensorFlow Lite, a comunidade ativa e crescente em torno do PyTorch facilita encontrar suporte e recursos adicionais.

Após as análises anteriores concluímos que a escolha acertada seria utilizar a biblioteca TensorFlow, isto devido a ter um amplo suporte ao nível dos dispositivos microcontroladores que iriamos utilizar para o reconhecimento e interpretação de gestos, como por exemplo o

ESP32C6. Com as suas extensões TensorFlow Lite e TensorFlow Micro conseguimos uma compatibilidade que nos seria vantajoso numa fase mais avançada do projeto. (*PyTorch*, n.d.)

2.5.3. Redes Neuronais Convolucionais

CNN é a sigla para Convolutional *Neural Network*, ou em português, Rede Neuronal Convolucional. É um tipo de rede neuronal especialmente projetada para processar dados com estrutura de grade, também conhecido por jogo da velha, como imagens. As CNNs têm sido amplamente utilizadas em tarefas de visão computacional, como reconhecimento de imagem, deteção de objetos e segmentação de imagens, devido ao seu grande potencial na captura de padrões nos dados.

As CNNs são compostas por várias camadas diferentes, incluindo:

Camada de Convolução: Esta camada aplica filtros (também conhecidos como *kernels*) à entrada de dados para extrair características importantes, como bordas, texturas e padrões.

Camada de *Pooling*: A camada de *Pooling* reduz a dimensionalidade da imagem, mantendo as características mais importantes. Isso é feito agrupando regiões de pixels num único valor representativo, reduzindo assim o tamanho da imagem e aliviando os recursos da máquina.

Camada de Ativação: Após a convolução e a operação de *Pooling*, é comum aplicar uma função de ativação, como ReLU (*Rectified Linear Unit*), para introduzir não linearidade na rede.

Camada *Fully Connected* (Totalmente Conectada): No final da rede, geralmente há uma ou mais camadas densas totalmente conectadas, que processam as características extraídas pela rede convolucional e transformam-nas numa saída final, como as probabilidades associadas a diferentes classes num problema de classificação.

As CNNs são altamente eficazes em tarefas de visão computacional devido à sua capacidade de capturar padrões locais e hierárquicos, o que lhes permite aprender representações mais discriminativas dos dados. Elas são fundamentais em muitas aplicações modernas, incluindo reconhecimento facial, classificação de imagens, diagnóstico médico por imagem, entre outros.

Existem 3 principais tipos de CNNs, diretamente relacionados com a sua dimensionalidade e o uso ao qual serão associadas. Assim, temos as CNN de 1 dimensão, 2 dimensões e 3 dimensões.

CNN 1D: Lidam com dados unidimensionais, como séries temporais. São utilizadas no processamento de sinais de áudio, dados financeiros, análise de ADN, entre outros. Funcionam aplicando filtros de convolução ao longo da sequência de dados, extraindo características locais. Camadas de *Pooling* reduzem a dimensionalidade dos mapas de características, e as saídas são passadas para camadas totalmente conectadas para classificação ou predição. Geralmente utilizado para analise de dados sequenciais.

CNN 2D: São aplicadas a dados bidimensionais, como imagens. São amplamente usadas em visão computacional, incluindo reconhecimento de imagem, deteção de objetos e segmentação de imagens. Funcionam com filtros de convolução que deslizam ao longo das duas dimensões da imagem, capturando características espaciais como bordas e texturas. Camadas de *Pooling* reduzem a dimensionalidade, e as saídas são processadas por camadas totalmente conectadas para classificação ou outras tarefas.

CNN 3D: Atuam em dados tridimensionais, como vídeos ou imagens médicas 3D, como raioX e TACs. São utilizadas em tarefas como análise de vídeos, segmentação de imagens médicas e deteção de objetos em 3D. Os filtros de convolução deslizam ao longo das três dimensões dos dados, capturando características volumétricas. Camadas de *Pooling* reduzem a dimensionalidade dos mapas de características, e as saídas são processadas por camadas totalmente conectadas para realizar a classificação ou outras tarefas específicas.

Concluímos que o tipo de CNN adequado ao problema seria uma CNN de uma dimensão, pelo que pesquisamos mais aprofundadamente sobre esta técnica onde ficamos a entender como funciona, detalhadamente e os passos necessários para o treinamento de modelos:

Entrada: A CNN 1D recebe uma sequência de dados unidimensional, por exemplo, uma série temporal com n pontos de dados.

Camada Convolucional: Um conjunto de filtros (ou *kernels*) é aplicado à sequência. Cada filtro desliza ao longo da sequência, aplicando convoluções que resultam em novos conjuntos de dados sequenciais (mapas de características). A convolução realça padrões nas sequências.

Camada de Ativação: Após a convolução, uma função de ativação (como ReLU) é aplicada para introduzir não-linearidade no modelo.

Camada de *Pooling*: Uma operação de *Pooling* (geralmente max *Pooling*) é aplicada para reduzir a dimensionalidade das características extraídas, reduzindo a sensibilidade, deslocamentos e variações na sequência.

Camadas Adicionais: O processo de convolução e *Pooling* pode ser repetido várias vezes de modo a extrair características de níveis mais altos.

Camadas Totalmente Conectadas: Após várias camadas convolucionais e de *Pooling*, as características extraídas são passadas para uma ou mais camadas totalmente conectadas (*fully connected layers*) que realizam a tarefa de classificação ou predição.

Saída: A última camada totalmente conectada produz a saída final, que pode ser uma classificação, um valor contínuo uma previsão, entre outras.

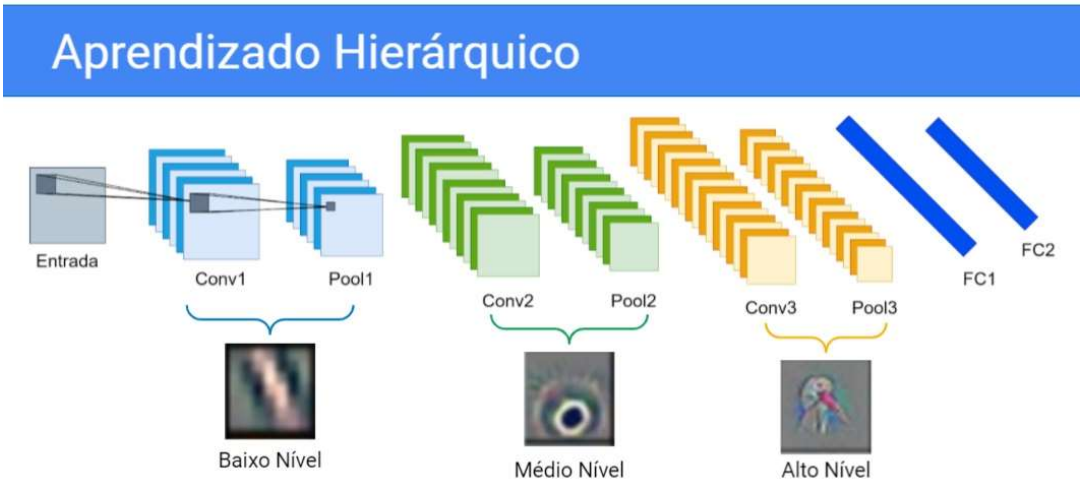

Figura 5 - Funcionamento de uma CNN - alura.com.br

(*Convolutional Neural Network (CNN) in Machine Learning - GeeksforGeeks*, n.d.)

2.6.Pré Processamento dos Dados

Esta foi uma pesquisa fulcral para o desenvolvimento deste projeto, onde podemos concluir os requisitos para interpretar os dados registados.

Com esta pesquisa para o desenvolvimento do sistema de reconhecimento de gestos encontramos alguns desafios relacionados ao pré processamento dos dados. Neste ponto vamos abordar o uso de coordenadas de *quaternions* para a orientação, mencionando a importância e as vantagens dos ângulos de Euler. Para além disso, vamos mostrar a relevância da orientação no eixo Z.

2.6.1. Coordenadas de *Quaternions*

As coordenadas de *quaternions* são compostas por quatro coordenadas: X, Y, Z e W. Ao contrário dos ângulos de Euler (X, Y, Z), os *quaternions* não têm problemas como o "*gimbal Lock*", fenómeno este que acontece quando o ângulo de inclinação se aproxima dos 90 graus. O uso da matemática envolvida nos *quaternions* é mais complexa que a dos ângulos de Euler, mas oferece uma solução robusta para medir a orientação.

Vantagens dos *quaternions*:

- Eliminação do *Gimbal Lock*: O *gimbal Lock* é uma limitação dos ângulos de Euler que ocorre quando o ângulo da inclinação atinge cerca de 90 graus, resultando assim na perda de um grau de liberdade. Os *quaternions* não têm problemas com esta limitação, tornando assim possíveis medições contínuas e precisas da orientação em todas as direções.
- Interpolação Suave: Os *quaternions* permitem a interpolação suave das rotações, o que é essencial para as transições suaves em aplicações de robótica e realidade virtual.
- Eficiência Computacional: As operações com *quaternions*, como a multiplicação e a normalização são eficientes e evitam problemas de singularidade associados aos ângulos de Euler.

2.6.2. Normalização dos *Quaternions*

A normalização dos *quaternions* é necessária de modo a corrigir os erros de precisão ocorrentes durante as operações matemáticas. Normalizar os *quaternions* garante que eles permaneçam de comprimento unitário o que é essencial para manter a precisão nas medições obtidas de orientação.

Importância da normalização:

- Correção de erros de precisão: Durante as operações matemáticas como a multiplicação dos *quaternions*, poderá originar pequenos erros de precisão. Ao usarmos a normalização dos *quaternions* garantimos que eles continuem a representar uma rotação válida.
- Manutenção de estabilidade: Nos sistemas de controlo de voo os *quaternions* não normalizados podem resultar em comportamentos instáveis. Ao haver uma normalização regular mantemos a estabilidade e a precisão do nosso sistema.
- Desempenho: Apesar de a normalização adicionar um passo adicionar nas operações é essencial de forma a evitar o agrupamento de erros que poderão vir a comprometer a precisão do sistema a longo prazo.

(Sebastian Krysmanski, n.d.)

2.6.3. Importância da Coordenada Z (Ângulo de Euler Z)

A orientação no eixo do Z é crítica, especialmente nos sistemas robóticos que operam em diversas superfícies. Se pensarmos num drone a voar por áreas urbanas ou ambientes rurais poderá haver diversas variações de altitude e na direção do vento, podendo causar desvios na trajetória do drone, exigindo assim o controlo preciso da orientação no eixo do Z.

Relevância no contexto dos drones:

- Adaptação a condições de voo variáveis: As condições de voo variáveis assim como as correntes de ar e obstáculos podem vir a causar desvios na trajetória do drone. Se houver uma orientação precisa no eixo do Z ajuda o drone a compensar essas variações mantendo-o num voo estável.
- Manutenção do equilíbrio: Durante as manobras e mudanças rápidas de direção, a orientação Z ajuda a manter o equilibro do drone, prevenindo assim quedas ou movimentos indesejados.

- Navegação precisa: Nos sistemas de navegação autônoma, a precisão na orientação no eixo do Z é fundamental para o cálculo de rotas evitando assim obstáculos de forma eficiente.

2.6.4. Conclusões sobre o Pré Processamento de Dados

Podemos concluir com esta pesquisa que é preferível o uso *quaternions* para o pré processamento de dados ao uso dos ângulos de Euler devido à inexistência do "*gimbal Lock*" e à sua maior precisão. Também realçar a normalização dos *quaternions*, apesar de não sempre necessária é recomendada de forma a evitar erros de precisão.

A compreensão de ambas estas técnicas são fundamentais para o desenvolvimento do sistema de controlo de drones, conseguindo assim a adaptação a condições de voo variáveis e garantido um desempenho confiável.

Ainda assim, decidimos utilizar os *Eulers Angles* em vez dos *quaternions* por estes serem unidades de medida mais simples de interpretar o que se iria traduzir numa maior precisão aquando da precisão de gestos por parte do modelo. Também pela questão de simplicidade optámos pela utilização do IMU MPU6050 no lugar do MPU9250 devido a não ser relevante a orientação do utilizador ou dos gestos.

Foi também limitado os dados de cada coordenada a 2 casas decimais, pois numa comparação de desempenho de modelos percebemos que limitar as 2 casas decimais seria o cenário mais otimizado.

(DANIEL FILIPE BAPTISTA FERREIRA DA SILVA, 2021)

3. Testes Preliminares

Neste capítulo iremos passar pelo processo de testes e aplicação dos conhecimentos adquiridos anteriormente na pesquisa preliminar ao desenvolvimento com recurso a *datasets* e exemplos de algoritmos de ML disponíveis na plataforma *Kaggle*. Segue-se uma explicação de como foi a primeira ronda de experiências, onde pudemos ganhar um maior "à vontade" com o uso de *datasets*, algoritmos de ML e toda a dinâmica de treinamento de modelos.

Esta etapa foi fundamental para a escolha e afirmação da tecnologia e modelos que iriamos usar.

3.1. Dificuldades a Encontrar um *Dataset* Compatível

Para começarmos a entender melhor, na prática, como funcionava a criação de *datasets* e modelos com CNN de uma dimensão, fomos á procura de projetos similares na plataforma *Kaggle* (*Kaggle*, n.d.), dos quais pudéssemos aprender mais sobre o tema e utilizar para testes o *dataset* e algoritmo desses projetos e experimentar nas nossas máquinas a fim de entender melhor o funcionamento e a estrutura dos mesmos.

A pesquisa por projetos similares ao nosso foi uma das grandes dificuldades iniciais, pois quando procuramos por identificação e reconhecimento gestual, apercebemo-nos que na maior parte dos materiais disponíveis na web são utilizados camaras e usada a tecnologia CNN 2D. Dito isto, a pesquisa por um projeto semelhante ao que pretendíamos foi alvo de bastante esforço e tempo despendido, dezenas de projetos consultados em que nada tinham a ver com o uso de IMU e por vezes, quando a metodologia não era o uso de câmaras e sensores de movimento, era usada uma tecnologia laser, o que também não viria ao encontro do pretendido.

3.2. *Dataset* de Reconhecimento de Superfícies

Após uma extensa pesquisa conseguimos encontrar um repositório com um projeto que realmente coincidia com o nosso objetivo, ainda que com um tema diferente, mas que seria o suficiente para que pudéssemos testar e praticar.

Este projeto teria como objetivo a identificação da superfície a partir de dados recolhidos pelos sensores supramencionados, que mais tarde seriam alimentados ao modelo para que fosse possível prever e identificar alguns tipos de superfícies pré-definidas.

Este *dataset* teria os dados provenientes de um acelerómetro, de um giroscópio e de um magnetómetro, o que não era exatamente a estrutura que iriamos usar, mas até á data era o mais semelhante ao nosso objetivo que tínhamos encontrado, e, portanto, com esse projeto prosseguimos para a fase de testes.

3.3. Aprender, Testar e Validar

Com o projeto bem configurado e a correr nas nossas máquinas, começamos por explorar o ficheiro Python, código para treinar e avaliar o modelo, juntamente com o *dataset* fornecido e a comparar a estrutura do mesmo com a injeção dos dados para o modelo.

3.3.1. Análise do Algoritmo para Treino de Modelos

Analisámos como é que o modelo era alimentado a partir do *dataset*, quais os parâmetros mais importantes no treino do modelo e qual era o comportamento deste ao modificar alguns desses parâmetros.

3.3.2. Análise do *Dataset* e da sua Estrutura

Analisámos o *dataset* fornecido para entender a sua estrutura ao nível do *header* (cabeçalho) do ficheiro e como eram representados os valores. Entendemos como era formatado e manipulado o *dataset*, de modo que o modelo pudesse usar o conjunto de treino e o conjunto de testes e como era feita a validação e avaliação do modelo após o treino.

3.3.3. Treino do Modelo e Resultados Obtidos

Dada como compreendida a estrutura a utilizar e o algoritmo disponibilizado, tentamos replicar o treino do modelo como havia sido documentado no repositório de onde retiramos o projeto em uso, treinamos um modelo com os parâmetros indicados nas notas do mesmo e a avaliação do modelo foi positiva, com *accuracy* de 95.7%.

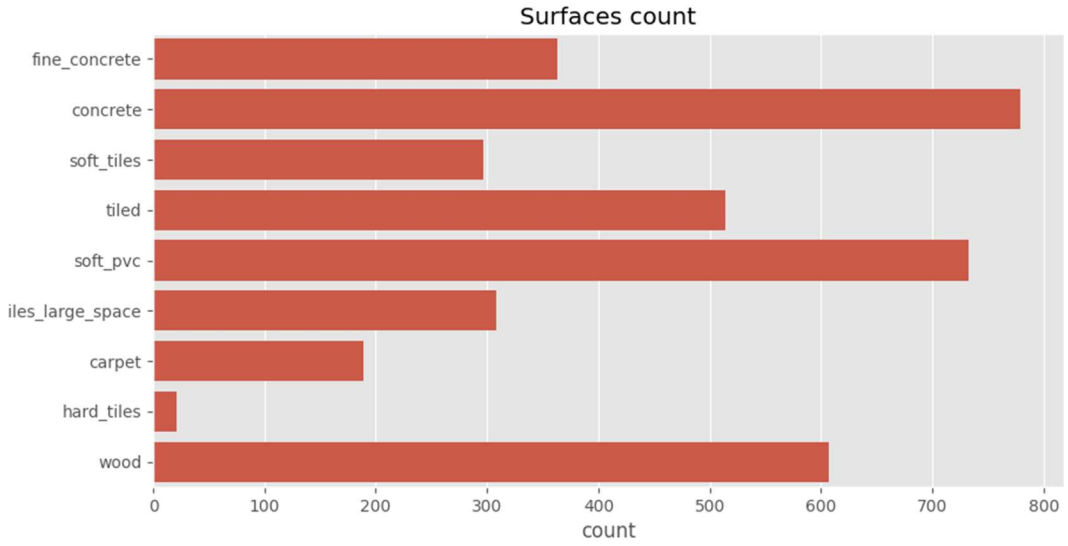

Figura 6 - Surfaces Count Bar Graph

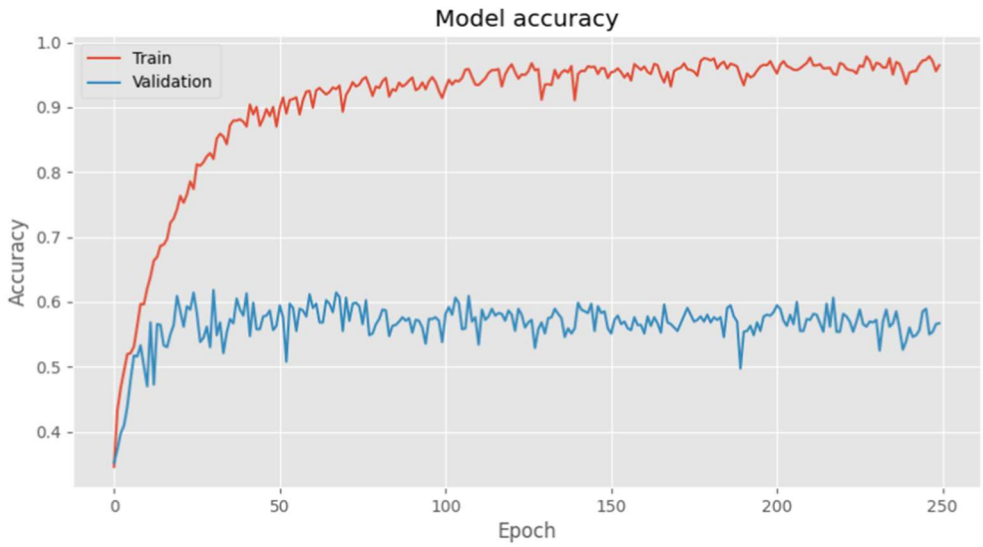

Figura 7 - Model Accuracy Plot (surfaces)

Figura 8 - Model Loss Plot (surfaces)

Sem dúvida que a fase de testes foi crucial para o sucesso deste projeto e principalmente para a passagem á próxima etapa, que seria criar um *dataset* de raiz, pelos estudantes, de modo a ter dados concretos e no formato desejado para mais tarde alimentar o modelo de inteligência artificial.

(NANASHI, 2019)

4. Construção do *Dataset*

No presente capítulo estão presentes os métodos utilizados para a construção do *dataset*, esta construção foi possível através da leitura de gestos dinâmicos que foram registados pelos estudantes utilizando os sensores disponíveis. Nesta etapa foram encontradas diversas dificuldades que ao longo do desenvolvimento foram sendo resolvidas e aperfeiçoadas.

4.1. Definição dos Gestos Utilizados

De modo a cumprir os requisitos impostos pelo caso de estudo do nosso trabalho, o controlo de um drone, definimos uma biblioteca de gestos.

Optámos por 9 gestos dinâmicos, simples e executados através do movimento da mão, do pulso e do braço direito.

Cima

Este gesto tem a função de fazer o drone ascender, sendo executado com um movimento perpendicular ao solo, de baixo para cima, seguido de um retorno à posição inicial.

Figura 9 - Gesto Cima (class 1)

Baixo

Este gesto tem a função de fazer o drone descer, sendo executado com um movimento perpendicular ao solo, de cima para baixo, seguido de um retorno à posição inicial.

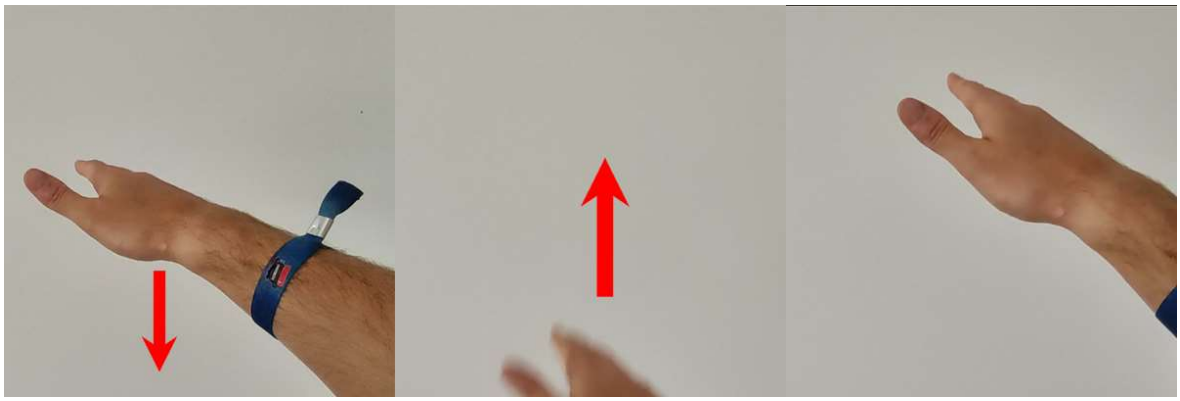

Figura 10 - Gesto Baixo (class 2)

Esquerda

Este gesto tem a função de fazer o drone se deslocar para a esquerda, é executado com um movimento paralelo ao solo, da direita para a esquerda, retornando no final à posição inicial.

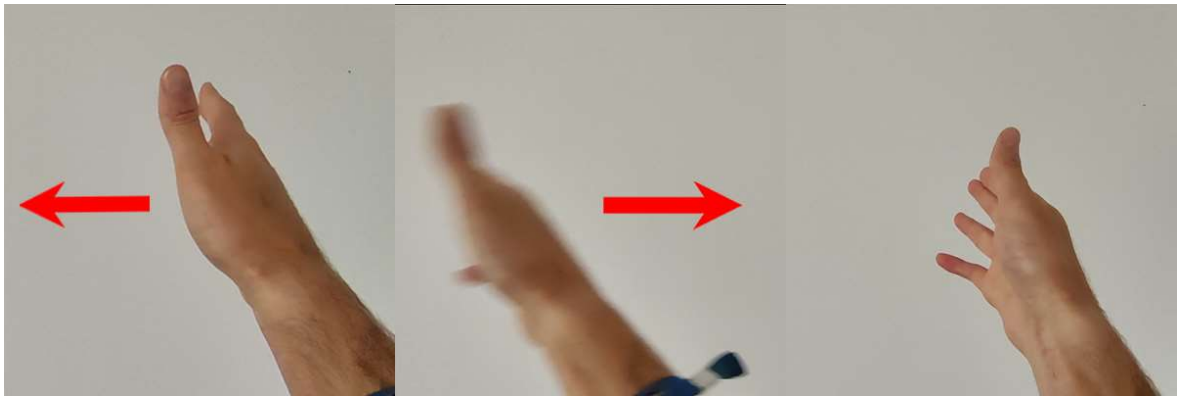

Figura 11 - Gesto Esquerda (class 3)

Direita

Este gesto tem a função de fazer o drone se deslocar para a direita, é executado com um movimento paralelo ao solo, da esquerda para a direita, retornando no final à posição inicial.

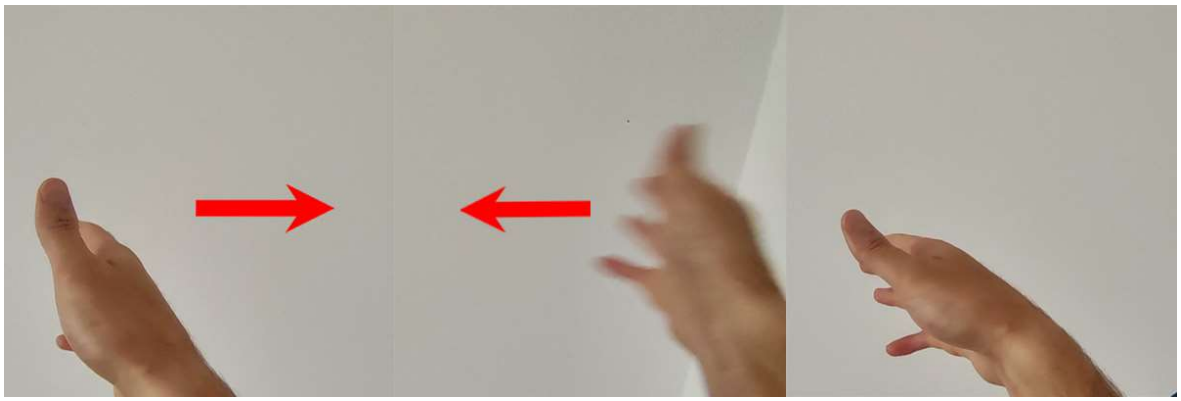

Figura 12 - Gesto Direita (class 4)

Trás

Este gesto tem a função de fazer o drone se mover para trás, é executado realizando um movimento paralelo ao solo, deslocando a mão para a frente e voltando a posição inicial.

Figura 13 - Gesto Trás (class 5)

Frente

Este gesto tem a função de fazer o drone se mover para a frente, é executado puxando a mão contra o utilizador paralelemente ao solo, retornando no final à posição inicial.

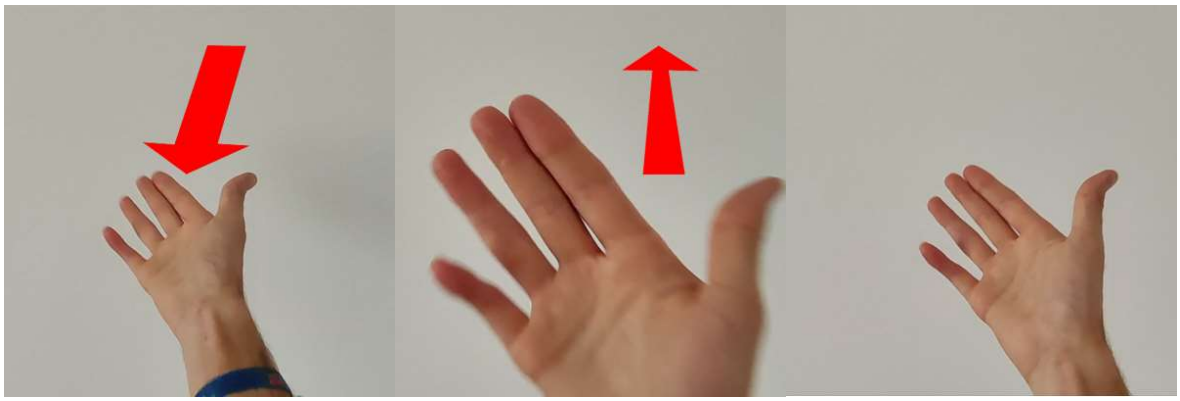

Figura 14 - Gesto Frente (class 6)

Girar

Este gesto faz com que o drone gire em torno do eixo do x, é executado girando o pulso 2 vezes perpendicularmente ao solo.

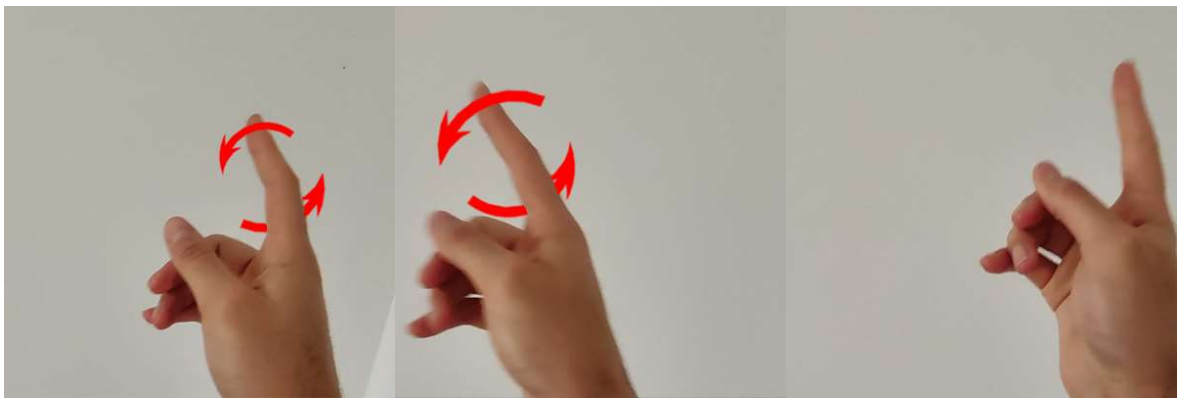

Figura 15 - Gesto Girar (class 7)

Palmas

Este gesto faz com que o drone aterre caso esteja em voo ou que decole caso esteja pousado numa superfície, é executado apenas por bater palmas comuns.

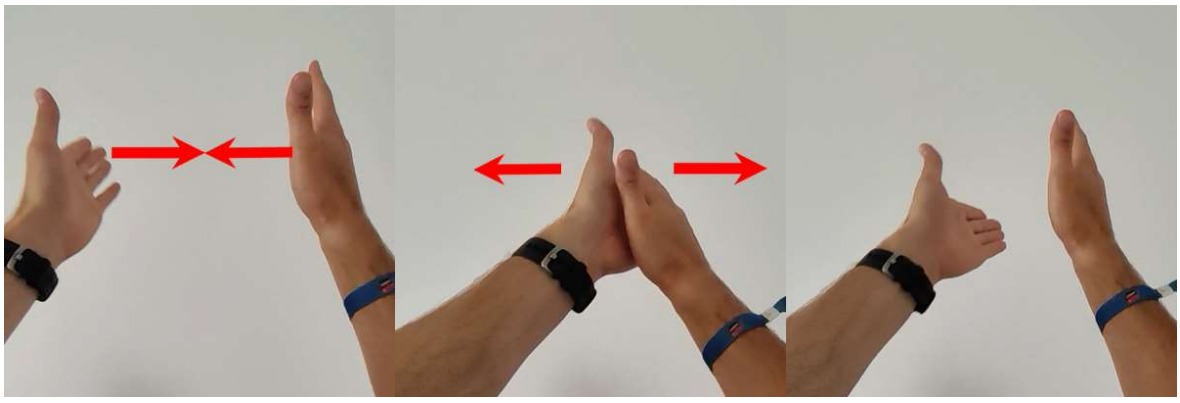

Figura 16 - Gesto Palmas (class 8)

Corte

Este é o gesto que faz com que o drone pare de executar alguma outra ação anteriormente enviada, como deslocar para a frente. Para este gesto ser executado basta simular um golpe de cortar paralelemente ao solo.

Figura 17 - Gesto Corte (class 9)

4.2.Construção do Código Utilizado para ler os Dados dos Sensores

Começamos esta fase criando um código que possibilitasse a leitura dos dados que fossem provenientes dos sensores. Para isso utilizamos o software arduino IDE possibilitando assim que conseguíssemos construir código e enviar diretamente para o arduino, tornando possível a leitura dos dados registados pelos sensores.

No início obtivemos uma versão simples do código, esta versão imprimia os valores recebidos pelos sensores, mas de forma inconsistente, se os dados variassem de maneira mais rápida o código perdia a capacidade de captar os dados, não imprimindo qualquer valor, assim sendo precisámos de melhorar o código. Para resolver este problema tivemos de utilizar um pedaço de código que permitisse a constante leitura dos dados, com recurso a uma variável conseguimos mudar o intervalo de leitura para 1000/32, sendo 1000 milissegundos sobre 32, representando os quadros por segundo

```
const unsigned long interval = 1000 / 32;
```

Figura 18 - Código arduino para definir intervalo de leitura

Com este pedaço de código melhorámos significativamente a leitura, tornando possível a leitura de movimentos mais lentos como mais rápidos. Esta melhoria expande consideravelmente o leque de utilizadores que podem beneficiar deste projeto, desde aqueles que realizam gestos mais lentos até aqueles que executam gestos mais rápidas e dinâmicos. Para além disso a versatilidade proporcionada por esta melhoria aumenta a precisão e eficiência do sistema, registando dados mais precisos, afetando diretamente o treino do modelo.

(*Software* | *Arduino*, n.d.)

4.3. Construção do *Script* para Registar os Movimentos

Nesta fase iniciamos a construção do *script* que nos permitisse efetuar o registo dos dados dos movimentos registados pelo arduino. Primeiro optámos por escolher a linguagem em que iriamos desenvolver o *script*, neste caso o Python, devido a esta ser uma linguagem versátil, bastante completa e com muita documentação disponível para consulta. Durante esta fase realizamos diversas alterações ao *script* inicial de modo a resolver os problemas que irão ser descritos mais para a frente do capítulo.

4.3.1. Inicio da Construção do *Script*

Para começarmos a criação do *script* pensámos como seria a estrutura do *dataset* que ia ser criado. Como tal, definimos na primeira linha o cabeçalho, estruturado de acordo com as suas *labels*, como apresentado de seguida.

```
header = "Group_id,Time(ms),AccelX,AccelY,AccelZ,GyroX,GyroY,GyroZ\n"
```

Figura 19 - Código Python para definir o header do *dataset*

Após definirmos o cabeçalho optámos por cada vez que o *script* é iniciado o programa iria pedir o nome do ficheiro e o *group_id* ao utilizador, sendo o nome do ficheiro o *dataset* onde vai ser gravado os dados e o *group_id* o número do movimento gravado. Apesar de isto ser uma boa maneira de registar os movimentos mantendo-os organizados e claros iria nos trazer problemas mais para a frente, sem contar que se tornaria entediante estar a colocar todas as vezes o número do *group_id,* visto que teríamos que ou decorar ou ir ao ficheiro do *dataset* de modo a ver o último *group_id* registado.

4.3.2. Melhoria do *Script*

De modo a melhorar o *script* e a resolver os erros anteriormente referidos tivemos de aprofundar nos comandos de acesso a ficheiros da linguagem Python, começamos por fazer uma verificação se o ficheiro inserido pelo utilizador existe, se existe ele acede ao mesmo, caso contrário cria um ficheiro novo e insere o cabeçalho com as *labels* para o ficheiro poder ter toda a informação necessária.

```
    with open(file_path, 'a' if file_exists else 'w') as file:
        # Se o arquivo não existir, escreve o cabeçalho
        if not file_exists:
            file.write(header)
```

Figura 20 - Código Python para leitura do ficheiro

Após termos acesso ao ficheiro, o programa ia começar a leitura dos dados recebidos pelos sensores, isto acontece porque no *script* está definido a porta em que o arduino está ligado, o programa tendo acesso a essa porta consegue ler os dados recebidos, de modo a inserirmos os dados tivemos primeiro de separar os dados recebidos.

```
# Lê os dados da porta serial e os escreve no arquivo
    while True:
        line = ser.readline().decode('utf-8').strip()  # Lê uma linha da
porta serial

        if line:  # Verifica se a linha não está vazia
            print("Linha recebida:", line)  # Verifica se os dados estão
a ser recebidos

            # Separa os dados da linha em valores individuais
            data = line.split(',')

            # Verifica se todos os dados estão presentes
            if len(data) >= 6:
                # Guarda os valores do acelerômetro e giroscópio
                accel_x = data[0].split(':')[1]
                accel_y = data[1].split(':')[1]
                accel_z = data[2].split(':')[1]
                gyro_x = data[3].split(':')[1]
                gyro_y = data[4].split(':')[1]
                gyro_z = data[5].split(':')[1]
```

Figura 21 - Código Python para leitura da porta serial

Após termos os dados da linha atual recebida separados com sucesso e guardados em variáveis guardamos esses dados no *dataset*, guardando também o *group_id* inserido e um

timestamp, permitindo-nos que tenhamos maneira de interpretar os dados sabendo qual a ordem dos movimentos que foi registada. Esta etapa é bastante importante pois se o registo dos dados não for consistente corremos o risco de ter um *dataset* com dados de baixa qualidade prejudicando o treino do modelo.

```
# Obtém o tempo atual em milissegundos
                current_time = int(round(time.time() * 1000))

                # Escreve os dados no ficheiro CSV
                file.write(f"{group_id},{current_time},{accel_x},{accel_y
},{accel_z},{gyro_x},{gyro_y},{gyro_z}\n")

                # Imprime os dados recebidos na consola
                print("Dados gravados no CSV:", group_id, current_time,
accel_x, accel_y, accel_z, gyro_x, gyro_y, gyro_z)
                file.flush()
            else:
                print("Dados incompletos na linha recebida:", line)
```

Figura 22 - Código Python para escrita no *dataset*

Até agora o *script* estava a funcionar bem, mas havia certos problemas ainda a resolver, como a inserção manual do *group_id,* para resolvermos isto utilizamos um método do Python que nos permite ler a última linha do ficheiro e com isso saber qual o último *group_id* registado, facilitando assim a utilização deste *script*.

```
if file_exists:
            with open(file_path, 'r') as f:
                last_line = f.readlines()[-1]
                last_group_id = int(last_line.split(',')[0])
                group_id = last_group_id + 1
        else:
            group_id = 1
```

Figura 23 - Código Python para verificar a existência do ficheiro

Para além do problema anterior com o *group_id* tínhamos também o problema de não saber qual o movimento que foi gravado nesse *group_id,* tendo de os apontar manualmente num ficheiro diferente, de modo a termos esse registo de forma automática optámos pela criação de um ficheiro de validação, onde se encontra o número do movimento e o número do *group_id.* De maneira a saber qual o número do movimento que iriamos gravar tivemos que no início do *script*, após a inserção do nome do ficheiro pedir qual o movimento, mas ao contrário do caso do *group_id* em que era um número desconhecido, o número do

movimento é conhecido por quem o está a fazer, portanto não iria prejudicar a maneira de utilização do *script*. O nome do ficheiro de validação não é pedido, isto porque é uma concatenação entre o nome do ficheiro onde se vai guardar os dados com a *string* "*_validation*", tornando-se assim intuitivo e fácil de reconhecer.

Inserção do número de movimento:

```
# Pede informações ao usuário
file_name = input("Digite o nome do arquivo: ")
gesture_id = int(input("Digite o ID do movimento (de 1 a 8): "))
```

Figura 24 - Código Python para inserção de informações

Definição e/ou criação do ficheiro de validação:

```
# Define o caminho do ficheiro
file_path = 'dataset/' + file_name + '.csv'
validation_file_path = 'dataset/validation_' + file_name + '.csv'

# Verifica se o ficheiro já existe
file_exists = os.path.exists(file_path)
validation_file_exists = os.path.exists(validation_file_path)
```

Figura 25 - Código Python de verificação do ficheiro de validação

Após a resolução dos problemas anteriores observámos que não tínhamos qualquer tipo de cópia destes ficheiros, sem ser as presentes no *GitHub*(*Github*, n.d.), então de modo a termos cópias e não correr o risco de perder trabalho desenvolvido optámos pela criação de backups constantes, ao executarmos o *script* e inserirmos os dados, automaticamente era criado uma cópia do ficheiro do *dataset* e do ficheiro de validação, salvaguardando assim o trabalho caso algum registo de movimento corresse sem sucesso ou ter ocorrido algum problema com o computador ou programa.

Código de criação de *backup*:

```
if file_exists and validation_file_exists:
    shutil.copyfile(file_path, "backups/" + file_path + "_" +
formatted_date + "_backup.csv")
    shutil.copyfile(validation_file_path, "backups/" +
validation_file_path + "_" + formatted_date + "_backup.csv")
    print("Backup Files Created.")
```

Figura 26 - Código Python para criação do *backup*

Neste ponto já tínhamos o *script* otimizado, seguro e com funcionalidades que tornavam a interpretação dos movimentos mais fácil, mas ainda havia tópicos a melhorar. Entre estes o tamanho dos dados dos movimentos guardados, para o modelo conseguir ler os movimentos registados no *dataset* é necessário que todos os movimentos guardados tivessem o mesmo tamanho, isto é, o mesmo número de linhas, de modo a definirmos este tamanho seguimos a opção de definir o intervalo de tempo do registo de um movimento em 2 segundos, isto é 64 quadros por segundo, tornando o tamanho dos movimentos não comprido e sem prejudicar a utilização do *script*, dando a possibilidade de quem estiver a registar os movimentos poder os registar da maneira que entender, seja esta mais rápida ou mais lenta, visto 2 segundos ser mais que suficiente para a execução de qualquer gesto a ser utilizado.

Código para manter movimentos no mesmo tamanho:

```
while movements_count < max_movements:
            # Regista o movimento por 2 segundos
            print(f"Registrando movimento {gesture_id} por 2
segundos...")
            start_time = time.time()

            # Define o número de amostras desejado para 2 segundos
            samples_per_period = 64
            samples_count = 0

            while samples_count < samples_per_period:
                line = ser.readline().decode('utf-8').strip()  # Lê uma
linha da porta serial
                print("lido: " + line)
                if line != 'MPU6050 Connected!':  # Verifica se a linha
não está vazia
                    data = line.split(',')
                    if len(data) >= 6:
                        accel_x = data[1]
                        accel_y = data[2]
                        accel_z = data[3]
                        gyro_x = data[4]
                        gyro_y = data[5]
                        gyro_z = data[6]
                        current_time = int(round(time.time() * 1000))
                        file.write(
                            f"{group_id},{current_time},{accel_x},{accel_
y},{accel_z},{gyro_x},{gyro_y},{gyro_z}\n")
                        file.flush()
                        samples_count += 1

                # Pause para manter a taxa de amostragem
                time.sleep(1 / 32)  # 32 amostras por segundo
```

Figura 27 - Código Python para manter movimentos do mesmo tamanho

Com este código teríamos amostras de movimentos mais consistentes formatadas com o mesmo tamanho, tornando possível a leitura destes dados para o algoritmo de ML. Posto estas alterações decidimos otimizar o registo dos movimentos, facilitando a utilização para quem os registasse, para isso decidimos colocar entre cada registo de movimento uma pausa de 3 segundos, dando algum tempo para o utilizador descansar.

Código para pausar registo:

```
# Pausa por 3 segundos entre os movimentos
            if movements_count < max_movements:
                print("Pausando por 3 segundos...")
                time.sleep(3)
```

Figura 28 - Código Python para pausar registo

4.4.Qualidade e Gestão dos Dados do D*ataset*

Neste ponto iremos abordar algumas tentativas de melhorar o *dataset*, usando diferentes formas e técnicas.

4.4.1. Duplicação de Dados do *Dataset*

Em busca de resolver este problema, que julgámos ser devido à pouca quantidade de amostras para treino, decidimos investir tempo a desenvolver um algoritmo em Python que, com base no *dataset* existente, multiplicava os dados em *loop* até este ter o número de movimentos pretendido. Com este algoritmo rapidamente chegávamos aos mil, 10 mil ou até mesmo 100 mil de movimentos se fosse essa a nossa vontade. O pedaço de código que faz a duplicação de código é representado na figura 68 presente nos anexos.

Contudo, mesmo com 10 mil movimentos, os resultados foram praticamente os mesmos pois, sendo os mesmos dados, em nada contribui para a generalização do modelo. Para um *dataset* de qualidade é mais importante a qualidade e diversidade dos dados do que a quantidade dos mesmos. Assim sendo, vimos que não seria benéfico o uso da duplicação de dados, pelo menos da forma como estaria a ser usada, por meio de

```
Data file: data_example
Validation data file: validation_data_example
Numero de duplicações (1/movimento): 10000
Backup Files Created.
Data duplicated successfully.

Process finished with exit code 0
```

repetir amostras. **Figura 29 - Output script generate_duplicated_data_from_csv.py**

4.4.2. Técnicas de *Data Augmentation*

Após uma pequena pesquisa encontramos uma técnica semelhante ao algoritmo que tínhamos desenvolvido, denominada de *data aumentation.* Esta variante consiste na duplicação de dados em loop, mas com uma micro variação dos valores existentes. Deste modo, aumentaríamos o *dataset* consideravelmente, sem afetar necessariamente a qualidade e diversidade dos dados.

Com o *script* que já havíamos construído seria de rápida implementação esta técnica, no entanto achamos melhor investir o tempo, desta vez, em efetivamente aumentar o *dataset* com novos dados e recolhendo-os de diversas pessoas, para assim termos uma maior diversidade e agilizar no processo de generalização do modelo.

4.4.3. Movimento *Noise* (*idle*)

Mesmo com a premissa de ter poucas amostras, o maior problema na falta de precisão e na identificação dos gestos prendia-se aos momentos em que não estávamos a executar nenhum movimento em específico. Reparamos que havia sempre uma previsão errada quando o sensor estava imóvel ou praticamente sem oscilar valores, isto é, quando não fazíamos qualquer movimento, mantendo a mão no mesmo lugar, *idle.*

Este fenómeno acontecia devido à falta de um movimento para representar o nada. O modelo ficava confuso quando, durante a previsão de um movimento, nenhum movimento era procedido. Deste modo, percebemos que teríamos de acrescentar um novo movimento ao nosso projeto e consequentemente ao *dataset*. Movimento este ao qual atribuímos o nome de *noise*, ou em português, ruído, pois representa o ruído no conjunto de dados de treino, equivalente ao não fazer movimento algum.

Deste modo, o modelo pôde identificar acertadamente quando não era executado qualquer movimento com clareza, presumindo que estes movimentos seriam apenas oscilações involuntárias, naturais do corpo humano.

4.5. Expansão do *Dataset*

Atendendo à questão da generalização, pedimos a alguns amigos e familiares que fizessem a sua própria gravação de movimentos de modo a obter uma maior diversidade do tipo e da execução dos movimentos em si.

Assim, direcionamos o nosso foco para expansão do *dataset* nos próximos tempos, até que conseguimos atingir o marco dos 3 mil gestos concentrados num único *dataset*, que já contemplava 9 dos 10 movimentos, ruído, cima, baixo, esquerda, direita, frente, trás, girar e palmas.

4.5.1. *Script* para Combinar *Datasets*

Com a necessidade de partilhar a construção do *dataset* com pessoas alheias ao projeto, surgiu também a necessidade de, posteriormente, juntar todos os dados recolhidos, ou seja, combinar todos os *datasets* que haviam sido criados pelos outros participantes aquando gravação de gestos.

Para este problema desenvolvemos um *script* que pega em tantos *datasets* quando desejado, lê as informações neles contidas e junta tudo num só. Isto foi possível através do algoritmo que passo a explicar o funcionamento:

```
def get_last_group_id(validation_file):
    # Lê o ficheiro de validação
    df = pd.read_csv(validation_file)
    # Obtém o último group_id
    if not df.empty:
        last_group_id = df.iloc[-1]['Group_id']
    else:
        last_group_id = 0
    return last_group_id
```

Figura 30 - Código Python para receber último *Group_id*

Esta função faz a recolha do *group_id* (número de identificação) do último movimento guardado no *dataset* original, para que, ao juntar os dados esta junção seja feita de forma organizada e corrida, de forma a não haver falhas e/ou buracos no *dataset*.

```
def update_group_id_and_append_data(new_data_file, data_file, validation_file,
movement_id):
    # Obtém o último group_id do ficheiro de validação
    last_group_id = get_last_group_id(validation_file)

    # Lê os novos dados
    new_data = pd.read_csv(new_data_file)

    # Atualiza os group_ids no novo conjunto de dados
    new_data['Group_id'] = new_data['Group_id'].apply(lambda x: x +
last_group_id)

    # Lê os dados existentes
    existing_data = pd.read_csv(data_file)

    # Junta os novos dados com os dados existentes
    updated_data = pd.concat([existing_data, new_data], ignore_index=True)

    # Lê os dados de validação existentes
    existing_validation_data = pd.read_csv(validation_file)

    # Adiciona uma linha ao ficheiro de validação para cada 64 linhas do novo
conjunto de dados
    for i in range(0, len(new_data), 64):
        group_id = new_data.iloc[i]['Group_id']
        validation_row = {'Group_id': group_id, 'Gesture_id': movement_id}
        existing_validation_data =
existing_validation_data._append(validation_row, ignore_index=True)

    # Escreve os dados de validação atualizados no ficheiro
    existing_validation_data.to_csv(validation_file, index=False)

    # Escreve os dados atualizados no ficheiro
    updated_data.to_csv(data_file, index=False)

    print("Dados atualizados com sucesso!")
```

Figura 31 - Código Python para atualizar *Group_id*

Esta função faz a leitura dos dados novos, corrige a sua contagem com base no *group_id* obtido na função anterior e de seguida adiciona os novos dados ao *dataset* original. De seguida, faz ainda a atualização do ficheiro de validação associado, escrevendo as *labels* e os *group_id* correspondentes, em cordialidade com os novos dados.

Este processo é repetido pelo número de vezes desejado, até que seja terminado o programa, como podemos ver na imagem que se segue:

```
Dataset Original: data
Dataset a copiar: data_copy_1
Movement ID: 1
Dados atualizados com sucesso!
Dataset a copiar: data_copy_2
Movement ID: 5
Dados atualizados com sucesso!
Dataset a copiar:
```

Figura 32 - Output ficheiro combine*datasets*.py

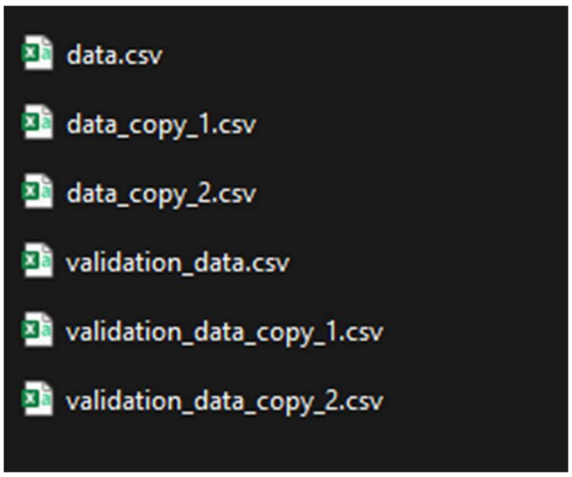

Figura 33 - Exemplo de ficheiros de output

4.5.2. Resultado da Expansão do *Dataset*

Ainda que por gravar o movimento cut e balancear o *dataset* que a esta altura teria exatamente 3317 gestos, usamo-lo para voltar a treinar alguns modelos, desta vez com uma restruturação e melhoramento do *script* de treino, para confirmar o bom desempenho do desenvolvimento do *dataset* a respeito das previsões do modelo.

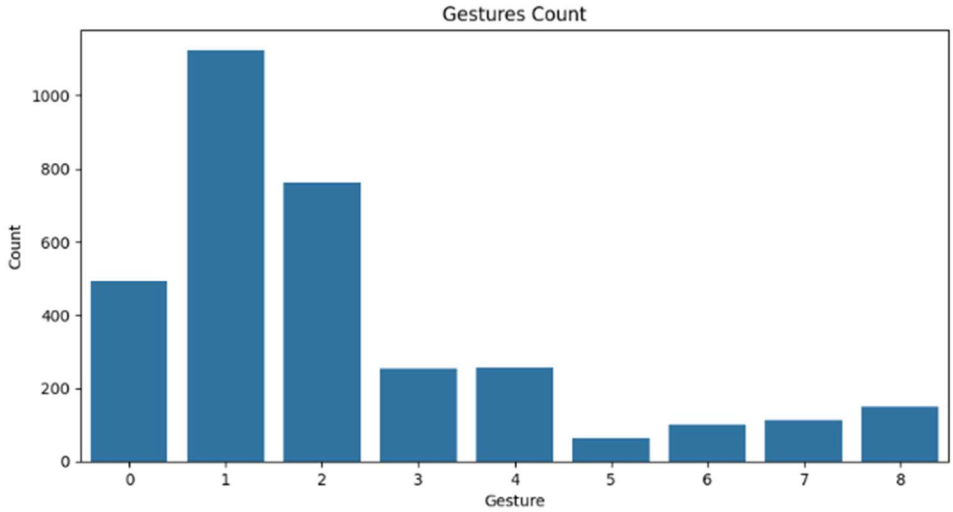

Figura 34 - Gesture Count (contagem de gestos por class) dataset 3300 gestos

4.6.Erros Conhecidos

Neste subcapítulo vamos mostrar quais os erros conhecidos durante o registo dos movimentos. Estes erros foram acontecendo em momentos diferentes e alguns deles foram persistentes e continuaram a acontecer, lembramos que estes erros em nada têm a ver com o funcionamento do projeto, apenas na parte do registo dos movimentos, todos os erros que forem mencionados estão previstos e existe código para os tratar sem qualquer prejuízo ao *dataset* final.

4.6.1. Erro: *IndexError*

O primeiro erro registado é durante a gravação de um movimento, seja este qualquer movimento, aparece como demonstrado na imagem seguinte, referindo que o *index* está fora de alcance. Este erro acontece devido a um atraso no sensor, fazendo com que a leitura aconteça antes do tempo. A resolução deste erro é simples, sendo apenas necessário reiniciar o *script* que voltará a registar os movimentos normalmente.

```
lido: 6.30,-3.52,5.84,2.59,-0.70,1.56
Traceback (most recent call last):
  File "C:\dev\DroneGestProject\V5 - ModeloDatasetCustomCimaBaixo\gravarMovimentos.py", line 97, in <module>
    gyro_z = data[6]
             ~~~~^^^
IndexError: list index out of range
```

Figura 35 - Erro *index* out of range

4.6.2. Erro de Resposta do Arduino

Em algumas execuções no *script* ocorreu casos em que o sensor parava de responder sem uma razão aparente, deixando de retornar os valores nas leituras. Quando este erro acontece devemos parar o mais rápido a execução do *script*. Lembramos que o *script* está preparado para tratar este erro, portanto apesar de ocorrer o mesmo não há qualquer tipo de prejuízo ao trabalho realizado.

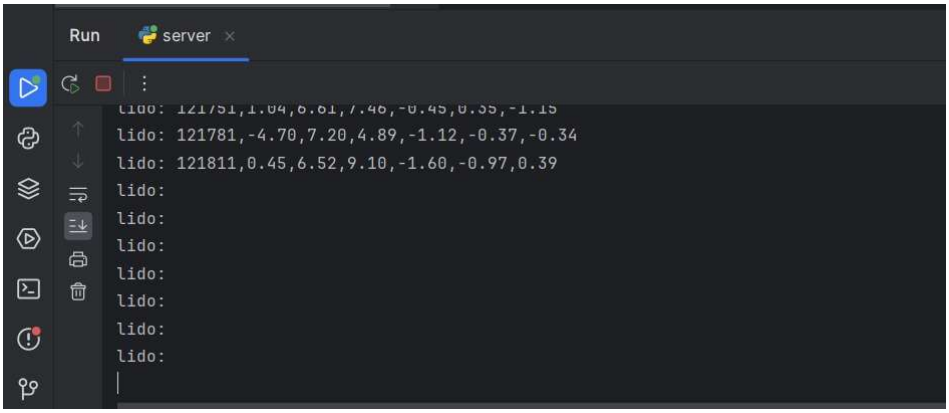

Figura 36 - Erro de o sensor parar de ler os dados

5. Treino de Modelos com o D*ataset* Próprio

Neste capítulo iremos abordar a criação dos primeiros modelos treinados com o *dataset* elaborado anteriormente, ainda que em fase de construção, mas já com um número aceitável de amostras recolhidas. O ficheiro para treino do modelo estará disponível para consulta em anexo a este relatório, ficheiro este que no decorrer do projeto foi sofrendo melhorias e modificações.

5.1. Primeira Ronda de Treino com *Dataset* Próprio

Após algum tempo dedicado à elaboração de um *dataset* de qualidade, equilibrado e á partida com capacidade para treinar o nosso modelo, decidimos partir para o treino de um modelo de teste, para medir e analisar como estaria o modelo a prever com o *dataset* que estávamos a fazer.

5.1.1. Treino do Modelo

A esta altura teríamos um *dataset* apenas com dois movimentos, *up* (0) e *down* (1), que somavam exatamente 353 movimentos, um total de 22592 amostras, mas que, ao contrário do primeiro *dataset*, teria o número de amostragem certo e fixado a 64 amostras por movimento.

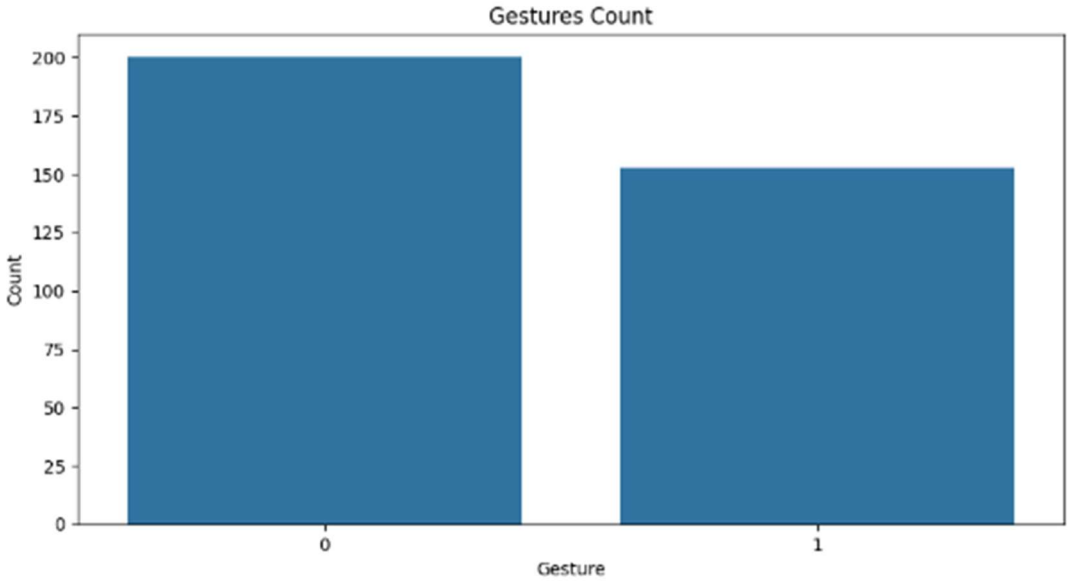

Figura 37 - Gesture Count Plot (contagem de gestos por classe)

	A	B	C	D	E	F
1	Group_id,Time(ms),AccelX,AccelY,AccelZ,GyroX,GyroY,GyroZ					
2	1,1714732987989,-3.54,-5.06,8.19,0.10,0.00,0.04					
3	1,1714732988022,-3.18,-5.00,8.01,0.01,-0.12,-0.02					
4	1,1714732988055,-3.08,-5.32,8.00,-0.04,-0.12,-0.01					
5	1,1714732988088,-3.15,-5.52,7.92,-0.12,-0.06,-0.03					
6	1,1714732988122,-3.30,-5.78,7.88,-0.21,-0.06,-0.05					
7	1,1714732988155,-3.18,-5.89,7.71,-0.21,-0.01,-0.08					
8	1,1714732988188,-3.11,-6.03,8.10,-0.21,0.13,-0.04					
9	1,1714732988222,-3.48,-6.04,8.59,-0.14,0.12,0.01					
10	1,1714732988256,-3.39,-6.15,8.11,-0.02,0.09,0.03					
11	1,1714732988289,-3.20,-6.07,7.74,0.06,0.13,0.03					
12	1,1714732988322,-3.33,-6.06,7.94,0.06,0.17,0.04					

	A	B	C
1	Group_id,Gesture_id		
2	1,1		
3	2,1		
4	3,1		
5	4,1		
6	5,1		
7	6,1		
8	7,1		
9	8,1		
10	9,1		
11	10,1		
12	11,1		

Figura 38 - Ficheiro de dados CSV (treino) e ficheiro de dados CSV (validação)

22583	353,1714821080005,0.66,2.66,5.00,3.04,-1.26,1.22
22584	353,1714821080037,1.44,2.70,5.17,2.47,-0.60,1.10
22585	353,1714821080070,1.23,3.80,6.24,2.15,-0.43,0.92
22586	353,1714821080103,1.59,4.89,6.05,1.53,-0.32,0.60
22587	353,1714821080135,2.29,5.16,5.81,1.09,-0.02,0.42
22588	353,1714821080169,1.98,5.15,6.68,0.83,0.01,0.27
22589	353,1714821080202,1.86,5.82,7.14,0.65,0.00,0.04
22590	353,1714821080235,1.43,6.39,7.12,0.47,0.20,-0.09
22591	353,1714821080269,0.77,6.64,7.69,0.20,0.16,-0.08
22592	353,1714821080301,0.25,6.48,8.49,0.23,-0.03,-0.08
22593	353,1714821080334,1.13,6.58,8.52,0.18,0.11,-0.13
22594	
22595	

345	344,2
346	345,2
347	346,2
348	347,2
349	348,2
350	349,2
351	350,2
352	351,2
353	352,2
354	353,2
355	
356	
357	

Figura 39 - Ficheiro de dados CSV (treino) e ficheiro de dados CSV (validação) parte 2

Desta forma foi possível de facto treinar o modelo, por enquanto sem modificar a parametrização original, que apresentou os seguintes resultados:

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 0.9296 - val_loss: 98.7640
Epoch 249/250
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 0.9296 - val_loss: 98.6761
Epoch 250/250
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 0.9296 - val_loss: 98.6021
Model saved
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
Avg score:  0.9295774647887324
```

Figura 40 - Output do treino do modelo

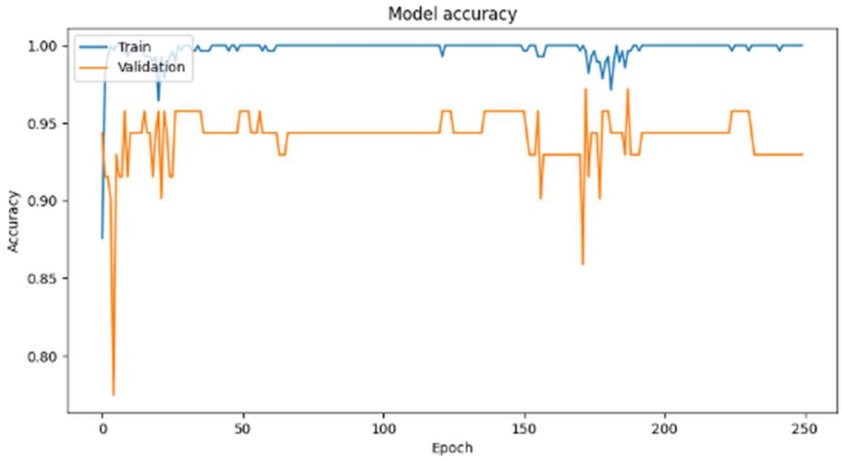

Figura 41 - Model accuracy plot do modelo treinado

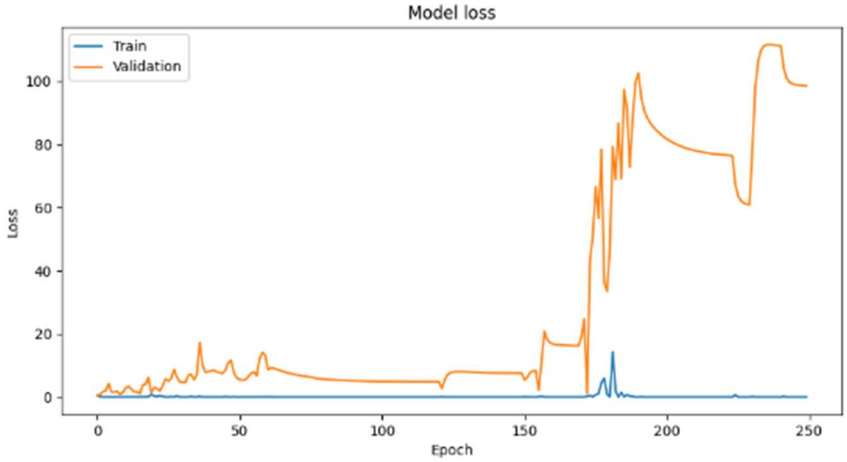

Figura 42 - Model Loss plot do modelo treinado

Treinado o modelo percebemos que, apesar de obtida uma *accuracy* elevada, com cerca de 93%, o modelo apenas conseguia prever com exatidão movimentos bem executados, sem diferir muito daqueles feitos para treino, ou seja, se fosse o autor do *dataset* a executar o movimento o modelo, normalmente, iria prever adequadamente. No entanto se fosse outra pessoa o modelo tendia a errar ou até mesmo não identificar o movimento correto. Percebemos então que teríamos problemas de generalização.

5.1.2. Treino de Modelos com Ajuste de Parâmetros

Mesmo sabendo que o problema principal estaria, à partida, no *dataset* com a pequena quantidade de dados e falta de diversificação, tentámos ajustar os principais parâmetros de modo a tentar atingir uma *accuracy* mais alta. Os resultados obtidos nos treinos são apresentados na Tabela 1.

Tabela 1 - Resultados dos modelos com o *dataset* de 350 gestos

Epochs	*Batch size*	*Learning rate*	*Accuracy*
1000	64	0.02	**0.5633**
1000	64	0.1	**0.5633**
1000	64	0.1	**0.5633**
50	32	0.003	**0.9295**
250	32	0.003	**0.9295**
50	64	0.003	**0.9436**
1000	32	0.003	**0.9436**
1000	64	0.005	**0.9436**
2500	64	0.003	**0.9436**
10000	32	0.003	**0.9436**
10000	64	0.003	**0.9436**
250	64	0.003	**0.9577**
500	64	0.003	**0.9577**
1000	64	0.0085	**0.9577**
1000	64	0.003	**0.9718**
1000	64	0.01	**0.9859**

No final, conseguimos terminar com um modelo de 98.5% de precisão, mas que, como descrito acima, não generalizava muito bem. Assim, avançamos para a expansão do *dataset* e algumas outras técnicas.

5.2.Parametrização e Funções em Algoritmos de ML

Quando satisfeitos com a taxa de acerto do modelo, na altura apenas focado no movimento *up*, *down* e agora o novo que se chamara *noise*, percebemos que ainda havia espaço para melhorias.

5.2.1. Ajuste dos Principais Parâmetros

Começamos por ajustar os três principais parâmetros, *epochs number, Batch size e Learning rate.*

O *epochs number* ou, número de épocas é uma iteração completa por todo o conjunto de dados durante o treino de um modelo. Serve para atualizar os pesos da rede neural após passar por todas as amostras. Valores padrão variam entre 10 e 1000 consoante o conjunto de dados em questão.

O *Batch size*, ou tamanho da *batch* é o número de amostras processadas antes de atualizar os pesos do modelo. Afeta a estabilidade e velocidade do treino. Valores padrão variam entre: 16 e 128.

O *Learning rate* ou, taxa de aprendizagem é uma medida de ajuste dos pesos durante a otimização. Um valor adequado acelera a convergência do modelo, enquanto um valor inadequado pode levar a resultados subótimos, que não são os melhores. Valores padrão para esta métrica estão entre 0.01 e 0.00001

5.2.2. Definição dos Hiper Parâmetros

De seguida, definimos alguns Hiper parâmetros tais como o *Dropout*, *Dense* e max*Pooling*, que ajudariam na precisão do modelo e prevenção de fenómenos como *overfitting*.

Dropout é uma técnica de regularização usada em redes neurais para tentar prevenir principalmente o *overfitting*. Durante o treino, desativa aleatoriamente uma fração das unidades da camada, forçando a rede a ser independente de unidades específicas. Valores padrão variam entre 0.2 e 0.5. Por exemplo, *Dropout* (0.5) indica que 50% dos neurônios serão desativados aleatoriamente em cada *epoch*.

Dense, ou camada totalmente conectada, onde cada neurônio desta camada está conectado a todos os neurônios da camada anterior numa rede neural. É usada para

aprender combinações não-lineares complexas dos dados de entrada. Por exemplo, *Dense* (128, activation='relu') especifica uma camada com 128 neurônios e função de ativação ReLU.

Max*Pooling* é uma função de *Pooling* (polimento) usada em redes neurais convulsionais para reduzir a dimensionalidade dos dados, retendo as características mais importantes. No caso, Max*Pooling*1D, aplica-se ao longo de uma dimensão e seleciona o valor máximo em cada janela de *Pooling*. Por exemplo, Max*Pooling*1D (3) indica o tamanho máximo da janela de *Pooling* de 3 unidades consecutivas.

5.2.3. Funções de ML

Por fim, passamos á implementação de algumas funções interessantes para facilitar ao modelo, atingir o máximo de precisão possível.

Reduce Learning rate, ou *Reduce* LR, amplamente usada em algoritmos de ML, reduz a taxa de aprendizagem (*Learning rate*) quando a métrica monitorada, no caso a precisão, parar de melhorar.

Early Stopping, como o nome indica, é a função que termina automaticamente o treino do modelo se a métrica monitorada, no caso a precisão, parar de melhorar.

Deste modo, o treino do modelo é feito com maior controlo e dinamismo quando pensamos em retrações no desempenho o que torna possível obter sempre o melhor modelo de cada treino.

5.3. Algoritmo Final e Treino em Bulk

Com as definições acima, chegamos ao resultado do algoritmo apresentado de seguida:

```
def create_model(EPOCH, BATCH, LEARNING_RATE):
    model = Sequential()
    model.add(Conv1D(64, 3, activation='relu', input_shape=(shape[1],
shape[2])))
    model.add(BatchNormalization())
    model.add(Dropout(0.3))
    model.add(Conv1D(128, 3, activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling1D(3))
    model.add(Dropout(0.3))
    model.add(Conv1D(256, 3, activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling1D(3))
    model.add(Dropout(0.3))
    model.add(Conv1D(512, 3, activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling1D(3))
    model.add(Flatten())
    model.add(Dense(256, activation='relu', kernel_regularizer=l2(0.01)))
    model.add(Dropout(0.5))
    model.add(Dense(unique_gestures, activation='softmax'))

    optimizer = Adam(learning_rate=LEARNING_RATE)
    model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5,
min_lr=1e-6, verbose=1)
    early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True, verbose=1)

    history = model.fit(x_train, y_train, batch_size=BATCH, epochs=EPOCH,
validation_data=(x_val, y_val),
                        callbacks=[reduce_lr, early_stopping], verbose=1)
    return model, history
```

Figura 43 - Código Python para criar um modelo

Este será o algoritmo usado daqui para a frente na fase seguinte, que será o treino de modelos, de modo a encontrar a parametrização ótima com métrica de avaliação a precisão do modelo aquando previsão e identificação de gestos.

Ao treinar os primeiros modelos percebemos que iria ser algo de bastante trabalhoso e com alto consumo de tempo o treino de modelos com parametrizações diferentes, isto porque consoante a parametrização inserida, o treino levaria desde alguns minutos até horas pelas qual teríamos de esperar para poder pôr o *script* a correr novamente e treinar outro

modelo. Dado este problema, desenvolvemos no ficheiro de treino de modelos um automatismo que perguntaria a quantidade de modelos a treinar, com que parametrização e no final de cada treino o programa gera um ficheiro PDF com as estatísticas do modelo, quer ao longo do treino, quer os resultados do mesmo.

```
prefix = "Modelos/" + input("Enter the model prefix name: ")
MODEL = prefix + ".keras"
if not os.path.exists(MODEL):
    n_models = int(input("Number of models to train: "))
    if n_models == 1:
        EPOCH = int(input("Epochs: "))
        BATCH = int(input("Batch size: "))
        LEARNING_RATE = float(input("Learning rate: "))
    else:
        EPOCH = [0] * n_models
        BATCH = [0] * n_models
        LEARNING_RATE = [0] * n_models
        for i in range(n_models):
            print("Model ", i + 1)
            EPOCH[i] = int(input("Epochs: "))
            BATCH[i] = int(input("Batch size: "))
            LEARNING_RATE[i] = float(input("Learning rate: ")
```

Figura 44 - Código Python para criação de ficheiro com dados do modelo treinado

Desta forma era possível o treino de múltiplos modelos sem supervisão constante por parte dos estudantes, o que proporcionou conforto e uma otimização significante no tempo dedicado ao treino de modelos. Na prática, foi possível treinar por exemplo 30 modelos, cujo tempo de treino total se refletira em algumas horas, sem a necessidade do acompanhamento humano, sendo possível mais tarde a analise de resultados através dos documentos PDF gerados.

Ainda assim, surgiu a necessidade de identificar os modelos treinados, numa ótica de saber diretamente e com mais simplicidade quais os parâmetros usados e qual a *accuracy* atingida do modelo em questão. Dada a situação implementamos, ainda no ficheiro de treino de modelos, uma vertente de avaliação, que através do *dataset* usado para o treino, calculava e apresentava todas as estatísticas novamente. Adicionalmente, achamos que seria interessante guardar cada modelo com a seguinte nomenclatura: prefixo + *epochs_number* + *batch_size* + *learning_rate* + *accuracy* + ". keras". Deste modo, facilmente percebíamos e identificávamos qual era o modelo em questão.

Finalmente, este foi o resultado visual do *script*, que agora se chamara "TrainAndEvaluateModel.py" e uma parte das respetivas estatísticas em formato PDF.

```
Enter the dataset filename: data
Shape of data: (3317, 64, 6)
Shape of labels: (3317,)
Number of unique gestures:  9
Treinar novo modelo ou avaliar um modelo existente? (t/a)
t
Enter the model prefix name: model_allMoves
Number of models to train: 10
Model  1
Epochs: 50
Batch size: 16
Learning rate: 0.003
Model  2
Epochs: 100
Batch size: 16
Learning rate: 0.003
Model  3
Epochs: 50
Batch size: 32
Learning rate: 0.003
Model  4
Epochs: 100
Batch size: 32
Learning rate: 0.003
Model  5
```

Figura 45 - Output da versão final do ficheiro TrainAndEvaluateModel.py

E após o treino dos modelos inseridos anteriormente no *script*, é gerado o documento PDF com as estatísticas do mesmo, as quais apresentamos algumas nas imagens abaixo.

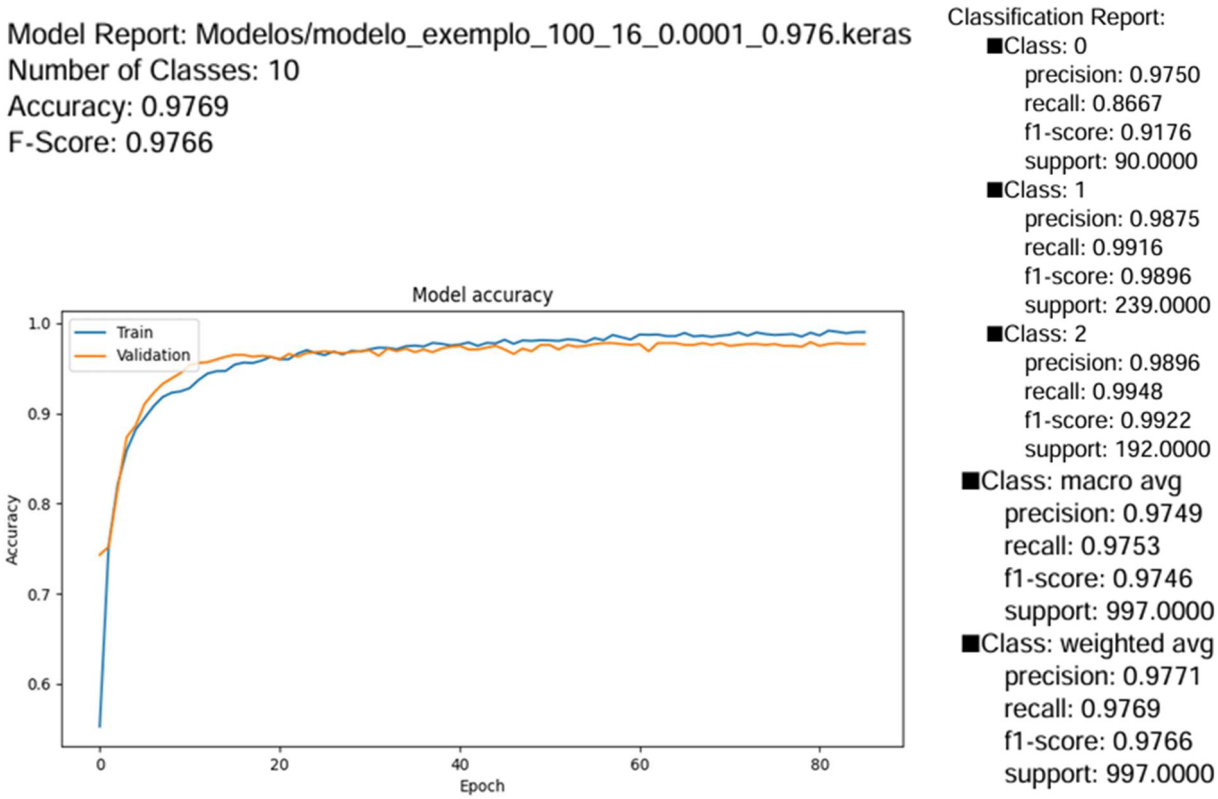

Figura 46 - Imagens ilustrativas do ficheiro PDF gerado

5.4. Treino de Modelos com o Novo Algoritmo

Desta forma, usando o algoritmo de treino melhorado e o *dataset* mencionado acima com cerca de 3300 registos, distribuídos por 9 movimentos, obtivemos os seguintes resultados representados na tabela 2.

Tabela 2 - Resultados dos modelos treinados com o *dataset* de 3300 gestos

Epochs	***Batch size***	***Learning rate***	***Accuracy***
650	64	0.03	**0.331**
1000	64	0.03	**0.331**
650	64	0.01	**0.332**
1000	64	0.01	**0.335**
1000	64	0.01	**0.592**
1000	64	0.01	**0.681**
50	128	0.0001	**0.963**
650	64	0.01	**0.964**
100	16	0.00005	**0.968**
100	64	0.00005	**0.969**

50	16	0.003	**0.984**
50	32	0.003	**0.984**
100	32	0.003	**0.987**
100	64	0.01	**0.989**
100	128	0.01	**0.989**
100	16	0.003	**0.989**

Percebemos através dos gráficos de *model accuracy* e *model loss*, que mostram a evolução do modelo ao longo do treino por *epochs*, que a partir de uma certa quantidade de épocas o modelo tendia a baixar significativamente a precisão. Fenómeno conhecido amplamente na área por *overfitting*.

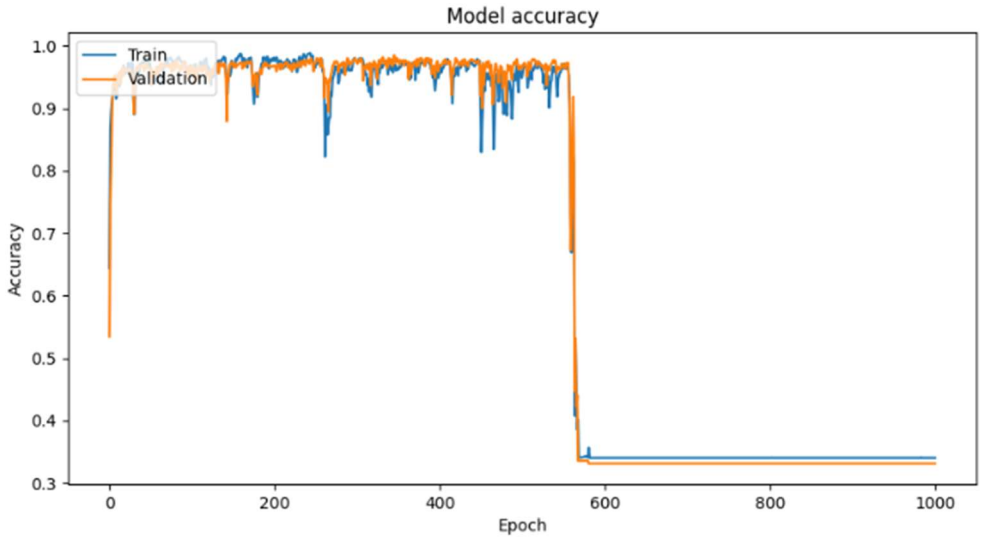

Figura 47 - Grafico model accuracy com demonstração de overfitting

Dado isto, decidimos baixar o número de *epochs*, já que o modelo atingiria máximos relativos de precisão já a partir das 50 épocas. Assim, não haveria necessidade de prolongar muito mais o treino pelo que começamos a trabalhar com a faixa de 50 a 100 épocas por treino, prevenindo o *overfitting*, acelerando o tempo de treino por modelo e mantendo os resultados elevados.

Além do problema de *overfitting* percebemos também que, devido ao aumento de dados, poderia ser benéfico ajustar o valor inicial do parâmetro *Learning rate*, visto que o modelo teria uma quantidade mais elevada de dados a considerar na sua aprendizagem e acabava por ter uma alteração de *accuracy* demasiado precipitada.

6. Finalização da implementação, demonstração e Testes

Neste capítulo iremos abordar a fase da finalização da implementação do projeto e a forma escolhida para a demonstração dos gestos num ambiente simulado.

6.1. Conclusão do *Dataset*

Nesta fase começamos por concluir o *dataset* com os todos os movimentos, procuramos ter o mesmo número de movimentos em todos, permitindo assim uma maior precisão para o treino do modelo. O *dataset* já finalizado, inclusive com o gesto de cortar, conta com 5500 movimentos no total, sendo estes distribuídos de maneira uniforme.

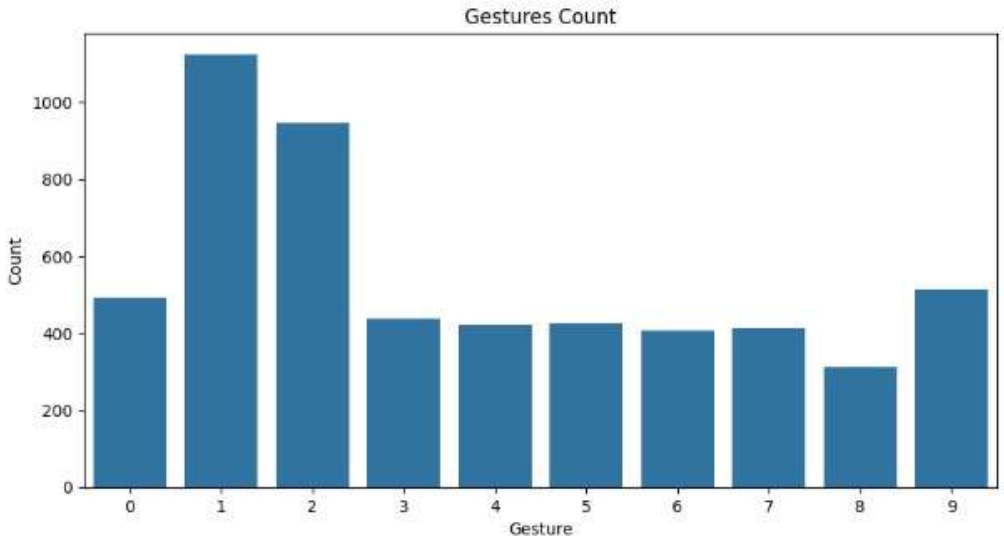

Figura 48 - Gesture Count (contagem de gestos por classe) do *dataset* final

Como podemos observar no gráfico de barras anterior notamos que todos os gestos gravados têm cerca de 500 movimentos, também podemos ver que gesto número 1 e número 2, gesto de subir e descer respetivamente, têm aproximadamente 1000 movimentos cada, isto devido a terem sido os primeiros gestos a serem gravados, mas não apenas isso como também por terem sido os gestos utilizados para testar a versão final do *script* de registo de movimentos, consequentemente aumentando o número de movimentos destes mesmos gestos.

Com este *dataset* já finalizado podemos começar a criar mais modelos, onde para cada um deles alteramos os parâmetros de forma a obter resultados diferentes, possibilitando assim encontrar um modelo com a maior percentagem de precisão, modelo este que teria a melhor capacidade de prever os movimentos com a menor taxa de erro. No próximo subcapítulo falaremos da comparação entre os parâmetros e os resultados dos modelos treinados.

6.2. Treino e comparação de modelos

Como podemos observar pela tabela 3, a seguir representada, foram executados vários testes para tentar encontrar o melhor modelo, que nos permitisse ter um modelo eficaz, mas também sem necessitar de muito poder computacional.

Tabela 3 - Resultados dos modelos treinados com o *dataset* final de 5500 gestos

Epochs	***Batch size***	***Learning rate***	***Accuracy***
100	64	0.0001	**0.953**
50	128	0.0001	**0.956**
100	128	0.0001	**0.959**
50	128	0.00005	**0.961**
50	64	0.00005	**0.962**
100	128	0.00005	**0.964**
100	64	0.00005	**0.968**
100	32	0.00005	**0.971**
50	32	0.00005	**0.971**
50	64	0.0001	**0.971**
50	16	0.00005	**0.973**
10000	16	0.00005	**0.975**
100	16	0.00005	**0.977**
50	32	0.0001	**0.977**
50	16	0.0001	**0.977**
100	64	0.001	**0.977**
100	32	0.0001	**0.978**
100	16	0.005	**0.980**

100	16	0.01	**0.980**
80	16	0.0001	**0.980**
120	16	0.005	**0.980**
120	16	0.00005	**0.980**
50	32	0.001	**0.980**
120	16	0.01	**0.981**
100	16	0.0001	**0.981**
100	128	0.001	**0.981**
50	128	0.001	**0.981**
50	64	0.001	**0.981**
100	32	0.001	**0.982**
100	16	0.001	**0.982**
1000	16	0.0001	**0.983**
100	64	0.01	**0.983**
50	16	0.001	**0.983**
500	16	0.001	**0.983**

Ao analisarmos a tabela mais detalhadamente podemos concluir que nem sempre um *Batch size* elevada ajuda na precisão do modelo, pelo contrário, obtivemos valores mais altos após reduzir o mesmo. Juntamente com o *Batch size* existe outro valor importante para o treino do modelo e o aumento da precisão, este é o número de *Epochs*, após a realização dos treinos e a comparação dos resultados obtidos podemos observar que diferente do que acontece no *Batch size* este é um valor que varia mais, os melhores resultados são de valores intermédios, nem muito altos nem muito baixos. Outro fator que influencia o resultado do modelo é o *Learning rate*, que depois dos testes obtivemos vários valores ideais, variando entre os 0.0001 e o 0.1.

Focando agora no pior modelo, observamos que obteve uma precisão de 95.3% em conjunto com um *F1-Score* de 0.9508. Não estando presente na tabela também podemos observar as classes do modelo, onde podemos observar quais as classes que obtiveram uma melhor precisão. Neste pior modelo a classe que obteve um melhor resultado teve uma precisão de 100%, sendo várias classes, mas observando a que teve o pior resultado vemos que tem uma diferença significativa, tendo uma precisão de 81.9%, sendo a classe 3, baixando a precisão geral do modelo totalizando os 95.3%.

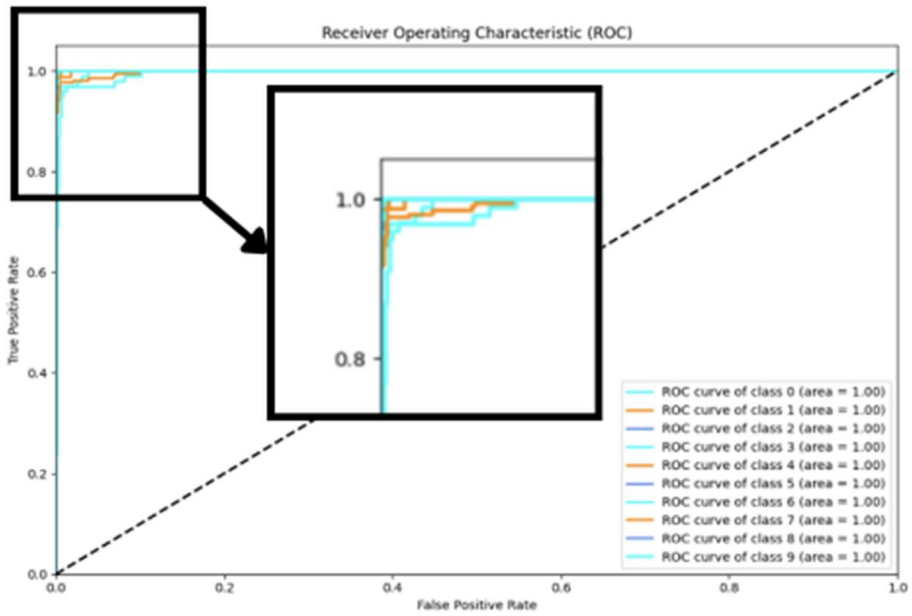

Figura 49 - Gráfico das curvas ROC do pior modelo treinado

No gráfico acima podemos observar as curvas ROC do modelo, observando que a pior curva é a da classe 3, que é a pior classe neste modelo.

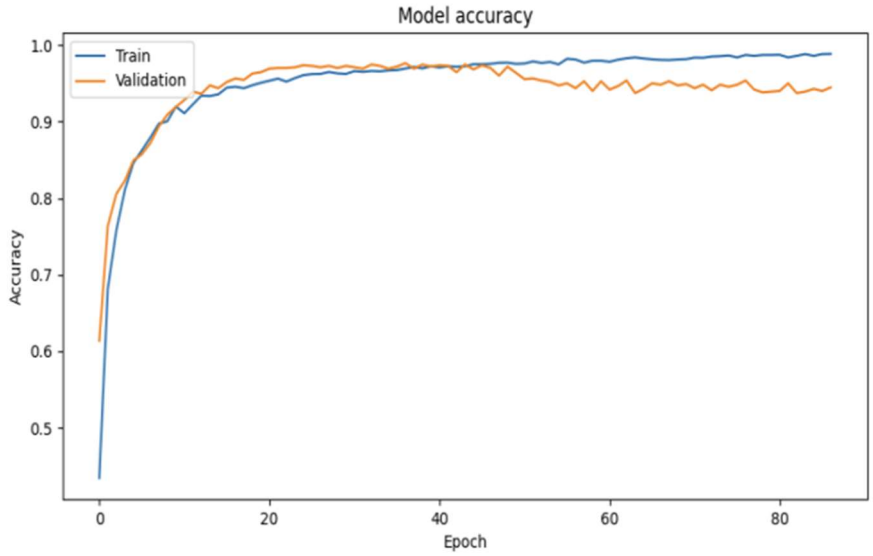

Figura 50 - Gráfico da precisão do pior modelo treinado

No gráfico acima observamos o gráfico que demonstra a curva da precisão, realçando que a partir das 40 *Epochs* a precisão com os dados de validação tendeu a baixar cerca de 5%.

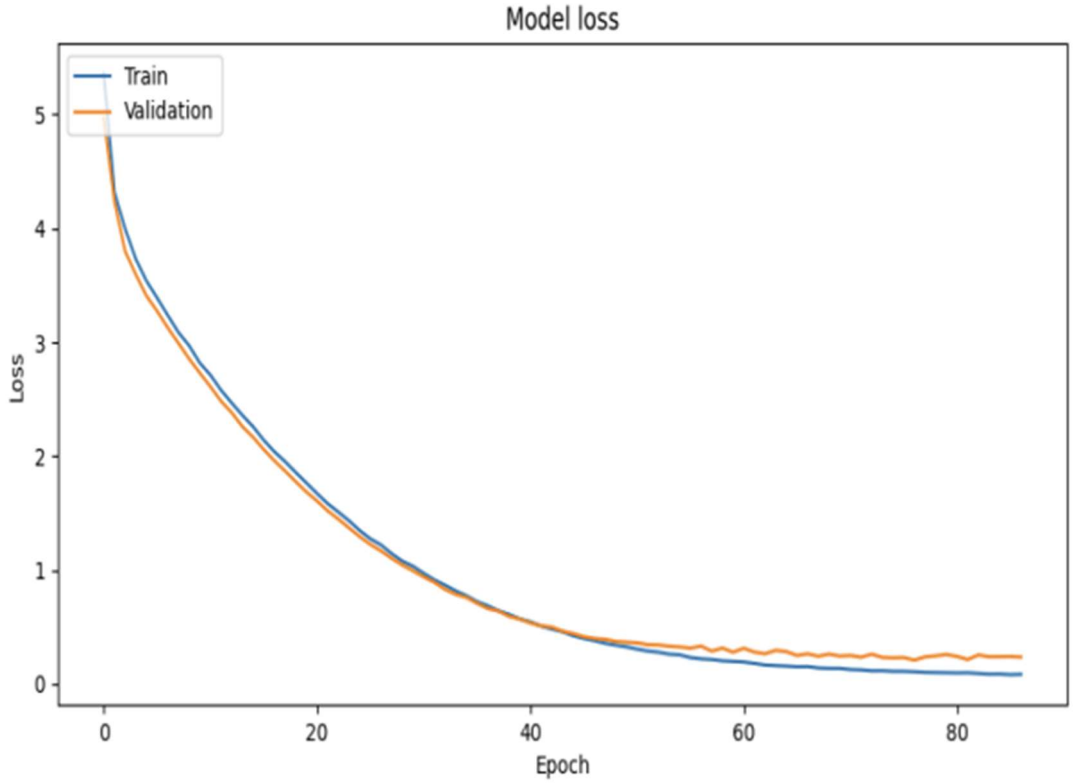

Figura 51 - Gráfico da curva de perda do pior modelo treinado

O gráfico acima representado mostra a informação da curva de perda do modelo.

Observando agora o melhor modelo treinado, com uma precisão de 98.3% e um F-score de 98.3%. Este modelo com a melhor precisão conseguida teve um número de classes maior com a precisão e 100%. Observando agora a pior classe com a pior precisão, sendo esta a classe 0, notamos que tem uma precisão de 91.2%, o que comparando ao pior modelo é uma diferença de aproximadamente 10%, o que é bastante no contexto do projeto, podendo ser esta a diferença para uma má performance no controlo do drone.

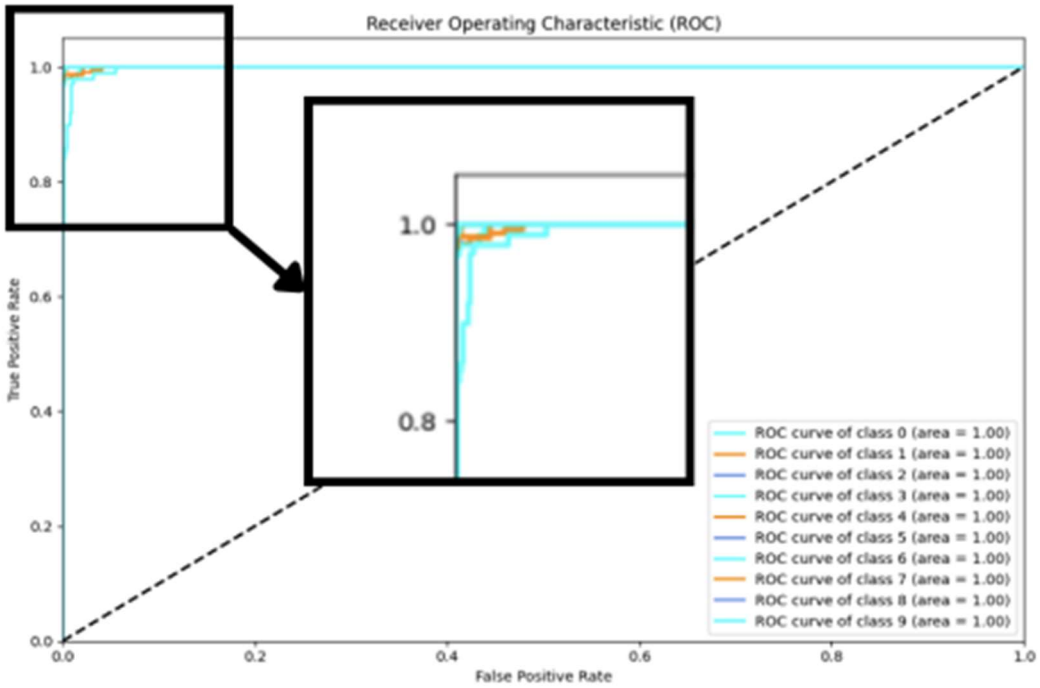

Figura 52 - Gráfico das curvas ROC do melhor modelo treinado

Observando o gráfico acima, neste caso o gráfico das curvas ROC do melhor modelo obtido, notamos que comparado ao pior modelo, conta com uma subida muito mais acentuada, resultando numa precisão maior.

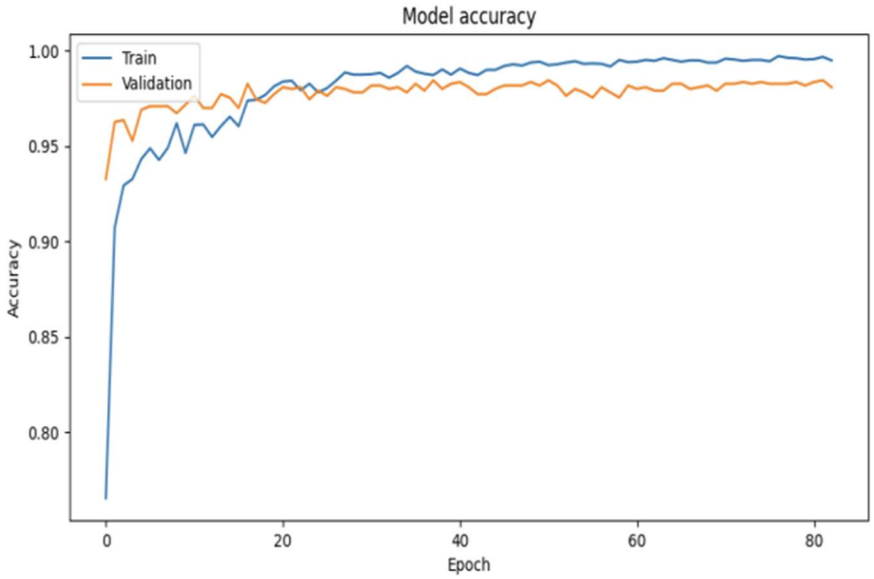

Figura 53 - Gráfico da precisão do melhor modelo treinado

O gráfico acima representa a curva da precisão do modelo, podemos observar que ao contrário do gráfico do pior modelo, que a partir das 40 *Epochs* tendia a baixar a precisão, neste caso não o acontece, mantendo sempre uma precisão aproximada dos 95% usando os dados de validação. Resultando isto num modelo ideal com alto nível de precisão, sendo este capaz de reconhecer a maioria dos gestos executados com precisão e baixa taxa de erro.

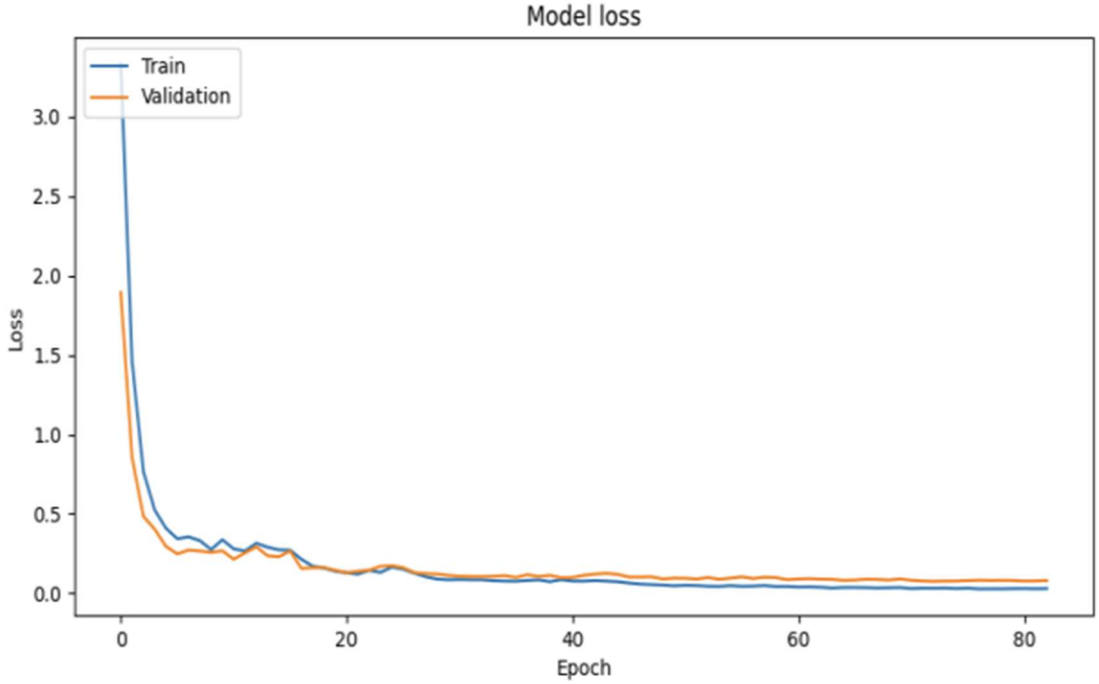

Figura 54 - Gráfico da curva de perda do melhor modelo treinado

O gráfico acima presenta a curva de perda do melhor modelo treinado.

No próximo subcapítulo iremos abordar os hyperparameters e as suas caracterizações.

6.3.Caracterização dos Hyperparameters

Aqui podemos observar a caracterização dos diferentes hyperparameters utilizados no melhor modelo treinado, as definições estão representadas na tabela 4.

Tabela 4 - Tabela dos HyperParameters do melhor modelo treinado

Hyperparameter	***Type***	***Search Interval or set***	***Chosen Value***
L2 *Regularization*(λ)	*Real-valued*	[0.00001, 0.01]	0.01

Number of Convolutional Layers	*Integer*	[2, 4]	4
Number of Filters	*List*	[64, 128, 256, 512]	[64, 128, 256, 512]
Kernel Size	*Integer*	3	3
Max Pooling	*Boolean*	[True, False]	True
Dropout Rates	*List*	[0.3, 0.5]	0.3
Dense Layer Units	*Integer*	[128, 512]	256
Activation Functions	*List*	[‘relu’, ‘softmax’]	‘relu’
Batch size	*Integer*	[16, 128]	16
Learning rate	*Real-valued*	[0.00005, 0.01]	0.001
Epochs	*Integer*	[50, 10000]	500

6.4.Escolha do Modelo

De modo a conseguirmos escolher o melhor modelo, precisámos de analisar outros parâmetros para alem da precisão, como por exemplo o *Recall* e o *F1-score*.

Utilizando então os últimos 3 modelos com 98.3% de precisão, vamos analisar as métricas da precisão fornecidas para cada modelo.

6.4.1. Resumo das Métricas

Na tabela 5, encontra-se um resumo das métricas dos 3 melhores modelos, com esta tabela conseguimos uma interpretação mais fácil dos valores.

Tabela 5 - Resumo de métricas dos modelos

	Precision	*Recall*	*F1-score*
Macro Average Modelo 1	0.9827	0.9847	0.9836
Weighted Average Modelo 1	0.9837	0.9836	0.9836
Macro Average Modelo 2	0.9829	0.9847	0.9837
Weighted Average Modelo 2	0.9839	0.9836	0.9837
Macro Average Modelo 3	0.9832	0.9845	0.9837
Weighted Average Modelo 3	0.9841	0.9836	0.9838

6.4.2. Análise das Métricas

Na seguinte tabela, tabela 6, está representado o resumo da precisão dos diferentes modelos.

Tabela 6 - Resumo da precisão dos modelos

	Macro Average	*Weighted Average*
Precision do modelo 1	0.9827	0.9837
Precision do modelo 2	0.9829	0.9839
Precision do modelo 3	0.9832	0.9841

Analisando as métricas de todos os modelos podemos notar que apresentam resultados semelhantes a nível da precisão, mas o modelo 3 apresenta um resultado ligeiramente superior aos restantes.

6.4.3. Decisão

Como neste cenário a Precisão é a métrica mais relevante e importante, é nela que vamos focar para tomar a decisão do melhor modelo.

Macro Average: O modelo 3 é o que tem a maior precisão, com 0.9832

Weighted Average: O modelo 3 também é o que tem a maior precisão, com 0.9841

Por fim após todas as análises de parâmetros entre os diferentes modelos, podemos concluir que o modelo 3 seria a melhor escolha para identificar um gesto, isto porque apresenta a maior precisão tanto na *Macro Average* como na *Weighted Average*, resultando num modelo capaz de identificar gestos com a maior precisão. Também é o modelo que conta com uma taxa de falsos positivos mais baixa devido à sua alta precisão.

6.5. Desenvolvimento de Protótipo Demonstrativo

Nesta etapa vamos abordar o desenvolvimento do protótipo escolhido para demonstrar o funcionamento do projeto da melhor maneira possível com os recursos disponíveis

6.5.1. Escolha do Protótipo

Por diversos motivos não podemos utilizar um drone verdadeiro para a demonstração deste projeto, apesar de ter sido a primeira forma pensada.

Os principais motivos para não termos utilizado um drone verdadeiro foram:

- Segurança: Usar um drone real viria a apresentar riscos de segurança tanto para quem o estivesse a utilizar como para quem tivesse ao redor. O drone poderia cair causando ferimentos ou danos materiais por alguma falha técnica envolvendo o drone ou a comunicação.
- Ambiente controlado: De maneira a testar e desenvolver este projeto precisávamos de um ambiente controlado sem quaisquer tipos de obstáculos que viessem a prejudicar o funcionamento do drone. Os drones reais também são afetados pelas condições climatéricas como o vento ou a chuva, ao esperar por condições ideais para o teste do projeto iriamos perder tempo valioso que poderia ser utilizado a resolver problemas ou a melhorar o projeto.
- Reparações e Downtime: Possivelmente durante o projeto iriamos estar sujeitos ao desgaste do drone e a danos, resultando assim num tempo de inatividade enquanto ocorria a reparação, prejudicando gravemente o teste do projeto.
- Tempo: De modo a podermos ultrapassar os motivos anteriormente referidos necessitávamos de tempo e de testes intensivos, mas devido ao prazo de entrega não nos foi possível ultrapassar esses motivos.

Estes foram os principais motivos para não utilizarmos um drone verdadeiro, apesar de mais realista e mais apelativo descartamos esta opção.

Após descartarmos a opção anterior optámos pela realização de um protótipo virtual 3D, simulando assim o funcionamento semelhante ao de um drone verdadeiro, mas sem envolver todas as questões de segurança e todos os problemas associados.

6.5.2. Desenvolvimento do Protótipo

Começamos por procurar um modelo 3D de um drone que nos permitisse obter um resultado semelhante ao de um drone verdadeiro. Após a procura por diversos websites encontrámos o modelo ideal, um modelo realista e com uma licença que nos permite utilizar o modelo em projetos desde que não o alteremos.

Figura 55 - Imagem do drone utilizado no desenvolvimento do protótipo

(panchalyogendra2194, 2024)

Após a escolha do nosso modelo começamos a ver como aplicá-lo num ambiente virtual, para isto usámos a framework Three.Js do Java*script* em conjunto com as principais bibliotecas GLTFLoader de maneira a conseguirmos carregar o modelo para a cena do Three.Js.

No que se relaciona à arquitetura usamos um esquema server side para o processamento e cliente side para a apresentação. No server side mantivemos o uso da linguagem Python, isto devido a ter sido usada para vários *script*s relacionados a interação dos dados, IA e modelos. Com recurso a biblioteca *websockets* permite-nos a criação de um canal de comunicação na porta e endereço desejados. Por uma questão de conveniência e isenção de custos mantivemos este ambiente local, no endereço: http://127.0.0.1:8080/

Para o servidor utilizamos o programa http-server, uma tecnologia baseada em node.js, que torna possível o acesso ao endereço supra mencionado.

6.5.3. *Server-side* – Codigo e Configurações

No *server-side* tivemos como base o ficheiro ModelRealTimePredicition.py, já anteriormente explicado no relatório, que faz a captura de dados e dá a previsão do

movimento em tempo real, utilizando o modelo previamente treinado, e também já descrito e explicado anteriormente. Com base no *script* do ficheiro fizemos algumas modificações para que após a leitura dos dados capturados pelos sensores gerasse a previsão do modelo e enviasse diretamente ao cliente através dos *websockets* utilizando a porta: 8081.

```
async def handle_message(websocket, message):
    try:
        data = json.loads(message) # Analisar a mensagem JSON recebido
        await websocket.send(json.dumps("recording")) # Envia uma
mensagem ao cliente para iniciar a contagem regressiva de gravação
        await asyncio.sleep(0.1)
        data = await getMove() # Recebe os dados de movimento e a
previsão do modelo
        await websocket.send(json.dumps(data)) # Envia os dados de
movimento ao cliente em formato JSON
        await asyncio.sleep(0.1)
    except json.JSONDecodeError as e:
        print(f"Failed to decode JSON: {e}")

async def send_commands(websocket, path):
    print("Client connected.")
    async for message in websocket: # Ciclo para lidar com mensagens
recebidas
        await handle_message(websocket, message)

# Configura o servidor para enviar e receber mensagens do cliente na
porta 8081
start_server = websockets.serve(send_commands, "localhost", 8081)

# Inicia o servidor e executa o ciclo de eventos
asyncio.get_event_loop().run_until_complete(start_server)
# Deixa o servidor a correr indefinidamente
asyncio.get_event_loop().run_forever()
```

Figura 56 - Código Python de conexão entre servidor e cliente

O código acima é referente ao ficheiro *server.py* que através de *websockets*, permite estabelecer as comunicações necessárias entre o lado do servidor e o lado do cliente, lidando com as diferentes respostas que pode receber, também define o endereço e portas onde vai ocorrer a comunicação.

A restante lógica de programação relativa à previsão do movimento por parte do modelo de ML é a mesma já apresentada anteriormente.

```python
async def getMove():
    user_input = ''

    # Record the movement
    samples_data = record_movement()

    # Feed the data to the model
    prediction = predict_movement(samples_data)

    # Print the prediction
    print("Prediction:", prediction)

    # percentage 0-100% of the model's confidence in the (movement) class
    if prediction[0] > 75:
        print("Movimento: Nenhum (noise)")
        move = "noise"
    elif prediction[1] > 75:
        print("Movimento: Up")
        move = "up"
    elif prediction[2] > 75:
        print("Movimento: Down")
        move = "down"

    ...
```

Figura 57 - Código Python de previsão de gesto

6.5.4. *Client-side* – Código e configurações

No lado do cliente criamos um cenário em *ThreeJS* de raiz, sendo que os principais objetos da cena são uma camara, o drone e luzes ambiente e direcionais, para que as texturas possam ser visualizadas corretamente. Alem disso foi também colocado um cenário como fundo de modo a proporcionar uma maior integração no contexto do problema.

```
//É criada uma cena com um background meramente enquadrativo, definida
uma camara estatica a apontar para esse background e a sua posição, assim
como o renderizador (WEBGLRenderer)
const camera = new THREE.PerspectiveCamera(75, window.innerWidth / win-
dow.innerHeight, 0.1, 1000);
const renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);

//É adicionada á cena uma lz ambiente e uma luz direcional, para iluminar
o drone em qualquer parte da cena
const light = new THREE.AmbientLight(0x808080); // Stronger ambient light
scene.add(light);
const directionalLight = new THREE.DirectionalLight(0xffffff, 0.8);
scene.add(directionalLight);
```

Figura 58 - Código Javascript para criação da cena

No código acima representado é criada uma cena utilizando um *background* representativo de uma floresta, define-se uma camara estática a apontar para o *background*, define-se o renderizador e adiciona-se luz ambiente e direcional.

```
//Com a cena preparada importamos o modelo escolhido para representar
o  drone através da biblioteca GLTFLoader e definida a sua textura.
const loader = new THREE.GLTFLoader();
let drone;
loader.load('drone-model.glb', function (gltf) {
    drone = gltf.scene;
    scene.add(drone);
    drone.position.set(0, -3, 0);  // Position the drone at the origin
    drone.scale.set(1, 1, 1);  // Scale the drone if necessary

    //aplicação da textura
    const droneTexture = new
THREE.TextureLoader().load('textures/droneTex.png'); // Replace with your
texture path
    drone.traverse((node) => {
        if (node.isMesh) {
            node.castShadow = true;
            node.receiveShadow = true;
            node.material = new THREE.MeshStandardMaterial({ map:
droneTexture }); // Apply the texture
        }
    });
```

Figura 59 - Código Javascript para importação do modelo

Importamos o modelo do drone anteriormente mencionado e selecionado e aplicamos uma textura a esse objeto para tornar o drone mais realista.

```
//É instanciado o endereço do websocket para onde irá receber e enviar
mensagens, que será o mesmo endereço que usa o servidor (localhost:8081).
//Inicializa-se a conecção com o servidor.
    const socket = new WebSocket('ws://localhost:8081'); // same as
server

    socket.onopen = function () {
        console.log('WebSocket connection opened');
    };

    socket.onmessage = function (event) {
        console.log('WebSocket message received:', event.data);

        if (command === 'recording') {
            // Atualiza o tempo de gravação na tela começando em 2.00
segundos para 0.00 segundos diminuindo 0.01 segundos
            let recordingTime = 2.00;
            document.getElementById('recording-time').style.color =
'red';
            const recordingInterval = setInterval(() => {
                recordingTime -= 0.01;
                document.getElementById('recording-time').innerText =
recordingTime.toFixed(2);
                if (recordingTime < 0.01) {
                    clearInterval(recordingInterval);
                    document.getElementById('recording-time').innerText =
'0.00';
                    document.getElementById('recording-time').style.color
= 'white';
                }
            }, 10); // Ajusta o tempo do intervalo conforme necessário
            return;
        }

    socket.onclose = function () {
        console.log('WebSocket connection closed');
        if (moveInterval) {
            clearInterval(moveInterval);
        }
    };

    socket.onerror = function (error) {
        console.error('WebSocket error:', error);
        if (moveInterval) {
            clearInterval(moveInterval);
        }
    };
```

Figura 60 - Instanciação do *websocket* no cliente-side

No código apresentado acima é onde instanciamos o endereço do *websocket* no lado do cliente, no nosso caso: ws://localhost:8081, que é o mesmo que o do servidor (localmente), abrimos a conexão e por fim, aquando da gravação do movimento, atualizamos o tempo de gravação no canto do ecrã começando em 2 segundos e diminuindo progressivamente 0.01 segundos.

```
// Definição de parametros
let speed = 0.02;
let moveInterval;
let moveCommand;
let droneFlying = false;
let droneStartingPosition = drone.position;
const maxheight = 4;
const minheight = -5;
const maxwidth = 9.5;
const minwidth = -9.5;
const maxdepth = 4;
const mindepth = -7.5;

socket.onopen = function () {
    console.log('WebSocket connection opened');
};
```

Figura 61 - Definição de parâmetros

No código acima declaramos as variáveis gerais utilizadas no código e fazemos uma verificação de quando a conexão do *websocket* é aberta, de maneira a termos essa confirmação e possamos começar a utilizar o drone.

```
socket.onmessage = function (event) {
        console.log('WebSocket message received:', event.data);
        const command = JSON.parse(event.data);

        if (command === 'cut') {
            moveCommand = null;
            document.getElementById('current-movement').innerText =
"Hovering";
            if (moveInterval) {
                clearInterval(moveInterval);
            }
            return;
        }

        if (command === 'noise' || command === 'unknown') {
            return;
        }

        if (command === 'recording') {
            // update recording time on the screen starting on 2.00
seconds to 0.00 seconds decreasing 0.01 seconds
            let recordingTime = 2.00;
            document.getElementById('recording-time').style.color =
'red';
            const recordingInterval = setInterval(() => {
                recordingTime -= 0.01;
                document.getElementById('recording-time').innerText =
recordingTime.toFixed(2);
                if (recordingTime < 0.01) {
                    clearInterval(recordingInterval);
                    document.getElementById('recording-time').innerText =
'0.00';
                    document.getElementById('recording-time').style.color
= 'white';
                }
            }, 10); // Adjust interval time as needed
            return;
        }
```

Figura 62 - Código Javascript com evento que lida com mensagens recebidas

A função *onmessage* é despoletada quando o cliente recebe uma mensagem do servidor a que nós ligámos anteriormente, e que a partir dessa mensagem façamos a restante logica para controlar o drone, neste excerto verificamos o movimento executado, caso este seja cut, o drone irá parar o movimento que estiver a fazer, caso seja *noise*, o drone irá continuar a

executar o movimento pois nenhum movimento propositado foi feito e caso seja *recording* significa que o utilizador está a realizar um gesto então devemos aguardar que termine.

```
moveCommand = command;
// update the drone movement on the screen with the new movement
document.getElementById('current-movement').innerText = moveCommand;
        if (droneFlying || moveCommand === 'clap') {
            moveInterval = setInterval(() => {
                switch (moveCommand) {
                    case 'up':
                        if (drone.position.y >= maxheight) {
                            break;
                        } else {
                            drone.position.y += speed;
                            break;
                        }
                    case 'down':
                        if (drone.position.y <= minheight) {
                            break;
                        } else {
                            drone.position.y -= speed;
                            break;
                        }
                    case 'left':
                        if (drone.position.x <= minwidth) {
                            break;
                        } else {
                            drone.position.x -= speed;
                            break;
                        }
                    case 'right':
                        if (drone.position.x >= maxwidth) {
                            break;
                        } else {
                            drone.position.x += speed;
                            break;
                        }
                    case 'front':
                        if (drone.position.z >= maxdepth) {
                            break;
                        } else {
                            drone.position.z += speed;
                            break;
                        }
                    case 'back':
                        if (drone.position.z <= mindepth) {
                            break;
                        } else {
                            drone.position.z -= speed;
                            break;
                        }
```

Figura 63 - Código Javascript para identificação do gesto executado

No excerto acima, atualizamos o texto do elemento *movimento atual* e após a atualização fazemos uma verificação com um *switch* que identifica o gesto executado e com faz mover o drone de acordo com o movimento desejado.

Para isso, o drone deverá já estar a voar, sendo feito o movimento *clap* (palmas), que faz com que o drone levante voo. Só assim, com o drone em funcionamento, são permitidos outros movimentos.

```
if (moveCommand === 'clap') {
            if (!droneFlying) {
                // play the sound
                if (!sound.isPlaying) {
                    changeSoundTo('takeoff');
                    //sound.play();
                }
                takeOff = setInterval(() => {
                    if (drone.position.y >= 0) {

                        moveCommand = null;
                        document.getElementById('current-
movement').innerText = "hovering";
                        if (takeOff) {
                            clearInterval(takeOff);
                        }

                        droneFlying = true;
                        if (currentSound != 'flying') {
                            changeSoundTo('flying');
                            currentSound = 'flying';
                        }
                        return;
                    } else {
                        console.log('Taking off:', drone.position);
                        // label the current movement
                        document.getElementById('current-
movement').innerText = "Taking off";
                        drone.position.y += speed;
                    }
                }, 50);
```

Figura 64 - Código Javascript para pousar/levantar voo

O comando *clap* é também utilizado para retornar e pousar o drone na posição inicial, processo esse executado através do código abaixo.

```
if (Math.abs(drone.position.y - droneStartingPosition.y) < 0.05 &&
Math.abs(drone.position.x - droneStartingPosition.x) < 0.05 &&
Math.abs(drone.position.z - droneStartingPosition.z) < 0.05){
    console.log('Landed:', drone.position);
    moveCommand = null;
    document.getElementById('current-movement').innerText = "Landed";
    if (land) {
        clearInterval(land);
    }
    clearInterval(moveInterval);
    droneFlying = false;
    // stop the sound
    if (sound.isPlaying) {
        sound.stop();
    }
    return;
} else {
    // label the current movement
    document.getElementById('current-movement').innerText = "Landing";
    console.log('Landing:', drone.position);
    if (Math.abs(drone.position.y - droneStartingPosition.y) > 0.05) {
        if (drone.position.y < droneStartingPosition.y) {
            drone.position.y += speed;
        } else {
            drone.position.y -= speed;
        }
    }
    if (Math.abs(drone.position.x - droneStartingPosition.x) > 0.05) {
        if (drone.position.x < droneStartingPosition.x) {
            drone.position.x += speed;
         } else {
             drone.position.x -= speed;
         }
    }
    if (Math.abs(drone.position.z - droneStartingPosition.z) > 0.05){
         if (drone.position.z < droneStartingPosition.z) {
             drone.position.z += speed;
         } else {
             drone.position.z -= speed;
         }
    }
}
```

Figura 65 - Código javascript para retornar à posição inicial

No código acima o sistema verifica se o drone está na posição inicial, se não estiver faz o drone movimentar se em direção a esta, até atingir uma distancia inferior a 0.05 da posição inicial em todos os eixos (x,y,z), momento em que pousa.

```
    break;
          default:
              console.log('Unknown command:', moveCommand);
            }
              console.log('Drone position:', drone.position);
            }, 100); // Adjust interval time as needed
        }
    };

    socket.onclose = function () {
        console.log('WebSocket connection closed');
        if (moveInterval) {
            clearInterval(moveInterval);
        }
    };

    socket.onerror = function (error) {
        console.error('WebSocket error:', error);
        if (moveInterval) {
            clearInterval(moveInterval);
        }
    };
```

Figura 66 - Código Javascript do fecho do método *Switch*

Acima encontra-se o fecho do método *switch*, como também uma verificação e conjunto de ações para quando o *socket* é fechado ou encontra um erro.

Desta forma o modelo recebe os dados corretamente e temos também um maior controlo sobre o sistema o que evitaria que num cenário real alguns problemas relacionados com *failsafe*.

Por fim, foi implementado um controlo de áudio para melhorar a experiência de utilização do software, que reproduz efeitos sonoros quando o drone levanta voo, pousa no ar, se movimenta ou procede á aterragem. Esta gestão de som é feita por todo o código, consoante o movimento a ser executado. Sendo importado para a cena de início com o seguinte código:

```
// Add audio to the drone
    const audioLoader = new THREE.AudioLoader();
    sound = new THREE.PositionalAudio(listener);
    audioLoader.load('sounds/takeoff.mp3', function (buffer) {
        sound.setBuffer(buffer);
        sound.setVolume(1);
        sound.setRefDistance(20);
        //sound.play();
    });
    drone.add(sound);
```

Figura 67 - Código Javascript para execução do som do drone

6.6. Testes Finais e Validação de Protótipo

No presente capítulo vamos abordar a fase de testes e de validação do nosso protótipo virtual, para isso conectámos o microcontrolador e iniciamos os scripts necessários. Depois de tudo pronto começámos a testar os gestos.

Figura 68 - Drone a Levantar

Figura 69 - Drone a Ir Para Trás

Figura 70 - Drone a Pairar

Figura 71 - Drone a Descer

Figura 72 - Drone a Girar

Figura 73 - Drone a Pousar

Como podemos concluir pelas imagens anteriores o protótipo funciona da maneira pretendida e com uma resposta rápida e precisa.

7. Conclusão

O projeto DroneGest demonstrou a viabilidade e a eficiência do controlo de drones através de gestos, utilizando uma interface e algoritmos de ML. A implementação dos sensores para a captura de movimentos, em vez de câmaras, mostrou-se uma escolha acertada, proporcionando uma maior liberdade de movimento, um custo reduzido e uma maior velocidade na resposta dos comandos.

7.1. Objetivos Alcançados

Os principais objetivos do projeto foram atingidos com sucesso:

1. Interpretação de Gestos Dinâmicos:
   - O reconhecimento de gestos dinâmicos através dos sensores foi implementado eficazmente, permitindo a identificação precisa dos movimentos do utilizador.
   - A utilização de acelerómetros e giroscópios (MPU6050) conectados a um microcontrolador possibilitou a medição precisa da aceleração e rotação em três eixos coordenados (x, y, z).
2. Desenvolvimento e Treino do Algoritmo:
   - Foi criado um *dataset* específico para o projeto, com gestos personalizados para as ações de controlo do drone.
   - O algoritmo de ML foi treinado com sucesso utilizando este *dataset*, alcançando altos níveis de precisão e *recall*, conforme evidenciado pelos resultados obtidos (*Precision*: 0.9832, *Recall*: 0.9845, *F1-score*: 0.9837).
3. Protótipo Funcional:
   - Um protótipo foi desenvolvido, capaz de interpretar os gestos e simular os movimentos de um drone numa interface de visualização 3D.
   - Este protótipo mostrou-se funcional e eficaz, validando a abordagem proposta.

7.2.Impacto e Potencial Futuro

Os resultados do projeto DroneGest indicam não apenas a viabilidade do controlo de drones através de gestos, mas também o potencial da aplicação desta tecnologia em diversas áreas. Alguns exemplos são tarefas industriais e operações de resgate, onde o controlo ágil de drones é essencial. A capacidade de controlar drones com gestos pode abrir novas oportunidades para a automatização e a interação entre um humano e uma máquina, aumentando a eficiência e a segurança em operações críticas.

7.3.Trabalho Futuro

Futuramente temos como objetivo expandir o número de gestos para melhorar e facilitar ainda mais a interação com o drone, consecutivamente, devido ao aumento do número de gestos teremos que aumentar o tamanho do *dataset* e respetivas classes, mas não só com os gestos novos como com os gestos atuais, melhorando assim a precisão do modelo e mantendo o *dataset* balanceado, aumentando assim o grau de confiança no modelo, pretendemos também a criação de um *dataset* para esquerdinos, de modo a aumentarmos a quantidade de utilizadores que podem usufruir do nosso projeto. Em questões práticas está nos planos o desenvolvimento de uma pulseira pequena com conexão Wireless de modo a conseguirmos controlar o drone mais facilmente e de maneira mais cómoda e discreta, tendo a conexão Wireless feita o último objetivo de trabalho futuro é a implementação num drone real, passando por fases de teste e validação em ambientes seguros e posteriormente em ambientes comuns.

7.4.Considerações Finais

O sucesso do projeto DroneGest ressalta a importância da investigação e desenvolvimento na interseção entre ML e as interfaces gestuais. O uso dos sensores como acelerómetros e giroscópios, aliado a algoritmos avançados de ML, demonstra como é possível criar sistemas de controlo inovadores e eficazes. O projeto não só alcançou os seus objetivos principais, mas também estabeleceu uma base sólida para futuras pesquisas e desenvolvimentos nesta área promissora.

Bibliografia ou Referências Bibliográficas

Semantic Scholar - https://www.semanticscholar.org/paper/Continuous-Arm-Gesture-Recognition-Based-on-Natural-Wu-Wu/3968dd5cf0ea78a66897423b87de68984a06c4d4 - Consultado a 07/05/2024

Ansel Barret. (2022, March 8). *8 Machine Learning Themes You Need To Know*. https://www.octoparse.com/blog/8-keywords-must-know-for-machine-learning

Convolutional Neural Network (CNN) in Machine Learning - GeeksforGeeks. (n.d.). Retrieved March 25, 2024, from https://www.geeksforgeeks.org/convolutional-neural-network-cnn-in-machine-learning/

DANIEL FILIPE BAPTISTA FERREIRA DA SILVA. (2021). *Pré-processamento de Dados e Comparação entre Algoritmos de Machine Learning para a Análise Preditiva de Falhas em Linhas de Produção para o Controlo*. https://recipp.ipp.pt/bitstream/10400.22/18266/1/DM_DanielSilva_2021_MEI.pdf

Fábio Guimarães. (2018, February 21). *Acelarometro MPU6050 com o arduino*. https://mundoprojetado.com.br/acelerometro-mpu6050-arduino/

Github. (n.d.). https://github.com/

Hadri, S. (2018). *HAND GESTURES FOR DRONE CONTROL USING DEEP LEARNING*. https://doi.org/10.13140/RG.2.2.15939.02089

Hu, M., Li, J., Jin, R., Shi, C., Xu, L., & Liu, R. (2024). *HGIC: A Hand Gesture Based Interactive Control System for Efficient and Scalable Multi-UAV Operations*. http://arxiv.org/abs/2403.05478

Kaggle. (n.d.). Retrieved March 20, 2024, from https://www.kaggle.com/

Leandro Castro. (2021, April 24). *5 passos para utilizar o sensor de inclinacao giroscopio*. https://blog.arduinoomega.com/5-passos-para-utilizar-o-sensor-de-inclinacao-giroscopio/

Matheus Silva Pereira. (2022). *Reconhecimento de Gestos Estáticos da Mão a partir de Rede Neural Artificial e Coordenadas do Esqueleto*. https://bdm.unb.br/bitstream/10483/34512/1/2022_MatheusSilvaPereira_tcc.pdf

Mohammed, H., & Waleed, J. (2023). Hand gesture recognition using a convolutional neural network for arabic sign language. *AIP Conference Proceedings*, *2475*, 70012. https://doi.org/10.1063/5.0104256

NANASHI. (2019, April 12). *#1 Smart Robots. Most Complete Notebook*. Version 43. https://www.kaggle.com/code/jesucristo/1-smart-robots-most-complete-notebook/input?select=X_train.csv

panchalyogendra2194. (2024). *Drone 3D*. https://sketchfab.com/3d-models/drone-3d-5b9def2ac64c4a4285035ada65e4f0c1

PyTorch. (n.d.). Retrieved March 24, 2024, from https://pytorch.org/

SAP. (n.d.). *What is Machine Learning*. Retrieved May 1, 2024, from https://www.sap.com/portugal/products/artificial-intelligence/what-is-machine-learning.html

Sebastian Krysmanski. (n.d.). *Quaternion and normalization*. Retrieved April 10, 2024, from https://stackoverflow.com/questions/11667783/quaternion-and-normalization

Software | Arduino. (n.d.). Retrieved March 25, 2024, from https://www.arduino.cc/en/software

TensorFlow. (n.d.). Retrieved March 23, 2024, from https://www.tensorflow.org/?hl=pt-br

TensorFlow Lite para microcontroladores. (n.d.). Retrieved March 23, 2024, from https://www.tensorflow.org/lite/microcontrollers?hl=pt-br

Anexos

***dataset* que encontramos com valores devolvidos de sensor:**

https://www.*Kaggle*.com/code/jesucristo/1-smart-robots-most-complete-notebook/input?select=X_train.csv

```
# se existirem, duplicar os ficheiros antes de usar (como backup)
if file_exists and validation_file_exists:
    shutil.copyfile(data, "backups/" + data + "_" + formatted_date +
"_backup.csv")
    shutil.copyfile(validation_data, "backups/" + validation_data + "_" +
formatted_date + "_backup.csv")
    print("Backup Files Created.")


# Open input and output files for appending
with open(data, "a") as data_file, open(validation_data, "a") as
validation_file:
    # Assuming you also want to read the existing content for reference
    with open(data, "r") as data_file_read, open(validation_data, "r") as
validation_file_read:
        # Find the last non-empty line in data_file_read
        last_line = None
        for line in reversed(list(data_file_read)):
            if line.strip():  # Check if the line is not empty
                last_line = line
                break
        last_group_id = last_line.split(',')[0] if last_line else 0
        start_number = int(last_group_id) + 1
        copied_group_id = 0
        data_file_read.seek(0)

        for i, line in enumerate(data_file_read, start=1):
            # Split the line into parts and skip fist line
            if i == 1:
                continue

            parts = line.strip().split(",")
            if len(parts) >= 7:
                # Extract data
                data = ",".join(parts[1:])
                # Write original group_id
                copied_group_id = int(parts[0])

                # Write original data
                data_file.write(f"{start_number},{data}\n")
            if i % 64 == 0:
                # Extract movement by searching for the group id in the
validation_data file
                validation_file_read.seek(0)
                for j, line2 in enumerate(validation_file_read, start=1):
                    if j == 1:
                        continue

                    parts2 = line2.strip().split(",")
```

Figura 74 - Código Python para duplicação de dados

Glossário

Accuracy: é uma medida de avaliação usada em modelos de classificação que mede a proporção de previsões corretas em relação ao total de previsões feitas. É calculada como o número de previsões corretas dividido pelo número total de previsões. A *accuracy* é útil quando as classes estão balanceadas, mas pode ser enganosa em casos de classes desbalanceadas.

Amostra: normalmente uma linha num *dataset* que representa 1/total de dados. Aplicado ao problema em questão, uma amostra representa 1/32 dos dados capturados por segundo.

Batch size: Número de amostras processadas antes de atualizar os pesos do modelo. Afeta a estabilidade e velocidade do treino. OS valores padrão variam entre 16 e 128.

Bulk: Refere-se ao processo de usar grandes volumes ao mesmo tempo em vez de apenas um de cada vez. No contexto de treinamento de modelos de inteligência artificial, bulking significa o treino de vários modelos de uma vez.

Curvas ROC: Gráfico da taxa de verdadeiros positivos em relação à taxa de falsos positivos em cada limiar. Representa o desempenho de um modelo de classificação em todos os limiares de classificação.

dataset: Um *dataset* é um conjunto ou uma coleção organizada de informações ou dados, geralmente armazenados em formato tabelado, que podem ser utilizadas para análise, pesquisa ou processamento para os mais diversos fins. Aplicado ao problema em questão o *dataset* é usado para treinar modelos de machine learning.

Dense: Camada totalmente conectada onde cada neurônio está conectado a todos os neurônios da camada anterior numa rede neural. Usada para aprender combinações não-lineares complexas dos dados de entrada.

Dropout: Técnica de regularização usada em redes neurais para prevenir o *overfitting*. Durante o treino, desativa aleatoriamente uma fração das unidades da camada, forçando a rede a ser independente de unidades específicas.

ESP32-C6: Microcontrolador avançado da Espressif que inclui um núcleo de CPU RISC-V de 32 bits e suporte para Wi-Fi 6 e Bluetooth 5.0.

IMU: Componente eletrônico que utiliza uma combinação de acelerômetros, giroscópios e magnetômetros para medir a aceleração, velocidade angular e orientação do sensor.

Learning rate: Taxa de aprendizagem, medida de ajuste dos pesos durante a otimização. Um valor adequado acelera a convergência do modelo, enquanto um valor inadequado pode levar a resultados subótimos.

Max*Pooling*: Função de *Pooling* usada em redes neurais convulsionais para reduzir a dimensionalidade dos dados, retendo as características mais importantes.

Plataforma *Kaggle*: O *Kaggle* é uma plataforma online focada no aprendizado de máquina onde junta vários recursos e entusiastas da área tais como desenvolvedores de inteligência artificial. É amplamente utilizado tanto por iniciantes quanto por profissionais experientes para melhorar as suas capacidades em IA e aprendizado de máquina, além de ser uma ferramenta útil para aprender e partilhar de forma open-source projetos pessoais e por vezes associados a concursos dentro da própria plataforma.

Python: linguagem de programação com um grande potencial, conhecida pela sua sintaxe clara e legibilidade. É amplamente usada para desenvolvimento web, análise de dados, automação de tarefas e desenvolvimento de software, com uma vasta lista de bibliotecas e uma comunidade que contribui ativamente para o crescimento da mesma.

Reduce Learning rate: Função que reduz a taxa de aprendizagem quando a métrica monitorada, no caso a precisão, para de melhorar.

Early Stopping: Função que termina automaticamente o treino do modelo se a métrica monitorada, no caso a precisão, parar de melhorar.

Epoch: Iteração completa por todo o conjunto de dados durante o treino de um modelo, utilizada para atualizar os pesos da rede neural após passar por todas as amostras.