POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

AI Driving Classification

Licenciatura em Engenharia Informática

Alberto Manuel de Matos Pingo, nº 2202145

João Pedro Quintela de Castro, nº 2201781

Leiria, setembro de 2024

POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

AI Driving Classification

Licenciatura em Engenharia Informática

Alberto Manuel de Matos Pingo, nº 2202145

João Pedro Quintela de Castro, nº 2201781

Trabalho de Projeto da unidade curricular de Projeto Informático realizado sob a orientação do Professor Sílvio Priem Mendes, da Professora Anabela Moreira Bernardino e do Professor Paulo Jorge Gonçalves Loureiro.

Leiria, setembro de 2024

Agradecimentos

Começamos por agradecer aos Docentes orientadores, Sílvio Priem Mendes, Anabela Moreira Bernardino e Paulo Jorge Gonçalves Loureiro pela orientação, apoio e disponibilidade ao longo do desenvolvimento deste projeto, assim como o acompanhamento semanal ao projeto.

Agradecemos também a todos os docentes que nos acompanharam ao longo deste curso. O conhecimento que nos transmitiram foi não só fundamental para a concretização deste projeto, mas também para o nosso crescimento académico e pessoal.

Os nossos agradecimentos vão também para os nossos colegas que realizaram o projeto Aplicação para Rastreio de Viaturas, que nos deram uma ferramenta para obter o conjunto de dados para o desenvolvimento do nosso projeto.

Gostaríamos também de agradecer ao Instituto Politécnico de Leiria, pelos recursos disponibilizados que tornaram possível a concretização deste projeto.

Por último, queremos expressar a nossa gratidão aos nossos colegas, amigos e família que tiveram um papel importante no apoio moral e técnico.

A todos, expressamos o nosso mais sincero agradecimento.

Resumo

Este trabalho apresenta uma investigação sobre o uso de Inteligência Artificial (IA) e técnicas de *Deep Learning* (DL) para analisar e classificar diferentes comportamentos de condução, com base em dados recolhidos por sensores presentes em dispositivos móveis. Em particular, são utilizadas Redes Neuronais Recorrentes (RNN), com foco na variante *Long Short-Term Memory* (LSTM), que se destaca na análise de sequências temporais, como os dados provenientes desses sensores. O objetivo principal deste estudo é desenvolver um sistema de classificação de comportamento de condução capaz de classificar cenários como aceleração, travagem e curvas, utilizando informações capturadas por acelerómetros, giroscópios e GPS de dispositivos móveis.

O projeto foi abordado sob duas perspetivas distintas. Na primeira abordagem, foram analisados dados de uma viagem completa, segmentando os dados do acelerómetro e giroscópio em valores positivos e negativos, o que permitiu uma análise mais detalhada dos padrões de condução. Foram usadas arquiteturas do tipo *Stacked* LSTM, *Bidirectional* LSTM e *Convolutional* LSTM. Na segunda abordagem, pequenos cenários de condução específicos, como manobras de aceleração, travagens, curvas e rotundas, foram capturados e pré-classificadas como "*Slow*", "*Normal*" ou "*Aggressive*". Foram usadas arquiteturas do tipo *Stacked* LSTM e *Bidirectional* LSTM.

A metodologia adotada inclui uma investigação aprofundada do estado da arte e dos fundamentos científicos que sustentam o problema em análise. Segue-se a análise de requisitos, planeamento, desenvolvimento, e treino e teste de diversos modelos LSTM.

Os resultados indicam que os modelos LSTM são eficazes na classificação de comportamentos de condução, com um desempenho geral satisfatório. As conclusões demonstram o potencial da aplicação de IA e dos sensores de dispositivos moveis para supervisionar e melhorar a segurança no trânsito, oferecendo uma solução para a classificação de comportamentos de condução acessível.

Palavras-chave: Inteligência Artificial, Deep Learning, Redes Neuronais Recorrentes, Long Short-Term Memory, Comportamentos de condução, Sensores

Abstract

This project presents an investigation into the use of Artificial Intelligence (AI) and Deep Learning (DL) techniques to analyze and classify different driving behaviours, based on data collected by sensors present in mobile devices. Recurrent Neural Networks (RNN) are used, specifically the Long Short-Term Memory (LSTM) variant, which stands out in the analysis of temporal sequences, as the data coming from these sensors. The main objective of this study is to develop a driving behaviour classification system that is capable of classifying behaviour in scenarios such as acceleration, braking and curves, using information captured by accelerometers, gyroscopes and GPS from mobile devices.

The project was approached from two different perspectives. In the first approach, data from a complete trip was analyzed, segmenting the accelerometer and gyroscope data into positive and negative values, which allowed a more detailed analysis of driving patterns. Architectures such as Stacked LSTM, Bidirectional LSTM and Convolutional LSTM were used. In the second approach, specific small driving scenarios such as acceleration, braking, cornering and roundabout maneuvers were captured and pre-classified as "Slow", "Normal" or "Aggressive". Architectures such as Stacked LSTM and Bidirectional LSTM were used.

The adopted methodology includes an in-depth investigation of the state of the art and the scientific foundations of the problem. Followed by requirements analysis, planning, and the development, training, and testing of various LSTM models.

The results indicate that LSTM models are effective in classifying driving behaviors, with satisfactory overall performance. The findings demonstrate the potential of applying AI and mobile device sensors to oversee and improve traffic safety by offering an accessible solution for classifying driving behaviours.

Keywords: Artificial Intelligence, Deep Learning, Recurrent Neuronal Networks, Long Short-Term Memory, Driving Behaviors, Sensors

Índice

Lista de Figuras

Lista de Listagens

Lista de Tabelas

Lista de siglas e acrónimos

AUC	*Area Under the Curve*
CNN	*Convolutional Neuronal Networks*
DL	*Deep Learning*
ESTG	Escola Superior de Tecnologia e Gestão
IA	Inteligência Artificial
IDE	*Integrated Development Environment*
JSON	*JavaScript Object Notation*
LLM	*Large Language Model*
LSTM	*Long Short-Term Memory*
ML	*Machine Learning*
MCPE	*Mean Per-Class Error*
PCE	*Per-Class Error*
RNN	*Recurrent Neuronal Network*
UML	*Unified Modeling Language*

1. Introdução

O presente projeto, intitulado "AI Driving Classification", investiga o uso de Inteligência Artificial (IA) e técnicas de Deep Learning (DL) para analisar e classificar diferentes comportamentos de condução com base em dados recolhidos por sensores presentes em dispositivos móveis. A motivação para este projeto reside na necessidade de estudar o comportamento ao volante, com o objetivo de tornar as estradas mais seguras. Ao explorar a capacidade de sensores de smartphones de supervisionar a condução em tempo real, pretende-se criar uma solução acessível e eficaz para detetar e classificar diferentes estilos de condução, contribuindo para a prevenção de acidentes e a promoção de uma condução mais responsável.

Uma parte significativa do projeto é dedicada à investigação dos fundamentos, técnicas e tecnologias na área da IA, como Redes Neuronais Recorrentes (RNN) em *Deep Learning* e técnicas de *Machine Learning* (ML). Também se foca na análise de como esses métodos têm sido aplicados em contextos similares, por meio de uma revisão crítica de trabalhos anteriores na área.

1.1. Objetivos

A análise de padrões de condução é fundamental não apenas para melhorar a segurança rodoviária, mas também para o desenvolvimento de sistemas de assistência à condução. Este projeto procura criar uma solução acessível e eficaz de supervisionar em tempo real de estilos de condução, contribuindo para a promoção de comportamentos mais responsáveis ao volante.

O objetivo central deste projeto é desenvolver um sistema de classificação de comportamento de condução que utilize dados capturados por acelerómetros, giroscópios e GPS integrados em dispositivos móveis. Através da aplicação de Redes Neuronais Recorrentes (RNN), em particular da variante *Long Short-Term Memory* (LSTM), pretende-se identificar e classificar diferentes padrões de comportamento de condução, como acelerações, travagens, curvas e rotundas.

Além do desenvolvimento técnico, o projeto também tem como objetivo investigar e compreender os fundamentos teóricos e práticos aplicados ao problema das áreas de Inteligência Artificial, *Machine Learning* e *Deep Learning*.

Por fim, o projeto visa criar uma solução acessível e eficaz que permita a supervisão em tempo real dos estilos de condução, promovendo comportamentos mais responsáveis e, consequentemente, aumentando a segurança rodoviária.

1.2.Metodologia

Neste capítulo, iremos descrever as etapas que possibilitaram a entrega do projeto. O subcapítulo 1.2.1 aborda detalhadamente as reuniões periódicas realizadas durante o desenvolvimento do projeto.

1.2.1. Reuniões

Um dos pilares fundamentais do desenvolvimento do projeto foram as reuniões que foram realizadas ao longo de todo o processo. Nestas reuniões estiveram presentes os orientadores do projeto e os alunos que desenvolveram o mesmo. Esta abordagem seguiu os princípios da metodologia *Scrum*, com reuniões semanais a funcionarem como *sprints*.

Estas reuniões foram realizadas de semana a semana com o intuito de monitorar o processo de desenvolvimento do projeto visando analisar todo o trabalho feito na semana antes da reunião e projetar novos objetivos para a semana seguinte.

Com estas reuniões conseguimos identificar possíveis erros no desenvolvimento do projeto e otimizações que puderam ser feitas e melhorando assim a qualidade do produto entregue.

Foram realizadas 18 reuniões no total, tendo como tópicos abordados os seguintes:

- Planeamento do Projeto
- Introdução ao Tema
- Resultados Esperados
- Apoio ao Desenvolvimento
- Resultado Obtidos
- Estruturar o Artigo Científico
- Correção do Relatório

1.3. Cronograma

A Figura 1 apresenta um cronograma das etapas de desenvolvimento, divididas por trimestres, para a construção das soluções final.

Figura 1 - Cronograma das etapas do projeto.

1.4.Estrutura do documento

O documento está organizado em 8 capítulos, cada um abordando uma parte fundamental no desenvolvimento deste projeto. Os capítulos seguem a seguinte ordem:

- Capítulo 1 - **Introdução** : Apresenta os objetivos do projeto e a metodologia seguida.
- Capítulo 2 - **Fundamentos e Contexto Tecnológico**: Oferece uma base sólida para a compreensão dos conceitos de IA, ML, DL, e as tecnologias usadas no projeto.
- Capítulo 3 - **Análise**: Descreve a análise de requisitos, a aplicação de recolha de dados e os *datasets* utilizados.
- Capítulo 4 - **Soluções Propostas**: Explica as arquiteturas LSTM, *Bidirectional* LSTM e *Convolutional* LSTM aplicadas.
- Capítulo 5 - **Desenvolvimento**: Detalha a implementação do projeto, desde o tratamento de dados até à criação dos modelos.
- Capítulo 6 - **Testes e Resultados**: Apresenta os resultados obtidos com as abordagens de classificação e discute a *performance* dos modelos.
- Capítulo 7 - **Artigo Científico**: Resume o artigo produzido no âmbito do projeto.
- Capítulo 8 - **Conclusão e Trabalho Futuro**: Conclui o trabalho realizado e propõe possíveis trabalhos futuros.
- **Bibliografia**: Lista as referências utilizadas na elaboração do relatório.
- **Anexos**: Contém materiais adicionais e complementares à compreensão do relatório.

2. Fundamentos e Contexto Tecnológico

Este capítulo oferece uma base sólida para a compreensão dos conceitos e tecnologias centrais que sustentam este trabalho. Primeiramente, abordam-se os fundamentos teóricos que estabelecem a base de conhecimento necessária para o desenvolvimento e compreensão do trabalho. Seguidamente, explora-se o contexto tecnológico, detalhando as principais ferramentas e tecnologias utilizadas. Por fim, são analisados os trabalhos relacionados, proporcionando uma visão abrangente do estado da arte e destacando como este trabalho se posiciona em relação a pesquisas anteriores.

2.1. Fundamentos teóricos

Neste capítulo iremos apresentar a fundamentação científica que irá dar suporte ao projeto desenvolvido, nomeadamente iremos abordar os seguintes conceitos: Inteligência Artificial, Redes Neuronais, Redes Neuronais Recorrentes, Arquitetura *Bidirectional*, Redes Convolucionais e Redes Convolucionais Unidimensionais

2.1.1. Sensores de Dispositivos Móveis

Os dispositivos moveis modernos, como *smartphone* e *tablets*, estão equipados com uma variedade de sensores que permitem a captura de dados detalhados sobre o ambiente e os movimentos do dispositivo. Esses sensores são usados numa ampla gama de aplicações, desde jogos a navegação até à classificação de condução.

Os sensores de dispositivos móveis são componentes eletrónicos que detetam e respondem a estímulos físicos do ambiente. Eles convertem esses estímulos em sinais elétricos que podem ser processados pelo dispositivo para diversas finalidades. No caso de *smartphones*, esses sensores são integrados num único *chip*, o que permite que sejam compactos e eficientes em termos de energia.

2.1.1.1. Tipos de Sensores

- **Acelerómetro (ver** Figura 2**)**
    - Função: Deteta mudanças na velocidade do dispositivo ao longo dos três eixos (x, y, z).
    - Aplicações: Útil para detetar movimentos lineares, como acelerações e travagens bruscas.

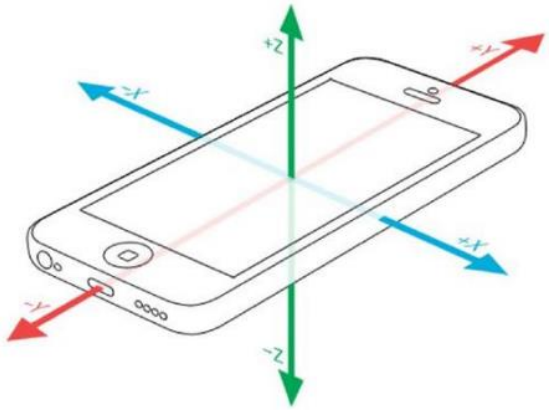

Figura 2 - Representação dos eixos de um acelerómetro. [1]

- **Giroscópio (ver** Figura 3**)**
    - Função: Mede a taxa de rotação do dispositivo em torno dos três eixos (x, y, z).
    - Aplicações: Deteção de mudanças de direção e rotações, como curvas durante a condução.

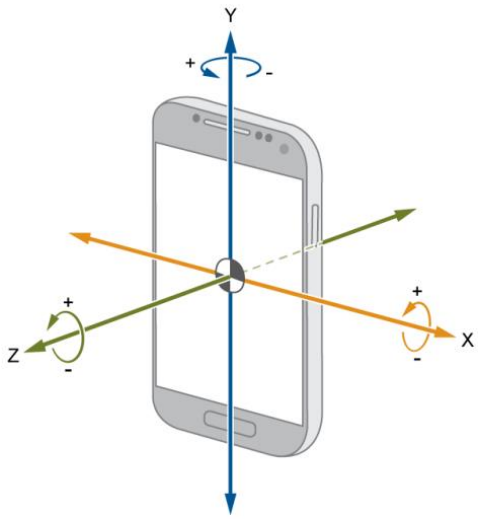

Figura 3 - Representação dos eixos de um giroscópio. [1]

- **GPS (Sistema de Posicionamento Global):**
    - Função: Fornece a localização geográfica precisa do dispositivo, além de informações sobre velocidade e direção.
    - Aplicações: Monitorizar a posição do veículo em tempo real, calcular a velocidade média e traçar rotas.

2.1.2. Inteligência Artificial (IA)

A Inteligência Artificial (IA) emergiu como um campo de estudo e aplicação que está a revolucionar diversos setores e indústrias, trazendo soluções inovadoras e melhorias significativas em processos e produtos.

A origem da IA tem mais de 50 anos, com base nos estudos iniciais de cientistas como Alan Turing, John McCarthy e Marvin Minsky. Entretanto, apenas nos últimos anos é que os progressos tecnológicos, principalmente na área do processamento de dados e algoritmos de aprendizagem, colocaram a inteligência artificial em destaque [2].

Atualmente, a IA é utilizada em diversas aplicações, como assistentes virtuais em dispositivos móveis, sistemas de diagnóstico médico e veículos autónomos. Empresas de todos os tamanhos e áreas estão constantemente à procura de formas de utilizar a inteligência artificial para melhorar a eficiência operacional, personalizar a experiência do cliente e impulsionar a inovação [3].

Além disso, a IA está a mudar o modo como os profissionais realizam as suas tarefas, trazendo novas competências e exigindo um maior conhecimento dos dados e algoritmos. Com o avanço da IA, questões éticas, de transparência, privacidade e segurança dos dados tornam-se cada vez mais relevantes, sendo preciso desenvolver uma abordagem cautelosa e responsável na criação e aplicação de sistemas de IA.

No campo da condução, a inteligência artificial tem sido fundamental para o desenvolvimento dos veículos autónomos. A IA é essencial para que este tipo de veículos consigam entender o ambiente que os rodeiam, tomar decisões rapidamente e navegar com segurança nas estradas. Tecnologias avançadas de IA, como redes neuronais profundas e algoritmos de visão computacional, possibilitam que os veículos autónomos identifiquem sinais de trânsito, pedestres, outros veículos e obstáculos, agindo de forma similar a um ser humano que esteja a dirigir. Além disso, a IA também está a ser utilizada para otimizar rotas, prever condições de tráfego e melhorar a eficiência energética dos veículos, contribuindo para uma condução mais segura, eficiente e autónoma [4].

2.1.1. *Machine Learning* (ML)

Machine Learning (ML) é um subcampo da IA que se dedica a criar algoritmos e modelos que possibilitam os sistemas computacionais aprenderem com dados, fazendo previsões ou decisões sem precisar de serem programados para tal. Neste processo de

aprendizagem, há a procura por padrões em grandes quantidades de dados, o que possibilita ao sistema aprimorar a sua performance gradualmente.

Os modelos de ML podem ser categorizados em três tipos principais: *supervised, unsupervised* e por *reinforcement*. No treino *supervised*, o modelo é treinado com um conjunto de dados rotulados, ou seja, onde o resultado desejado é conhecido. Aplicações em classificação de imagens e a previsão de séries temporais são bons exemplos de uso. No treino *unsupervised*, o modelo analisa dados não rotulados para encontrar padrões ocultos, como *clusters*. Por fim, o treino por *reinforcement* requer que um agente interaja com o ambiente, com o objetivo de aumentar uma recompensa total ao longo do tempo, sendo aplicado em situações como jogos e na manipulação de robôs.

O bom desempenho de um sistema de ML é determinado por diversos aspetos, como a qualidade dos dados, a seleção do modelo correto e a técnica de treino utilizada. Além disso, a compreensão dos resultados é fundamental para assegurar a confiabilidade e utilidade das previsões do modelo em situações práticas [5].

2.1.2. *Deep Learning* (DL)

Deep Learning (DL) é um subcampo de ML que se concentra em modelos baseados em redes neuronais, compostas por várias camadas de nós interligados, que simulam o funcionamento de um cérebro humano. Este tipo de redes neuronais possui a habilidade de adquirir entendimentos complexos e abstratos dos dados através do processamento em diversas camadas.

Uma das principais características de DL é a capacidade de trabalhar com grandes volumes de dados não estruturados, como imagens, texto e áudio. Deep Neuronal Networks (DNN) têm sido amplamente empregues em aplicações como reconhecimento de imagens, processamento de textos, tradução automática e veículos autónomos. A utilização de DNN para treino exige um elevado poder computacional e grandes conjuntos de dados, o que tornou o surgimento destas técnicas mais viável apenas nas últimas décadas, com o avanço do *hardware* e do acesso a grandes bases de dados.

Os principais tipos de redes neuronais utilizadas em DL são as Redes Neuronais Convolucionais (CNN) e as Redes Neuronais Recorrentes (RNN) [6] [7].

2.1.3. Redes Neuronais (RN)

As redes neuronais representam uma abordagem bastante útil e versátil no campo da IA, inspirando-se no funcionamento do cérebro humano.

Redes neuronais, ou sistemas neuronais artificiais, são estruturas computacionais compostas por unidades interconectadas, denominadas neurónios artificiais. Estes neurónios, semelhantes aos neurónios biológicos, recebem *inputs*, processam-nos através de operações matemáticas e produzem saídas. A interconexão de múltiplos neurónios em camadas forma a arquitetura de uma rede neuronal [8].

A arquitetura de uma rede neuronal geralmente inclui uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída [9]. Cada camada contém um conjunto de neurónios interligados, e as conexões entre os neurónios têm pesos que são ajustados durante o treino da rede. Este treino visa minimizar uma função de perda, que quantifica a discrepância entre as previsões da rede e os valores reais, através de algoritmos de otimização.

Existem diversos tipos de redes neuronais, cada uma com arquiteturas e aplicações específicas [10].

As redes neuronais têm uma vasta gama de aplicações em diversos domínios, incluindo reconhecimento de padrões, processamento de linguagem natural, visão computacional, previsão de séries temporais, entre outros. São utilizadas com sucesso em tarefas como reconhecimento de voz, classificação de imagens, tradução automática e diagnósticos médicos.

2.1.4. Redes Neuronais Recorrentes (RNN)

Uma rede neuronal recorrente (RNN) é um modelo de *Deep Learning* treinado para processar e converter uma entrada de dados sequenciais numa saída de dados sequenciais específica.

Um RNN (ver Figura 4) é um tipo de sistema que consiste em muitos componentes interconectados que tentam imitam a maneira como os humanos realizam conversões sequenciais de dados, como traduzir textos de um idioma para outro [11]. Cada vez mais as RNN estão a ser substituídas por IA e LLM, que são muito mais eficientes no processamento sequencial de dados.

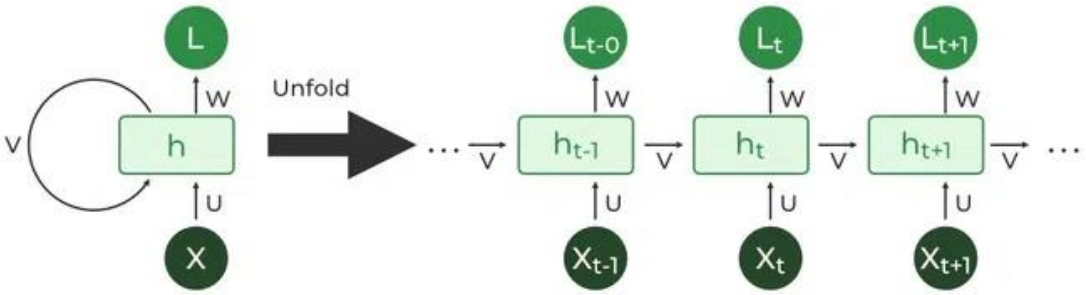

Figura 4 -Diagrama ilustrativo de uma RNN. [12]

2.1.4.1. Tipos de RNN

Nesta secção iremos apresentar os vários tipos de Redes Neuronais Recorrentes.

- ***One-to-Many***

Este tipo RNN canaliza uma entrada para várias saídas. Ele permite aplicações linguísticas como legendagem de imagens, gerando uma frase a partir de uma única palavra-chave [11].

- ***Many-to-Many***

O modelo usa múltiplas entradas para prever múltiplas saídas. Por exemplo, podemos criar um tradutor com uma RNN, que analisa uma frase e estrutura corretamente as palavras num idioma diferente.

- ***Many-to-One***

Várias entradas são mapeadas para uma saída. Isto é útil em aplicações como análise de sentimentos, onde o modelo prevê sentimentos dos utilizadores como positivos, negativos e/ou neutros a partir de depoimentos de entrada.

2.1.4.2. Estrutura e Funcionamento das RNN

Uma RNN é composta por unidades recorrentes que iteram através de uma sequência de dados. Em cada ponto de tempo, a RNN usa uma entrada atual e o estado anterior para produzir uma saída e um novo estado. Este processo é descrito pelas seguintes equações:

$$h_t = \sigma(W_t \cdot x_t + U_h \cdot h_{t-1} + b_h)$$

$$y_t = \emptyset(W_y \cdot h_t + b_y)$$

- h_t é o estado oculto no tempo *t*.
- x_t é a entrada no tempo *t*.
- W_t e U_h são pesos de entrada e recorrentes, respetivamente.
- b_h é o *bias* do estado oculto.
- y_t é a saída no tempo *t*.
- W_y é o peso de saída.
- b_y é o *bias* da saída.
- σ e $\emptyset$ são funções de ativação, geralmente não lineares, como tanh ou ReLU.

As RNN diferenciam-se das redes *feedforward* pela capacidade de utilizarem estados anteriores para influenciar as saídas atuais e isso é o que permite a captura de dependências temporais nos dados.

2.1.4.3. Vantagens e Limitações das RNN

Nesta secção iremos apresentar algumas das vantagens e limitações das Redes Neuronais Recorrentes.

Algumas vantagens são**:**

Capacidade Sequencial: As RNN são especificamente projetadas para lidar com dados sequenciais, o que as torna especialmente adequadas para capturar dependências temporais. A capacidade sequencial das RNN destaca-se por vários fatores, entre eles:

- **Processamento Temporal:** As RNN processam entradas sequenciais, mantendo um estado oculto que é atualizado a cada novo elemento da sequência. Este estado oculto atua como uma memória que armazena informações sobre elementos anteriores da sequência, permitindo que a rede capture dependências temporais.

- **Dependências Temporais:** O facto das RNN manter num estado oculto que é influenciado por toda a sequência faz com que a rede aprenda e modele dependências temporais complexas. Por exemplo, em problemas de reconhecimento de fala, a interpretação de um som pode depender de sons anteriores pertencentes à sequência.

Flexibilidade: Para além da capacidade sequencial, as RNN são altamente flexíveis, podendo ser aplicadas a uma ampla gama de problemas de diferentes tipos. Esta flexibilidade manifesta-se em várias características, tais como:

- **Adaptabilidade a Diferentes Tipos de Dados:** As RNN podem ser configuradas para lidar com diferentes tipos de dados sequenciais, incluindo texto, áudio, vídeo e séries temporais. A mesma arquitetura básica pode ser adaptada para diferentes formatos de entrada, desde palavras numa frase até valores numéricos numa série temporal.
- **Aplicações em Diversos Domínios:** As RNN podem ser utilizadas numa ampla gama de tarefas, incluindo:
  - **Processamento de Linguagem Natural (NLP):** Tradução automática, análise de sentimentos [12].
  - **Previsão de Séries Temporais**: Previsão de vendas, análise financeira, previsão do tempo [13].
  - **Reconhecimento de Padrões**: Reconhecimento de fala, classificação de gestos em vídeos, deteção de anomalias [14].
- **Integração com Outras Tecnologias:** As RNN têm a capacidade de serem integradas com outras tecnologias de *Machine Learning* (ML) e *Deep Learning* (DL), como redes convolucionais para processamento de vídeo ou mecanismos de atenção para melhorar o foco em partes específicas da sequência.
- **Escalabilidade**: As RNN podem ser escaladas para lidar com grandes conjuntos de dados e modelos bastante complexos, ajustando-se a todo o tipo de requisitos computacionais. Técnicas como paralelização e otimização de *hardware*, uso de GPU, têm um impacto positivo ajudando a treinar a rede de forma muito mais eficiente.

Algumas limitações são**:**

Problema do Gradiente Desaparece/Explode: Este problema é uma das principais dificuldades encontradas no treino de RNN. Este problema impacta negativamente a capacidade de a rede aprender dependências de longo prazo nos dados sequenciais, o que é crucial para muitas aplicações práticas.

Durante o treino da rede, o algoritmo *backpropagation* é usado para ajustar os pesos da rede. Este processo envolve o cálculo do gradiente do erro relativamente a cada peso, que é utilizado para atualizar os pesos de maneira a minimizar o erro. No entanto, em redes profundas e RNN, os gradientes podem sofrer um de dois tipos de problemas:

- *Vanishing Gradient Problem* (Gradiente Desaparece):
  - Os gradientes dos estados anteriores tendem a diminuir exponencialmente à medida que a informação é propagada de volta através da rede. Isto resulta em gradientes muito pequenos, o que significa que os pesos destas camadas são atualizados muito lentamente. Como resultado, a rede tem dificuldade em aprender e ajustar estes pesos corretamente, especialmente para sequências longas [15].
- *Exploding Gradient Problem* (Gradiente Explode):
  - Trata-se do oposto do problema do gradiente a desaparecer, sendo que neste problema os gradientes podem aumentar exponencialmente, resultando em valores extremamente grandes. Isto leva a atualizações de pesos muito grandes, que podem causar divergência no processo de treino, tornando o modelo instável e incapaz de aprender [15].

Solução para Resolver as Limitações

Para resolver a dificuldade de capturar dependências de longo prazo, diversas abordagens e melhorias nas arquiteturas de redes neuronais foram desenvolvidas. A mais importante no âmbito deste projeto foi o desenvolvimento da *Long Short-Term Memory* (LSTM), uma variante das RNN projetada especificamente para lidar com o problema do gradiente a desaparecer.

2.1.5. Long Short-Term Memory (LSTM)

LSTM (ver Figura 5) é uma versão melhorada da RNN projetada por Hochreiter & Schmidhuber [16]. A LSTM é adequada para tarefas de previsão de sequências e destaca-se na captura de dependências de longo prazo.

Uma RNN tradicional possui um único estado oculto que é transmitido ao longo do tempo, o que pode dificultar para a rede aprender dependências de longo prazo. As LSTM resolvem esse problema introduzindo uma célula de memória, que pode armazenar informações por um período prolongado. As redes LSTM são capazes de aprender dependências de longo prazo em dados sequenciais, o que as torna adequadas para tarefas como tradução de linguagem, reconhecimento de fala e previsão de séries temporais [17]. As LSTM também podem ser usadas em combinação com outras arquiteturas de rede neuronal, como Redes Neuronais Convolucionais (CNN) para análise de imagens e vídeos.

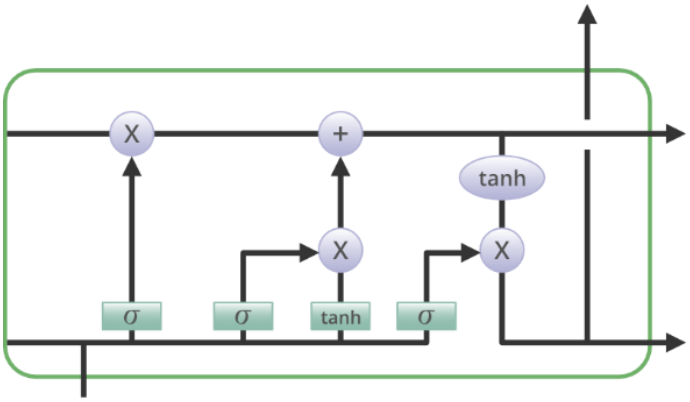

Figura 5 - Diagrama ilustrativo de uma LSTM.

2.1.5.1. Arquitetura de uma LSTM: Estrutura e Funcionamento

Uma LSTM é composta por uma série de células de memória, que são blocos responsáveis por armazenar e processar informações ao longo do tempo. Cada célula LSTM contém três *Gates*: *Input Gate*, *Forget Gate e Output Gate* [18].

Forget Gate: A informação que já não é útil no estado da célula é removida com a porta de esquecimento. Dois *inputs*, x_t (entrada no tempo específico) e h_{t-1} (saída da célula anterior), são alimentados na porta e multiplicados por matrizes de pesos, seguido pela adição de um *bias*. O resultado é passado por uma função de ativação que fornece uma saída binária [18].

Se, para um determinado estado da célula, a saída for 0, a informação é esquecida e, para saída 1, a informação é retida para uso futuro. A equação para a *Forget Gate* é a seguinte:

$$f_t = \sigma(W_f \cdot [h_{t-1},\, x_t] + b_f)$$

onde:

- W_f representa a matriz de pesos associada à *Forget Gate* (ver Figura 6).
- $[h_{t-1},\, x_t]$ denota a concatenação da entrada atual e do estado oculto anterior.
- b_f é o *bias* com a *Forget Gate*.
- σ é a função de ativação sigmoide

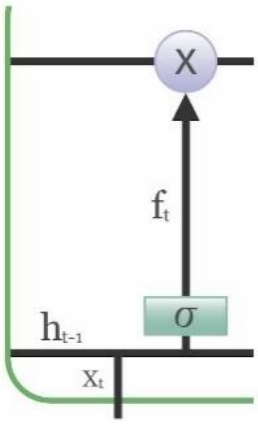

Figura 6 - Diagrama ilustrativo da *Forget Gate.*

Input Gate: A adição de informações úteis ao estado da célula é feita pela *Input Gate* (ver Figura 7). Primeiro, a informação é regulada usando a função sigmoide que filtra os valores a serem lembrados usando as entradas h_{t-1} e x_t. Depois, um vetor é criado usando a função *tanh* que fornece uma saída de -1 a +1, que contém todos os valores possíveis de h_{t-1} e x_t [19]. Por fim, os valores do vetor e os valores regulados são multiplicados para obter as informações úteis [18]. A equação para a *Input Gate* é:

$$i_t = \sigma(W_i \cdot [h_{t-1},\, x_t] + b_i)$$

$$\hat{C}_t = \tanh(W_c \cdot [h_{t-1},\, x_t] + b_c)$$

Multiplicamos o estado anterior por f_t não utilizando a informação que optamos anteriormente por ignorar. A seguir, incluímos $i_t * C_t$. Isto representa os valores candidatos atualizados, ajustados pelo valor que escolhemos para atualizar cada *state value* [18].

$$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t$$

onde:

- $\odot$ denota multiplicação elemento a elemento.
- $tanh$ é a função de ativação *tanh.*

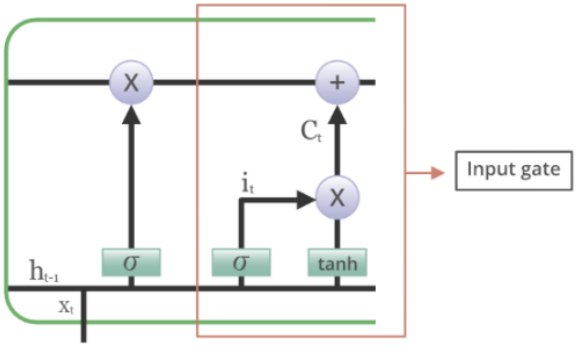

Figura 7 - Diagrama ilustrativo da *Input Gate.*

Output Gate: Quem extrai as informações úteis do estado atual da célula para serem apresentadas como *output* é a *Output Gate* (ver Figura 8). Primeiro, um vetor é gerado aplicando a função $tanh$ na célula. Depois, a informação é regulada através da função sigmoide e filtrada pelos valores a serem lembrados através das entradas h_{t-1} e x_t Por fim, os valores do vetor e os valores regulados são multiplicados para serem enviados como *output* e *input* para a próxima célula [18]. A equação para a *Output Gate* é:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

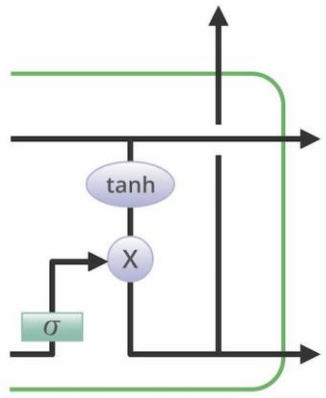

Figura 8 - Diagrama ilustrativo da *Output Gate.*

2.1.5.2. Parâmetros comuns de LSTM

A seguir, estão listados os parâmetros comuns que podem ser configurados ao utilizar camadas LSTM em modelos de redes neuronais:

Units

- Refere-se ao número de unidades LSTM na camada.
- Por exemplo, LSTM(50) define uma camada LSTM com 50 unidades.

Input Shape

- Corresponde a um formato tridimensional: (*batch_size*, *timesteps*, *features*)
  - ***batch_size***: Indica o número de amostras em cada lote de dados.
  - ***timesteps***: Representa o número de passos de tempo em cada sequência de entrada.
  - ***features***: Indica o número de características em cada passo de tempo.

Activation

- Refere-se à função de ativação aplicada às saídas da camada.

Recurrent Activation

- Indica a função de ativação usada para o portão de ativação recorrente.

Utilização de *Bias*

- Indica se a camada utiliza um *bias* nas suas operações.

Kernel Initializer

- Refere-se ao esquema de inicialização para os pesos da camada.

Recurrent Initializer

- Indica o esquema de inicialização para os pesos da conexão recorrente.

Bias Initializer

- Indica o esquema de inicialização para os *bias*.

2.1.6. LSTM VS RNN

As RNN são redes neuronais com conexões cíclicas, permitindo que informações anteriores influenciem as previsões atuais. No entanto, as RNN padrão sofrem de um problema conhecido como *vanishing gradient*, onde as informações relevantes podem ser perdidas ao longo do tempo devido à propagação do gradiente. Isso limita a sua capacidade de reter informações importantes em sequências longas.

Por outro lado, as LSTM foram projetadas para resolver este problema introduzindo unidades de memória especiais chamadas *memory cells*. Estas células têm uma estrutura mais complexa de várias *gates*, que regulam o fluxo de informações na rede. Isso permite que as LSTM aprendam quais informações devem ser lembradas ou esquecidas ao longo do tempo, facilitando a captura de dependências de longo prazo.

Em termos de desempenho, as LSTM tendem a superar as RNN convencionais em tarefas que exigem modelação de sequências mais longas, devido à sua capacidade de reter informações por períodos prolongados. No entanto, as LSTM são mais complexas computacionalmente e podem ser mais difíceis de treinar em conjuntos de dados menores devido à quantidade maior de parâmetros [20] [21].

Tabela 1 - Tabela comparativa entre LSTM e RNN.

Funcionalidade	LSTM	RNN
Memoria	Tem unidade de memória especifica que possibilita a aprendizagem de sequência de dados.	Não tem unidade de memória.
Direccionalidade	Pode ser treinada para processar data em várias direções.	Só pode ser treinada para processar data numa direção.
Treino	Mais difícil devido a complexidade das *gates* e da unidade de memória.	Mais fácil de treinar.
Long-term dependency learning	Sim.	Limitado.
Capacidade de aprender dados sequenciais	Sim.	Sim.

2.1.7. Arquitetura *Bidirectional*

A arquitetura *bidirectional* tem sido amplamente escolhida para desempenhar tarefas que requerem compreensão profundo das direções de uma sequência. Este tipo de arquitetura trata-se de uma extensão das LSTM tradicionais que processam a informação sequencialmente em ambas as direções fazendo com que seja possível captar padrões contextuais mais eficazmente.

2.1.7.1. Estrutura e Funcionamento

As LSTM Bidirecionais (ver Figura 9), ou BiLSTM, consistem em duas redes LSTM combinadas. Uma processa a sequência que entra, na direção *forward*, ou seja, da esquerda para a direita, e outra na direção *backward*, da direita para a esquerda. Este tipo de estrutura permite que a rede capte dependências temporais passadas e futuras melhorando assim a qualidade dos dados que posteriormente irá ser treinado [22].

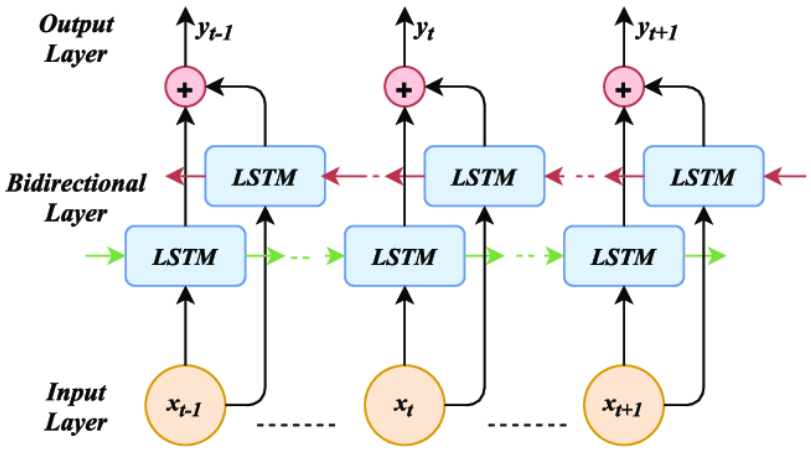

Figura 9 - Diagrama Representativo de uma LSTM *Bidirectional*. [23]

2.1.7.2. Vantagens e Desvantagens

Teoricamente, o uso desta arquitetura influencia, de forma positiva, significativamente na precisão do modelo pela sua capacidade de considerar não só o contexto passado, mas também o contexto futuro de cada ponto na sequência.

Embora as BiLSTM possam ter um impacto significativo no desempenho do modelo, também apresentam algumas desvantagens que devem ser consideradas. O uso deste tipo de arquitetura tem impacto considerável na complexidade computacional por serem muito mais complexas que as LSTM tradicionais. Também tema maior quantidade de parâmetros o que poderá levar ao aumento do consumo de memória. Poderá existir um risco de *Overfitting*,

especialmente em *datasets* pequenos, devido ao aumento de parâmetros que deve ser controlado com os respetivos métodos de contenção [22].

2.1.8. Redes Neuronais Convolucionais

As *Convolutional Neuronal Network* (CNN) são um tipo de redes neuronais que foram propostas pelo pesquisador francês Yann Lecun [24] que se destacam pela eficácia obtida na análise de dados que estejam no formato *grade.* Tal como o nome indica, *Convolutional*, significa uma função matemática que deriva da integração de outras duas funções completamente distintas. A arquitetura das CNN (ver Figura 10) deriva das *Artificial Neuronal Networks* (ANN). Estes tipos de redes neuronais costumam trabalhar sobre imagens e sinais de áudio, no entanto, também existem outras vertentes que aplicam este conceito a dados de texto, séries temporais de dados e sequências de dados.

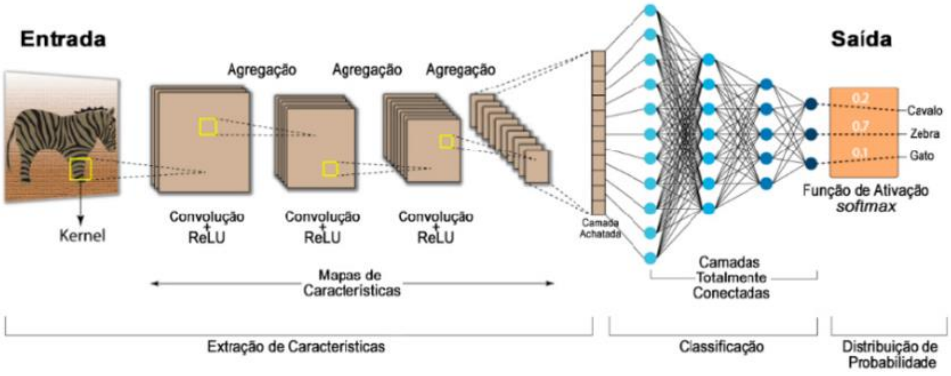

Figura 10 - Estágios e Camadas de uma Rede Neuronal Convolucional. [25]

2.1.9. Redes Convolucionais Unidimensionais

Redes convulsionais unidimensionais são uma variação das tradicionais que trabalham em duas dimensões. Neste formato, existe uma adaptação para este tipo de rede lidar com dados sequenciais em vez de operar com dados bidimensionais fazendo com que possa processar sequências unidimensionais tais como texto e sinais de áudio [26].

2.1.9.1. Estrutura e Funcionamento

Uma CNN pode ser dividida em três etapas principais (ver Figura 11): *Convolutional* com função de ativação, *Pooling* e *Fully-connected.*

Na primeira camada *Convolucional*, é processada a operação que realiza a convulsão dos dados de entrada. Será aplicado um *kernel*, que se trata de um filtro, sobre o *array* de dados de entrada, que irá calcular o valor da convolução para depois utilizá-lo como o valor

de uma célula do *array* de saída. Durante este processo, o *kernel* é deslocado por uma janela deslizante predefinida pelo *array* que deu entrada. Os parâmetros utilizados pelos *kernels* são definidos durante o processo de treino exceto o parâmetro que controla a dimensão do *kernel*, por se tratar de um híper parâmetro definido previamente ao processo de treino.

A segunda camada, *Pooling*, é bastante semelhante à camada convolucional tendo como principal objetivo diminuir o tamanho do espaço ocupado pelas variáveis convulsionais. Uma técnica bastante utilizada nesta etapa é o *max-pooling* que consiste em reduzir subpartes dos dados originais pelo maior valor encontrado nessas sub-regiões. Como estamos a trabalhar com sequências unidimensionais e o *polling* precisa duas dimensões, podemos fazer um *reshape* do *input shape* para solucionar este problema.

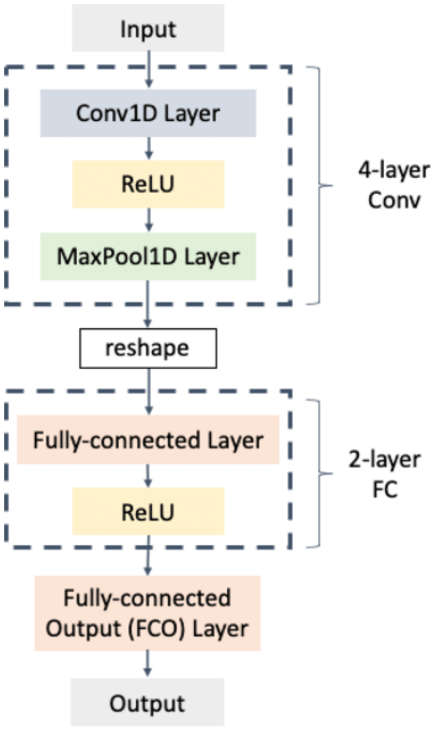

Figura 11 - Diagrama ilustrativo de uma implementação CNN 1D usando *Max Pooling*. [27]

A última etapa da rede é a *fully-connected* que irá fornecer a probabilidade, no nosso problema, de a condução ser classificada como agressiva ou não agressiva [28].

2.1.9.2. Vantagens e Desvantagens

As CNN, mais concretamente as Conv 1D, são menos honrosas computacionalmente que as LSTM tradicionais, sendo mais rápidas de treinar e executar. Este tipo de arquitetura é extremamente eficaz na extração de padrões importantes em subsequências, o que terá influência no resultado obtido. Como as CNN permitem-nos aplicar camadas de *pooling*, a

dimensão dos dados é reduzida, o que faz com que a complexidade do modelo também acabe por ser reduzida, evitando assim um potencial *overfitting* da rede. Mas a principal vantagem que se destaca é a capacidade desta arquitetura se poder combinar com outras, como LSTM, para aprimorar os resultados, fazendo com que a escalabilidade seja possível [29].

Contudo, as Conv 1D também possuem desvantagens que devem ser tidas em conta na tomada de decisão da escolha do tipo de arquitetura a utilizar. Entre estas desvantagens está a limitação em capturar dependências de longo prazo, pois são projetadas para capturar padrões locais. A tarefa da escolha dos melhores hiperparâmetros também pode ser um processo complexo e requerer tomar uma abordagem tentativa erro. A eficácia das Conv 1D depende principalmente da estrutura de dados de entrada, já que em casos onde os dados não possuam um padrão temporal ou sequências, as Conv 1D não irão produzir resultados precisos [29].

2.1.10. Análise de Soluções

A Tabela 2 apresenta um resumo comparativo das diferentes arquiteturas.

Tabela 2 - Tabela comparativa entre as diferentes arquiteturas.

Característica	LSTM	RNN	Conv1D	Bi-LSTM
Memória	Sim	Não	Não	Sim
Direcionalidade	Unidirecional /*Bidirectional*	Unidirecional	Unidirecional	*Bidirectional*
Treino	Difícil	Fácil	Fácil	Difícil
Long-term dependency learning	Sim	Limitado	Não	Sim
Capacidade de aprender dados sequenciais	Sim	Sim	Limitado	Sim
Captura de dependências temporais	Sim	Sim	Não	Sim

Complexidade computacional	Alta	Média	Baixa	Alta
Risco de *Overfitting*	Médio	Baixo	Médio	Alto
Capacidade de Processamento de Dados Grandes	Alta	Média	Alta	Alta
Integração com outras tecnologias	Sim	Sim	Sim	Sim

2.2.Tecnologias e Ferramentas

A área de classificação de condução e análise comportamental ao volante tem evoluído significativamente nos últimos anos, impulsionada pelo avanço em tecnologias de IA e ML As principais tecnologias e ferramentas atualmente utilizadas neste campo incluem:

- Redes Neuronais
- Sensores de medição inercial
- GPS
- Técnicas de Pré-processamento de dados
- Ambientes de desenvolvimento e bibliotecas de IA
- *Big Data*
- Computação em Nuvem

Neste capítulo são apresentadas as tecnologias e ferramentas utilizadas ao longo do desenvolvimento do projeto.

2.2.1. Linguagem *Python*

O *Python* é uma linguagem de programação amplamente usada em aplicações da *Web*, desenvolvimento de *software*, ciência de dados e ML. Os programadores usam o *Python* porque é eficiente e fácil de aprender e pode ser executada em muitas plataformas diferentes. O *software Python* pode ser instalado gratuitamente e integra-se bem com todos os tipos de sistemas agilizando assim o desenvolvimento [30]. Resumidamente, *Python* é um conjunto de instruções que fornecemos sob a forma de um programa ao nosso computador para realizar qualquer tarefa específica. É uma linguagem de programação que possui propriedades como interpretação, é orientada a objetos e também de alto nível. Devido à sua

sintaxe amigável para principiantes, tornou-se uma escolha clara para o desenvolvimento deste projeto. O principal foco por detrás da sua criação é facilitar a leitura e compreensão dos programadores, reduzindo também as linhas de código [31].

2.2.2. Bibliotecas

A seguir são apresentadas as bibliotecas em *Python* utilizadas no desenvolvimento do projeto.

2.2.2.1. os

O os é uma biblioteca padrão do *Python* que fornece funcionalidades para interagir com o sistema operativo, como manipulação de arquivos e diretorias. É essencial para operações que envolvem o sistema de arquivos e outros aspetos dependentes sistemas [32].

2.2.2.2. math

A math é uma biblioteca padrão que oferece funções matemáticas básicas, incluindo operações trigonométricas, logarítmicas e aritméticas. É amplamente utilizada para cálculos matemáticos em *scripts Python* [33].

2.2.2.3. NumPy

O *NumPy* é uma biblioteca fundamental para computação científica em *Python*. Proporciona suporte para *arrays* multidimensionais e funções matemáticas de alto nível para operarem nestes *arrays*, sendo essencial para operações numéricas eficientes [34].

2.2.2.4. Pandas

O *Pandas* é uma biblioteca essencial para analisar dados, permitindo manipulação e análise de estruturas de dados como *DataFrames*. Oferece as ferramentas precisas para a leitura, transformação e agregação de dados, facilitando o trabalho com grandes conjuntos de dados [35].

2.2.2.5. Matplotlib

O *Matplotlib* é uma biblioteca de visualização de dados que permite a criação de gráficos estáticos em *Python*. É amplamente utilizada para gerar visualizações detalhadas e personalizadas de dados [36].

2.2.2.6. Seaborn

O *Seaborn* é uma biblioteca baseada no *Matplotlib* que fornece uma interface de alto nível para a criação de gráficos estatísticos atraentes e informativos. Facilita a visualização de dados complexos de maneira percetível e estética [37].

2.2.2.7. TensorFlow

O *TensorFlow* é uma biblioteca de *software* de código aberto desenvolvida pelo Google, utilizada para construir e treinar modelos de ML e DL. É reconhecida pela flexibilidade e escalabilidade, permitindo a criação de uma variedade de modelos complexos numa ampla gama de plataformas, desde dispositivos móveis até grandes *clusters* de servidores [38].

2.2.2.8. Keras

O *Keras* é uma biblioteca de código aberto em *Python*, especialmente dedicada ao DL. Destaca-se pela sua interface simplificada, que facilita a criação e treino de redes neuronais artificiais. É compatível com o *TensorFlow* e oferece flexibilidade para desenvolver uma variedade de arquiteturas de redes neuronais [39].

- ***keras.models.Sequential, load_model***: Classes e funções para criar e carregar modelos sequenciais.
- ***keras.layers***: Inclui várias camadas usadas em redes neuronais, como LSTM, *Dense*, *Dropout*, *Conv1D*, *Flatten*, Bidirectional, *BatchNormalization*.
- ***keras.callbacks***: Contém ferramentas como *EarlyStopping*, *ReduceLROnPlateau* e *ModelCheckpoint* para otimizar e guardar modelos durante o treino.

2.2.2.9. scikit-*learn*

O *Scikit-learn* é uma biblioteca para ML em *Python* que oferece ferramentas eficientes para a construção e avaliação de dados [40].

- ***metrics***: Inclui várias funções para avaliar a performance de modelos, como *precision_score*, *f1_score*, *recall_score*, *accuracy_score*, *hamming_loss*, *jaccard_score*, *coverage_error*, *label_ranking_loss*.
- ***preprocessing***: Fornece métodos para pré-processamento de dados, como *LabelEncoder*.

- ***model_selection***: Inclui funções para dividir os dados em conjuntos de treino e teste, como *train_test_split*.

2.2.2.10. Folium

O *Folium* é uma biblioteca para criar mapas interativos utilizando *Leaflet.js*, permitindo adicionar marcadores, camadas e plugins de maneira simples [41].

- ***folium.plugins.MarkerCluster, FastMarkerCluster***: Plugins para agrupar marcadores em *clusters*, melhorando a visualização de mapas com muitos pontos de dados.
- ***folium.plugins***: Outros plugins úteis para funcionalidades adicionais em mapas interativos.

2.2.3. Plataformas

Neste subcapítulo são apresentadas as tecnologias utilizadas no desenvolvimento do projeto.

2.2.3.1. GitHub

O *GitHub* é uma plataforma *online* muito utilizada para controlo de versões de projetos. Foi nesta plataforma que foram guardadas as várias versões do projeto num repositório que ambos os programadores tinham acesso. Desta forma, foi possível ter sempre o código atualizado e sincronizado, facilitando assim o processo de desenvolvimento.

2.2.3.2. Lucidchart

O *Lucidchart*, um aplicativo de diagramação inteligente que corre na *cloud*, é um componente central da *Suíte* de colaboração visual da *Lucid Software*. Esta solução intuitiva de *cloud* oferece a possibilidade de colaborar em tempo real para criar fluxogramas, *mockups*, diagramas UML (*Unified Modeling Language*) e muito mais [42]. Foi utilizado o *Lucidchart* para criar os diagramas representativos das arquiteturas que desenvolvemos.

2.2.3.3. Microsoft Teams

O *Microsoft Teams* é uma aplicação de colaboração criada para trabalho híbrido para que uma equipa se mantenha informada, organizada e ligada, tudo num único local. Nesta ferramenta é possível criar uma equipa e canais para reunir pessoas e trabalhar em espaços

focados com conversações e ficheiros [43]. Foi através desta plataforma que foram realizadas algumas das nossas reuniões com os nossos orientadores.

2.2.3.4. Visual Studio Code

O *Visual Studio Code* é um editor de código com suporte para operações de desenvolvimento como depuração, execução de tarefas e controlo de versões. O seu objetivo é fornecer apenas as ferramentas que um programador precisa para um ciclo rápido de construção de código e depuração e deixa fluxos de trabalho mais complexos para IDE (*Integrated Development Environment*) com recursos mais completos, como o *Visual Studio IDE* [44]. Foi neste editor que desenvolvemos o código para a nossa solução.

2.3. Trabalhos Relacionados

Nesta secção serão apresentados trabalhos relacionados com o projeto desenvolvido e que serviram de suporte ao estudo do tema.

2.3.1. Driving Behavior Classification Based on Sensor Data Fusion Using LSTM Recurrent Neuronal Networks

Este artigo [45] visa a classificação de comportamentos de condução utilizando uma metodologia baseada na fusão de dados e sensores, mais concretamente utilizando RNN do tipo LSTM. Esta abordagem tem como finalidade identificar três classes distintas de condução: normal, agressiva e sonolenta.

2.3.1.1. Metodologia Proposta

O processo de classificação da condução foi tratado como um problema de classificação de series temporais. Para cada instante *T* de uma viagem, uma janela sequencia *S* de dados de sensores foi classificada como sendo uma das três categorias de condução.

2.3.1.2. Arquitetura do Modelo

O modelo que foi proposto denomina-se por *Stacked*-LSTM e consiste em duas camadas de células de memória, cada uma com 100 neurónios ocultos. Os dados de entradas incluem nove vetores com características vindas dos sensores internos do *smartphone* usado para os recolher. Os sensores utilizados e as respetivas características captadas foram os seguintes:

- Sensores de Inércia:
    - Aceleração ao longo dos eixos X, Y e Z.
    - Ângulos de rotação e inclinação.
- Sensor GPS:
    - Velocidade do veículo.
- Sensor da Câmera do *Smartphone*:
    - Distância para o veículo da frente.
    - Número de veículos detetados.

2.3.1.3. Processo de Treino do Modelo

O processo de treino do modelo foi tratado como um problema de minimização da função de perda *Softmax*. O algoritmo de otimização utilizado foi o *Adam* com uma taxa de aprendizagem correspondente a 0,0025 juntamente com uma regularização L2 para prevenir *overfitting*.

2.3.1.4. *Dataset* Utilizado

O conjunto de dados utilizado para o treino do modelo foi o "UAH-*DriveSet*". Este *dataset* contém dados de várias viagens que foram capturadas através de um *smartphone*, que incluem medições dos sensores: acelerómetro, giroscópio, GPS e *frames* da câmera do *smartphone* (ver Figura 12).

Figura 12 - *Software* utilizado para capturar os dados do UAH-*driveset*. [45]

2.3.1.5. Preparação e Pré-processamento de Dados

Os dados retirados do *dataset* passaram por várias etapas de preparação e pré-processamento antes de seguirem para o modelo. Nesta etapa foi incluída a sincronização dos diferentes tipos de dados. Também foi feita a normalização das medições feitas pelos sensores e a respetiva segmentação das sequências adequadas para o modelo LSTM (ver Figura 13).

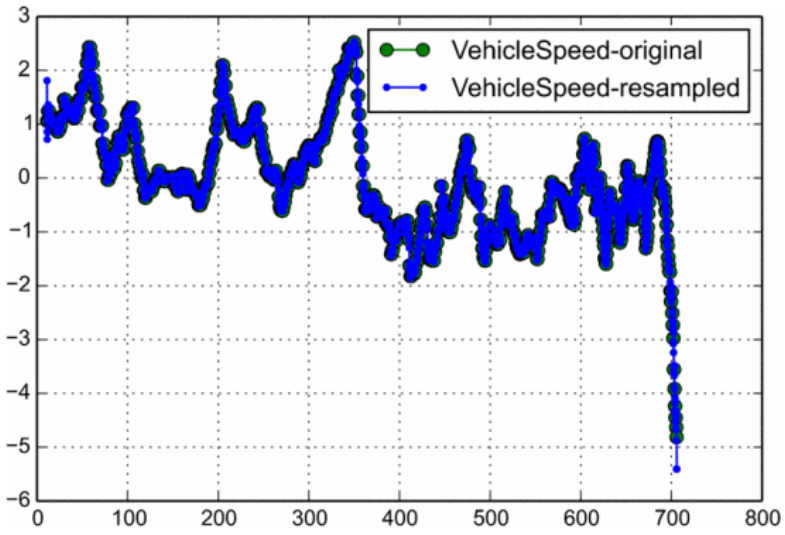

Figura 13 - Gráfico ilustrativo do *resampling* dos dados. [45]

2.3.1.6. Resultados Obtidos

Esta implementação obteve resultados bastante concisos por ter conseguido alcançar com elevada precisão a classificação dos comportamentos de condução. A avaliação foi feita utilizando um conjunto de teste separado, e os resultados foram comparados com abordagens também já existentes, demonstrando assim a eficácia do modelo (ver Figura 14).

		DriveSafe [6]			**proposed Stacked-LSTM**		
Road Type	**True State**	Normal	Aggressive	Drowsy	Normal	Aggressive	Drowsy
Motorway	Normal	7.3	1.5	1.2	**8.4**	0.9	0.7
	Aggressive	3.5	5.2	1.3	1.1	**8.0**	0.9
	Drowsy	3.0	1.6	5.4	0.6	0.2	**9.2**
Secondary	Normal	7.7	1.1	1.2	**9.5**	0.1	0.4
	Aggressive	0.6	7.3	2.1	0.4	**9.6**	0.0
	Drowsy	1.9	1.9	6.2	0.7	0.0	**9.3**

Figura 14 - Resultados da matriz de confusão da implementação proposta em comparação com a classificação de comportamento de condução descrita pelo UAH-*DriveSet*. [45]

2.3.2. Driving Behavior Classification Based on Oversampled Signals of *Smartphone* Embedded Sensors Using an Optimized Stacked-LSTM Neuronal Networks

2.3.2.1. Metodologia Proposta

A metodologia proposta neste artigo [46] é baseada na construção de um modelo LSTM otimizado para análise do tipo de condução, selecionando a melhor configuração e parâmetros em cada fase de desenvolvimento. Começa com a formulação do problema de classificação do comportamento do condutor e segue com o pré-processamento dos dados pertencentes ao dataset. Depois, uma arquitetura LSTM é desenvolvida para resolver o classificar a condução em duas tarefas de classificação: três classes (normal, sonolento e agressivo) e binária (agressivo e não agressivo).

2.3.2.2. Arquitetura do Modelo

O modelo que foi proposto também foi uma *Stacked* LSTM e consiste em duas camadas de células de memória, cada uma com 120 neurónios ocultos. As camadas utilizam a função de ativação *ReLU* e aplicam regularização L2 para evitar *overfitting*. A última camada do modelo é uma camada *Softmax* que recebe os vetores de características da segunda camada LSTM e produz uma pontuação de classificação para as três classes de comportamento de direção: normal, sonolento ou agressivo. Ao contrário de trabalhos anteriores que usaram uma janela de 64 vetores com nove características, este modelo usa uma janela de 16 vetores com 13 características.

2.3.2.3. Processo de Treino do Modelo

No treino dos modelos, os dados foram divididos em conjuntos de treino e teste. O modelo é treinado usando a função de *loss* e o optimizador *Adam*, e os melhores parâmetros são selecionados com base no desempenho dos modelos sobre os dados de validação. Este processo também envolveu o uso de técnicas de *oversampling* para balancear o conjunto de dados e desta forma generalizar o modelo.

2.3.2.4. Dataset Utilizado

O conjunto de dados utilizado para o treino do modelo também foi o "UAH-*DriveSet*".

2.3.2.5. Preparação e Pré-processamento de Dados

O pré-processamento dos dados envolveu várias etapas para preparar os dados que foram lidos dos sensores antes de estes alimentarem o modelo. Foi feita uma normalização, segmentação e aplicação de técnicas de *oversampling* para balancear das classes presentes no conjunto de dados. Estas etapas foram essenciais para melhorar a *accuracy* e a robustez do modelo.

2.3.2.6. Resultados Obtidos

Os resultados obtidos (ver Figura 15) nesta implementação mostram que o modelo LSTM alcançou uma alta precisão na classificação dos comportamentos de condução. As métricas de desempenho, como precisão, *recall* e *F1-score*, indicam que o modelo é extremamente eficaz em destingir e classificar comportamentos de condução, tanto na versão de classificação de três classes como na classificação binária.

Classification Type	Model	Accuracy	Precision	Recall	F1-score
Three-class classification	3-CCM based on FCPT-3C	99.47%	99.50%	99.48%	99.49%
	SL-LSTM based on FCPT-3C	97.47%	97.59%	97.44%	97.51%
Binary classification	2-CCM based on FCPT-2C	99.30%	99.26%	99.33%	99.34%
	SL-LSTM based on FCPT-2C	98.02%	98.02%	97.75%	97.88%

Figura 15 - Tabela comparativa dos resultados obtidos. [46]

2.3.3. Exploiting the use of recurrent neuronal networks for driver behavior profiling

O artigo "*Exploiting the Use of Recurrent Neuronal Networks for Driver Behavior Profiling*" [47] explora o uso de RNN para a classificação de eventos de condução, utilizando dados de acelerómetros de *smartphones*. O estudo propõe três arquiteturas distintas de RNN, incluindo LSTM, Gated Recurrent Unit (GRU), e *Simple* RNN. Cada uma dessas arquiteturas é aplicada a uma tarefa de classificação de eventos de condução, como aceleração e travagem agressiva, utilizando um dataset gerado a partir de dados coletados em smartphones.

O estudo conduzido pelo estudo demonstrou que a arquitetura GRU apresentou os melhores resultados, alcançando uma precisão superior a 95% na deteção de eventos agressivos. Em contrapartida, a Simple RNN apresentou a menor performance, com resultados de precisão abaixo de 70%. O LSTM, apesar de ter desempenho variável dependendo do número de neurónios na camada oculta, também obteve resultados comparáveis aos do GRU, especialmente com 10 neurónios na camada oculta.

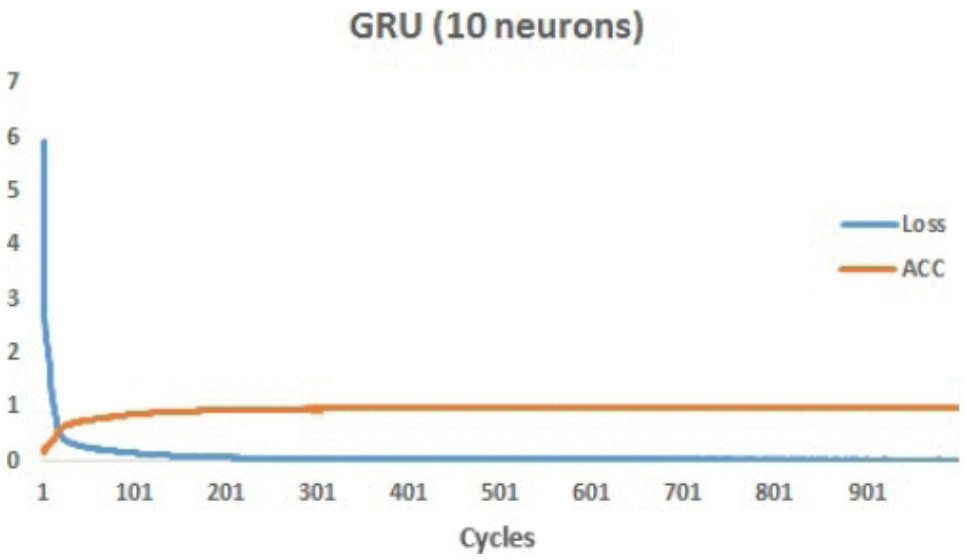

Figura 16 - Resultados GRU – Artigo [47]

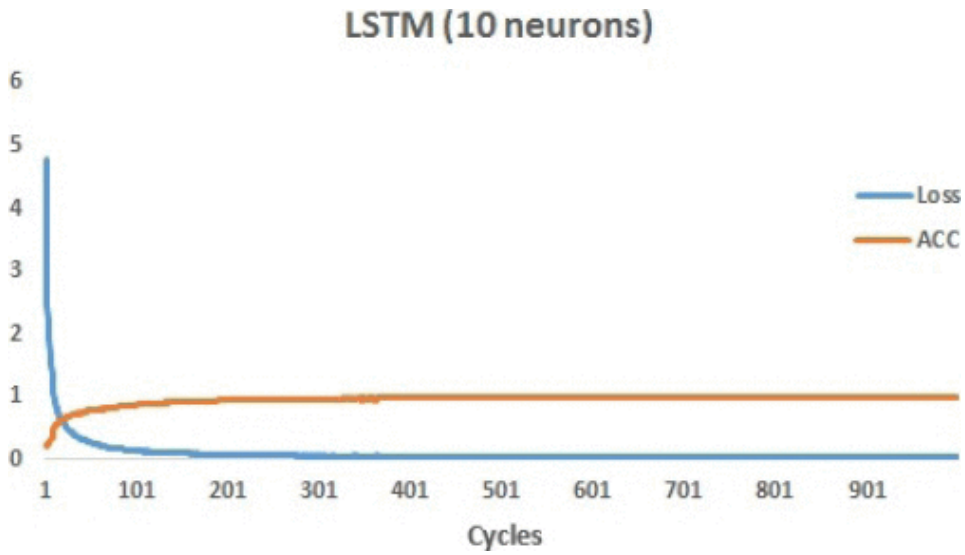

Figura 17 - Resultados GRU – Artigo [47]

Os autores sugerem que a vantagem do GRU está na sua habilidade de capturar dependências de longo prazo de maneira mais eficaz do que o Simple RNN. Essa vantagem pode ser atribuída ao uso de unidades de controle, como o "reset gate" e o "update gate", que permitem ao GRU ajustar dinamicamente a quantidade de memória necessária para cada passo temporal.

3. Análise

Neste capítulo iremos detalhar a analise feita aos sensores utilizados dos *smartphones*, da aplicação de recolha de dados e dos *datasets* usados que serviram de suporte da produção da solução final.

3.1. Análise de requisitos

Nesta secção, iremos abordar o planeamento do nosso projeto, dando destaque inicial aos requisitos funcionais e não funcionais que determinam as características e padrões de desempenho essenciais para a criação da nossa solução.

Os requisitos funcionais especificam as características essenciais que o sistema deve ter, como a aquisição e processamento de dados, o treino do modelo, a classificação, a interface e o relatório com os resultados para posterior análise.

Os requisitos não funcionais, por sua vez, determinam os atributos de qualidade que o sistema precisa de possuir, tais como desempenho, escalabilidade, confiabilidade e manutenibilidade. Estes requisitos são essenciais para garantir que o sistema funcione de forma sólida e precisa.

Ao estabelecer claramente estes pré-requisitos, criamos uma base robusta para a construção e execução da solução, assegurando que todos os recursos e padrões de desempenho sejam cumpridos de maneira organizada e metódica.

3.1.1. Requisitos Funcionais

3.1.1.1. Aquisição de Dados

A solução deve permitir a integração com dados no formato JSON e CSV.

3.1.1.2. Processamento de Dados

A solução deve ser capaz de processar os dados de entrada, incluindo normalização, limpeza e transformação dos dados para se adequarem ao modelo desenvolvido.

3.1.1.3. Treino do Modelo

A solução precisa de usar algoritmos para construir modelos de classificação baseados em LSTM. Deve ainda ser possível fazer ajustes nos hiperparâmetros do modelo para melhorar o seu desempenho.

3.1.1.4. Classificação

A solução precisa de ter a capacidade de classificar o comportamento do condutor com elevado grau de precisão e de confiança.

3.1.1.5. Relatórios e Análises

A solução deve criar relatórios minuciosos sobre o desempenho da classificação, incluídos gráficos e outros métodos de visualização de resultados. É necessário oferecer análises percetíveis e visualizações intuitivas dos resultados obtidos.

3.1.2. Requisitos Não Funcionais

3.1.2.1. Desempenho

Para garantir classificações rápidas, é necessário que o sistema seja eficiente e consiga produzir resultados fiáveis no menor espaço de tempo.

3.1.2.2. Escalabilidade

Deve possibilitar a inclusão de novos algoritmos ou algoritmos de classificação no futuro sem precisar de mudanças substanciais.

3.1.2.3. Manutenibilidade

A estrutura deve ser planeada de maneira modular para facilitar a manutenção e atualização dos elementos, que compõem o código, separadamente.

3.1.3. Arquitetura da Solução

A arquitetura que desenvolvemos para a nossa solução pode ser descrita visualmente na Figura 18.

A nossa proposta de solução baseia-se numa arquitetura modelar, projetada para receber um *dataset* com dados de sensores de um smartphone, processá-los e gerar

resultados. A arquitetura combina várias etapas que colaboram entre si, garantindo uma operação o mais eficiente possível.

- Componentes:
    - Recessão do *dataset*: Os dados são recebidos pela solução podendo esta suportar formato XML ou JSON.
    - Preprocessamento: Esta etapa envolve a limpeza, transformação e preparação dos dados provenientes do *dataset* para que possam ser usados pelo modelo LSTM. É essencial para garantir que os dados estejam no melhor formato possível, pois podem ter impacto direto nos resultados obtidos.
    - Modelo: O núcleo da nossa arquitetura é o modelo, pois é ele que ira aprender a classificar os comportamentos de condução. Ele recebe os dados pré-processados e realiza análises complexas para identificar padrões e prever resultados futuros.
    - Resultados: Após as tarefas realizadas pelo modelo, a rede é submetida a testes que irão avaliar o desempenho da mesma. Os resultados são apresentados de forma compreensível para os utilizadores, recorrendo a gráficos e tabelas para uma melhor visualização.

- Ligações:
    - Aplicação ↔ Preprocessamento: A aplicação envia os dados captados pelos sensores para a etapa de preprocessamento, onde são preparados para o modelo.

    - Preprocessamento ↔ Modelo: Os dados tratados servirão de alimento para a rede neuronal, onde são processados para gerar previsões.

- Modelo ↔ Resultados: Os resultados das análises realizadas pelo modelo são organizados e apresentados de maneira acessível aos utilizadores finais.

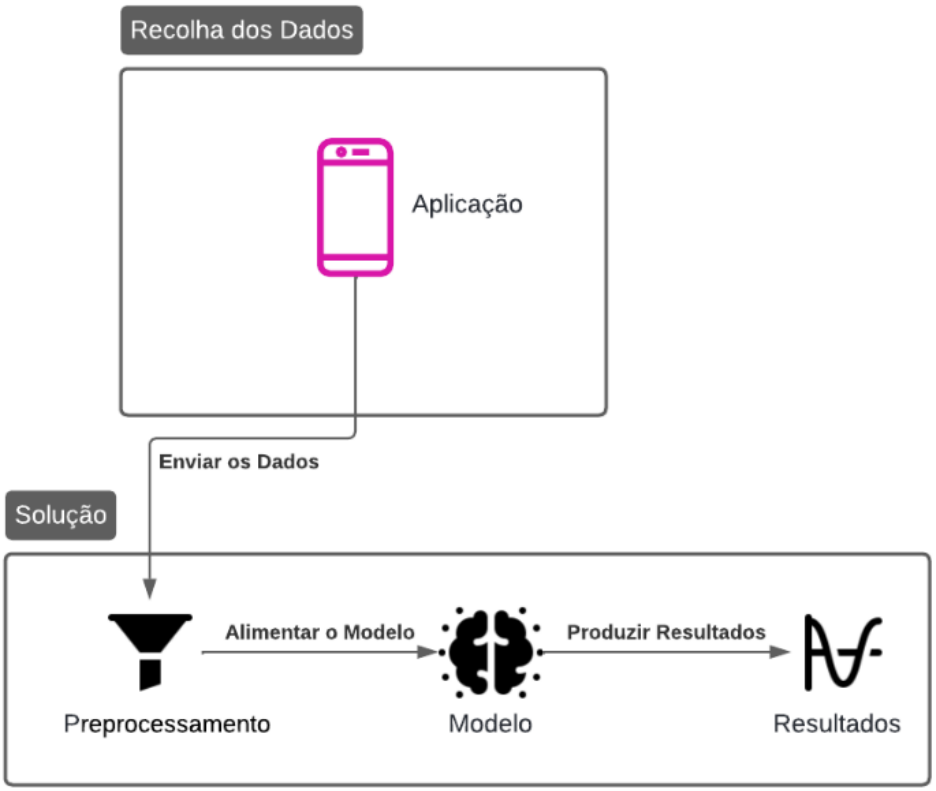

Figura 18 - Arquitetura do protótipo.

3.2. Análise da Aplicação de Recolha dos Dados

Para recolher os dados do IPL-*Dataset* foi utilizada uma aplicação que foi desenvolvida como parte do Projeto Informático "Aplicação para Rastreio de Viaturas" [1] em que o principal objetivo é de rastrear comportamentos de condução através de dados capturados por sensores embutidos em dispositivos móveis como giroscópio, acelerómetro e GPS. Entre estes comportamentos, os mais importantes para o nosso projeto são: travagens bruscas, guinadas e acelerações repentinas.

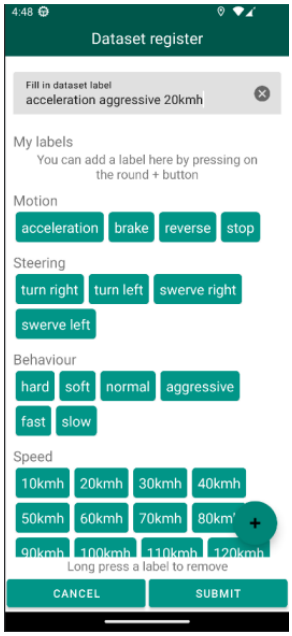

Figura 19 - Ecrã de criação de cenário.

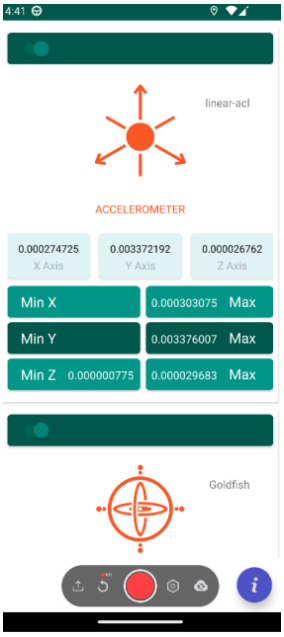

Figura 20 - Ecrã com os dados dos sensores

3.3. Análise de *Datasets*

Para este projeto optamos por usar dois *datasets*, o IPL-*Dataset* e o UAH-*DRIVESET*. Enquanto o IPL-*Dataset* foi fornecido no começo do desenvolvimento, o UAH-*DRIVESET* foi descoberto durante a nossa pesquisa e optamos por usá-lo por constatarmos que é um dataset amplamente usado em trabalhos relacionado com este por ser bastante equilibrado no que toca à informação que possui.

Os *datasets* usados contêm valores de sensores durante uma viagem de carro, do ponto A ao ponto B. Estes valores são utilizados para identificar diferentes tipos de manobras e comportamentos durante a condução.

3.3.1. IPL Dataset

O IPL-Dataset foi criado através da utilização da Aplicação para Rastreio de Viaturas [1] pelos estudantes que desenvolveram o projeto. Os dados no dataset IPL-*Dataset* possui informações capturadas por sensores em vários tipos de gravações como viagens completas e curtas gravações de manobras ou cenários comuns de condução. Esses cenários estão descritos na Tabela 3.

Tabela 3 - Descrição dos cenários de condução do IPL-Dataset.

Cenário	**Descrição**
Bigger Trips	O veículo simula a condução ao longo de uma viagem prolongada.
Acceleration	O veículo parte do estado de repouso e acelera progressivamente até atingir uma velocidade específica.
Brake	O veículo reduz a velocidade até parar, usando o pedal de travão.
Constant Speed	O veículo mantém velocidades constantes ao longo de um percurso.
Friction Straight Line	O veículo reduz a velocidade até parar usando apenas o atrito, sem acionar o pedal de travão.
Intersection (ver Figura 21)	O veículo simula o comportamento numa interseção.
Roundabout	O veículo simula o comportamento numa rotunda.

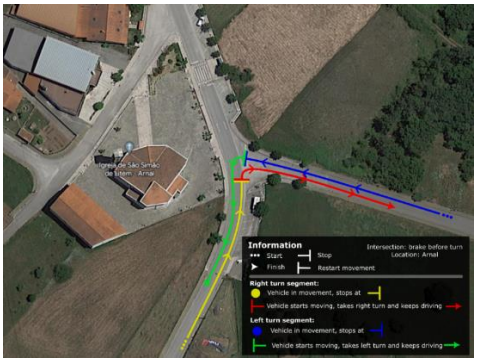

Figura 21 - Cenário de condução – Interseção.

3.3.1.1. Estrutura dos dados

Nesta subsecção, detalhamos a estrutura dos dados recolhidos, que inclui informações provenientes de três fontes principais: acelerómetro, giroscópio e GPS. Cada uma destas fontes fornece diferentes tipos de informação essenciais para a análise e compreensão do comportamento do veículo em diferentes cenários, como descrito na Tabela 4.

Tabela 4 - Descrição dos dados do IPL-Dataset.

Fonte	Taxa de amostragem	Informação do Sensor	Unidade
Acelerómetro	4 Hz	Aceleração em X	Gs
		Aceleração em Y	Gs
		Aceleração em Z	Gs
Giroscópio	4 Hz	Velocidade Angular em X	º/s
		Velocidade Angular em Y	º/s
		Velocidade Angular em Z	º/s
GPS	1 Hz	Latitude	º
		Longitude	º
		Velocidade Instantânea	Km/h

3.3.2. UAH-*DRIVESET*

O UAH-*DriveSet* é uma coleção pública de dados captados pela *DriveSafe*, uma aplicação de monitorização de condução, que foi utilizada por vários condutores em diferentes ambientes. Este conjunto de dados tenta facilita o progresso no campo da análise dos comportamentos de condução, fornecendo uma grande quantidade de variáveis que foram capturadas e processadas por todos os sensores e capacidades de um *smartphone* durante os testes de condução independentes realizados. A aplicação foi usada por 6 condutores em veículos diferentes, realizando 3 tipos de comportamentos diferentes (normal, sonolento e agressivo) e em dois tipos de estradas (autoestrada e estrada nacional), resultando em mais de 500 minutos de condução natural com os dados brutos associados e dados adicionais.

Figura 22 - Exemplo de carro de testes. (UAH-*Dataset*)

3.3.2.1. Estrutura dos Dados utilizados

Tabela 5 - Descrição dos dados do UAH-Driveset.

Fonte	Taxa de amostragem	Informação do Sensor	Unidade
Acelerómetro	10 Hz	Aceleração em X	Gs
		Aceleração em Y	Gs
		Aceleração em Z	Gs
Giroscópio	10 Hz	Velocidade Angular em X	º/s
		Velocidade Angular em Y	º/s
		Velocidade Angular em Z	º/s
GPS	1 Hz	Latitude	º
		Longitude	º
		Velocidade Instantânea	Km/h

4. Soluções Propostas

Este capítulo apresenta as soluções propostas após a análise detalhada do estado da arte e a identificação dos requisitos específicos para abordar a tarefa de classificação de condução, através de dados de sensores em dispositivos moveis.

Com base nos conhecimentos adquiridos e nas necessidades identificadas, foi proposto o uso de várias arquiteturas de redes neuronais LSTM, cada uma com as suas vantagens específicas para lidar com dados temporais. As arquiteturas propostas são: *Stacked* LSTM, *Bidirectional* LSTM e *Convolutional* LSTM. As arquiteturas seguem os mesmos processos descritos no Capítulo 9, mudando apenas a forma como modelo é construído. A seguir, detalharemos cada uma dessas arquiteturas.

4.1. *Stacked* LSTM

4.1.1. Introdução a *Stacked* LSTM

Uma arquitetura Stacked LSTM consiste em várias camadas de LSTM sobrepostas uma sobre a outra. Essa estrutura em camadas permite que o modelo aprenda representações hierárquicas dos dados sequenciais, onde cada camada pode capturar diferentes níveis de abstração temporal.

4.1.2. Estrutura da Stacked LSTM

1) Entrada: Dados preprocessados
2) Camadas LSTM:
    a. Primeira Camada: Recebe a sequência de entrada e aprende as representações iniciais dos dados temporais.
    b. Camadas intermédias: Cada camada subsequente recebe a saída da camada anterior, permitindo modelar padrões mais complexos e abstrações de longo prazo.
    c. Última Camada LSTM: Produz a representação final da sequência temporal, integrando informações de todas as camadas anteriores.
3) Camadas Densa (*Fully connected Layer*): Ligar a saída da última camada LSTM a uma ou mais camadas densas.
4) Camada de saída: Camada densa cujo número de neurónios corresponde ao número de classes que o modelo precisa de prever.

4.2. *Bidirectional* LSTM

4.2.1. Introdução a *Bidirectional* LSTM

A arquitetura *Bidirectional* LSTM é uma variação das redes LSTM que processa a sequência de dados em duas direções: do passado para o futuro e do futuro para o passado. Essa abordagem permite que o modelo capture melhor as dependências temporais nos dados, utilizando informações contextuais de ambas as direções.

4.2.2. Estrutura da *Bidirectional* LSTM

1) Entrada: Dados preprocessados.
2) Camadas *Bidirectional* LSTM:
    a. Camada Direcional Frontal: Processa a sequência de entrada na ordem original, aprendendo representações temporais avançadas.
    b. Camada Direcional Reversa: Processa a sequência na ordem inversa, permitindo que o modelo capture padrões que podem depender de informações futuras.
    c. Concatenação de Saídas: As saídas das duas direções são concatenadas para formar uma representação mais rica e contextualizada da sequência.
3) Camadas Densa (*Fully connected Layer*): Conecta a saída concatenada a uma ou mais camadas densas.
4) Camada de saída: Camada densa cujo número de neurónios corresponde ao número de classes que o modelo precisa prever.

4.3. *Convolutional* LSTM

4.3.1. Introdução a *Convolutional* LSTM

A arquitetura *Convolutional* LSTM combina as capacidades das camadas convolucionais e LSTM para lidar com dados sequenciais com características espaciais. Essa arquitetura é especialmente útil quando os dados temporais têm uma estrutura espacial, como séries temporais multivariadas que podem ser vistas como imagens ao longo do tempo.

4.3.2. Estrutura da *Convolutional* LSTM

1) Entrada: Dados preprocessados com estrutura espacial-temporal.
2) Camadas *Convolucionais* (Conv1D):

a. Primeira Camada Conv1D: Aplica filtros convolucionais para extrair características locais dos dados temporais.
b. Camadas Convolucionais Adicionais: Podem ser adicionadas para capturar características mais complexas e abstrações espaciais.
c. Camadas LSTM: Após a extração das características espaciais, as camadas LSTM processam as representações temporais.
d. Primeira Camada LSTM: Modela dependências temporais a curto prazo.
e. Camadas LSTM Intermédias: Cada camada subsequente captura padrões temporais mais complexos.
f. Última Camada LSTM: Produz a representação final das dependências temporais.

3) Camada de Saída: Camada densa cujo número de neurónios corresponde ao número de classes que o modelo precisa prever, com ativação *sigmoid*.

5. Desenvolvimento

Este capítulo apresenta o processo de desenvolvimento do projeto, detalhando as etapas fundamentais que foram seguidas para a concretização das abordagens propostas. Após a análise de requisitos e a conceção das arquiteturas discutidas no capítulo anterior, o foco aqui recai na implementação prática das soluções e na adaptação das metodologias de Inteligência Artificial e Deep Learning às necessidades específicas do sistema de classificação de comportamentos de condução.

O desenvolvimento foi organizado em duas abordagens principais: a primeira voltada para a classificação binária dos comportamentos de condução e a segunda para a classificação multiclasses. Cada uma destas abordagens é apresentada com uma descrição detalhada dos métodos de tratamento de dados, a preparação dos *datasets*, a criação dos modelos de redes neuronais LSTM e as etapas subsequentes de treino e avaliação.

Além disso, são discutidas as técnicas de pré-processamento aplicadas aos dados recolhidos, como normalização, redução de ruídos, segmentação e a separação em conjuntos de treino e teste, que foram fundamentais para otimizar o desempenho dos modelos.

Este capítulo é, portanto, uma componente central do trabalho, revelando como a teoria se traduziu em prática e destacando as estratégias adotadas para o desenvolvimento de um sistema de classificação de comportamentos de condução utilizando Redes Neuronais Recorrentes.

5.1. Implementação

A implementação deste projeto passa por várias etapas cruciais que vão desde as descrições até a compilação e treino do modelo. Esta secção irá detalhar cada um dos processos de implementação da solução, fornecendo uma visão abrangente sobre o processamento e a análise dos dados obtidos, além das técnicas de classificação e normalização utilizadas.

Para facilitar a compreensão de cada um dos processos, utilizamos diagramas para descrever a implementação de cada uma das fases. A tabela que se segue tem como propósito ajudar a interpretar os diagramas, proporcionando uma visão organizada dos esquemas de cores utilizados nos mesmos:

5.2.Primeira Abordagem - Classificação de Binária

Tabela 6 - Glossário de Cores Utilizadas nos Diagramas Ilustrativos.

Cor	Significado
Amarelo	Utilizado para identificar funções importantes.
Azul	Utilizado para indicar conjuntos de dados de entrada ou saída.
Verde	Indica estados positivos ou comportamentos classificados como agressivos.
Vermelho	Indica estados negativos ou comportamentos classificados como não agressivos.
Roxo **Rosa** **Azul-Turquesa**	Cores utilizadas para representar diferentes tipos de sensores utilizados na captação dos dados.
Castanho	Utilizado para identificar a classificação dos dados.
Laranja	Cor utilizada para identificar *threshold*.

5.2.1. Descrição e Caracterização dos Dados

Nesta secção, descrevemos e caracterizamos os dados utilizados na implementação, focando-nos em viagens completas de ponto A ao ponto B. Estes dados foram extraídos do IPL-Dataset, que, como descrito anteriormente, foi criado através da aplicação de rastreio de viaturas desenvolvida pelos estudantes do projeto. Os dados foram capturados por sensores durante as viagens e contêm informações detalhadas usadas para identificar diferentes tipos de manobras e comportamentos durante a condução.

5.2.1.1. Sensores Utilizados

- Acelerómetros
- Giroscópio
- GPS

5.2.1.2. Estrutura dos Dados

- **Acelerómetro:** Medidas de aceleração nos eixos X, Y, Z (em m/s^2)
- **Giroscópio:** Medidas angulares nos eixos X, Y, Z (em graus/segundo)
- **Latitude e Longitude**: Coordenadas GPS
- **Tempo**: Timestamp da recolha dos dados
- **Velocidade**: Medida em km/h

5.2.2. Tratamento dos Dados

Os dados foram processados para identificar e organizar diferentes manobras de condução como:

- Aceleração e desaceleração súbita
- Curvas e mudanças de faixa
- Paragens e arranques bruscos

5.2.2.1. Processo de Tratamento de Dados

O processo de Tratamento (ver Figura 23) inicia-se com a captação de dados através de seis sensores, três giroscópios e três acelerómetros, um para cada eixo. Os acelerómetros medem a aceleração ao longo dos eixos X, Y e Z e os giroscópios medem a velocidade angular ao longo desses mesmos eixos.

Com a captação dos dados feita, aplicamos uma função que separa os valores positivos dos negativos (ver Listagem 1) para evitar que possíveis valores negativos possam influenciar no desempenho do modelo. Ficamos assim com duas colunas, uma coluna onde só são armazenados os valores positivos e outra em que só são armazenados os valores negativos recolhidos pelos respetivos sensores.

Por fim estas duas colunas são organizadas num *array* 2D composto por 12 elementos no total, 6 elementos positivos e os 6 elementos negativos correspondentes aos dados processados pelos sensores.

```python
def separate_positives_negatives(data):
    # Ensure the input is converted to a NumPy array for easier manipulation
    data = np.array(data)

    # Create two empty arrays to store positive and negative values
    positives = np.zeros_like(data)
    negatives = np.zeros_like(data)

    # Use boolean indexing to separate positive and negative values
    positives[data > 0] = data[data > 0]
    negatives[data < 0] = -data[data < 0]

    # Combine the positive and negative values into a single 2D array
    return (positives, negatives)
```

Listagem 1 - Função *separate_positives_negatives*

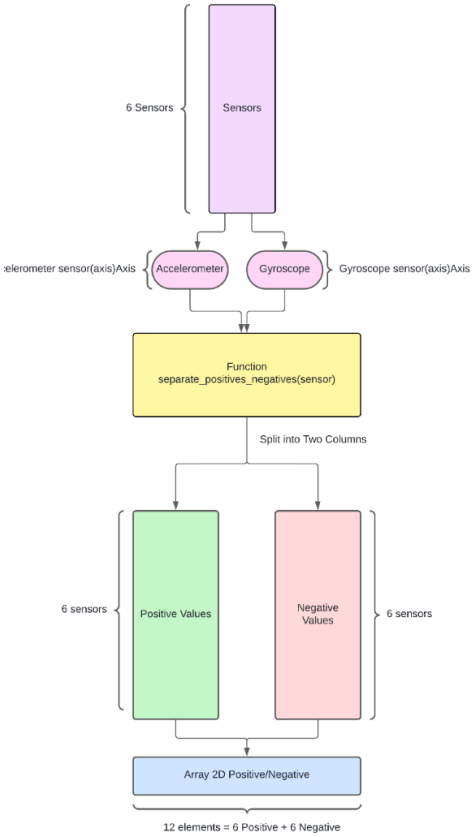

Figura 23 - Diagrama ilustrativo do tratamento dos dados.

5.2.3. Classificação dos Dados

Para efetuar a classificação dos dados foi feita uma função denominada por *y_classification***.** Esta função recebe um conjunto de dados e um *treshold* e retorna uma matriz binária que indica se os valores em cada coluna excedem esses *treshold***.**

5.2.3.1. Descrição dos Parâmetros de Entrada

- ***Data***: Matriz de dados de entrada, onde cada coluna representa um sensor específico.
- ***Treshold*:** Valor percentual usado para determinar o limiar acima do qual os dados serão classificados como 1.

5.2.3.2. Processo de Classificação dos Dados

O processo de classificação dos dados (ver Figura 24) inicia-se com a entrada de dois parâmetros, Data e o *threshold*, para a função *y_classification(data, threshold)*.

A função *y_classification* (ver Listagem 2) tem como objetivo classificar os dados de forma booleana, ou seja, 1 para agressivo e 0 para não agressivo. Para isso, a função começa por inicializar um vetor que irá servir de output no final do processo de classificação. Após isso irá ser feito um *loop* que percorre todas as 12 colunas e que irá calcular o valor máximo de cada coluna dos dados já tratados no tópico anterior, utilizando uma função da biblioteca Numpy denominada por *np.max(data[:, col])*. Com o valor máximo de cada coluna calculado, efetuamos o cálculo do *threshold_pos*, sendo este nada mais que o produto do valor máximo de cada coluna pelo valor do threshold fornecido como parâmetro de entrada, ou seja, *max_value * threshold*. Para a nossa implementação descobrimos, empiricamente, um threshold de **0.3**.

Depois de termos calculado o *threshold_pos*, a classificação dos dados e feita como:

- Se o valor dos dados for maior ou igual ao *threshold_pos*, o dado irá ser classificados como **1**, ou seja, **agressivo**.
- Se o valor dos dados for menor que *threshold_pos*, o dado irá ser classificados como **0**, ou seja, **não agressivo**.

Por fim, com os dados já classificados, é retornado o vetor inicializado no início da função contendo esses mesmos dados.

```
1. def y_classification(data, threshold):
2.   classification = np.zeros_like(data, dtype=int) # Initialize output array
3.
4.   for col in range(0, 12): # Loop through each column
5.     max_value = np.max(data[:, col])
6.     threshold_pos = max_value * threshold
7.     classification[:, col] = np.where(data[:, col] >= threshold_pos, 1, 0)
8.   return classification
```

Listagem 2 - Função *y_classification*

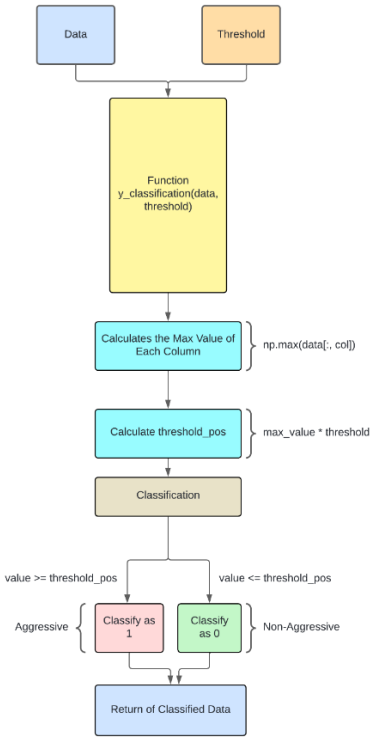

Figura 24 - Diagrama ilustrativo da classificação dos dados.

5.2.4. Normalização dos Dados

Foi feita uma normalização dos dados que permitiu garantir que os diferentes tipos de dados fossem comparáveis e melhorar o desempenho do modelo. Esta normalização envolveu:

- **Escalonamento**: Ajustar os valores de cada variável para um intervalo comum compreendido entre [0, *Max Value* de cada conjunto de sensores]

5.2.4.1. Processo de Normalização dos Dados

O processo de normalização (ver Figura 26) de dados começa com a aplicação da função *max_of_vectors* (ver Listagem 3) que pode ter duas variações:

- Para o acelerómetro, onde os parâmetros de entrada são os: *turnRightX*, *turnLeftX*, *accelY*, *breakY*, *positiveZ*, *negativeZ* (ver Figura 25);
- Para o giroscópio, em que os parâmetros de entrada são: '*gyrPositiveX'*, *gyrNegativeX*, *gyrPositiveY*, *gyrNegativeY*, *gyrPositiveZ* **e** *gyrNegativeZ*

O objetivo desta função e juntar todas as colunas de entrada e combiná-las num vetor único calculando depois o valor máximo deste vetor. O resultado é o retorno do valor máximo do acelerómetro ou do giroscópio que irá ter como papel o parâmetro de entrada da função *normalize_between_0_and_max_v2(data, max_value)* (ver Listagem 4)**.**

```
1. def max_of_vectors(vec1, vec2, vec3, vec4, vec5, vec6):
2.  # Combine all vectors into a single array
3.  all_vectors = np.array([vec1, vec2, vec3, vec4, vec5, vec6])
4.
5.  # Find the maximum value in the array
6.  max_value = np.max(all_vectors)
7.
8.  return max_value
```

Listagem 3 - Função *max_of_vectors*

```
1. def normalize_between_0_and_max_v2(data, max_value):
2.  return data / max_value
```

Listagem 4 - Função *normalize_between_0_and_max_v2*

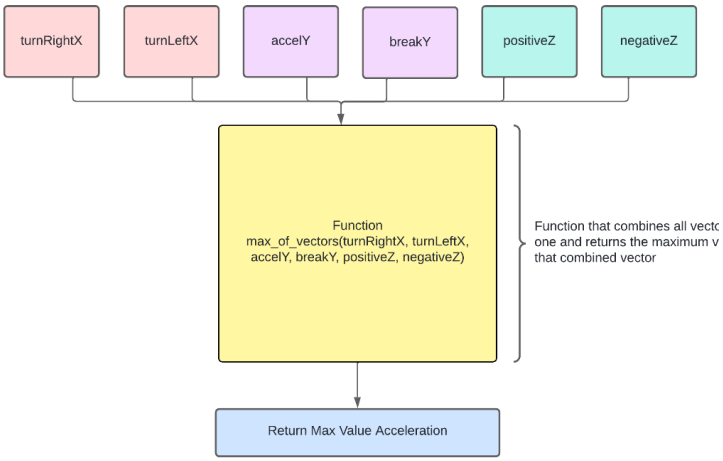

Figura 25 - Diagrama ilustrativo da função *max of vectors* aplicada ao acelerómetro.

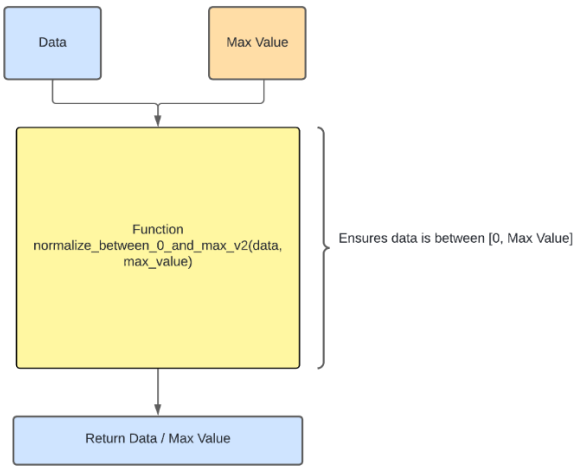

Figura 26 - Diagrama ilustrativo da normalização dos dados

5.2.5. Representação Visual dos Dados

Para uma melhor análise das manobras gravadas no *dataset*, utilizamos a Biblioteca Folium para criar um mapa interativo. Adicionamos um cluster de marcadores ao mapa para agrupar visualmente eventos próximos.

Foram então adicionados marcadores para cada posição de cada manobra efetuada, diferenciados por cores para representar diferentes tipos de manobras. A Figura 27 e Figura 28 representam estas visualizações dos conjuntos de dados:

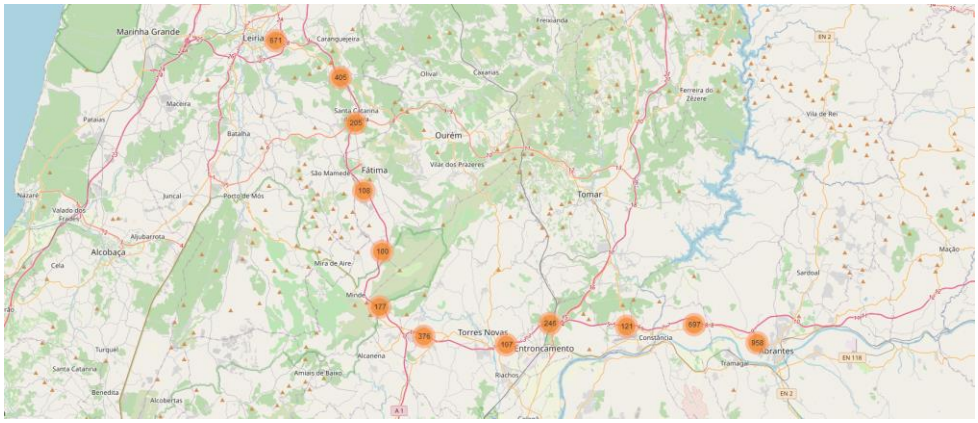

Figura 27 - Gráfico de manobras do *dataset.*

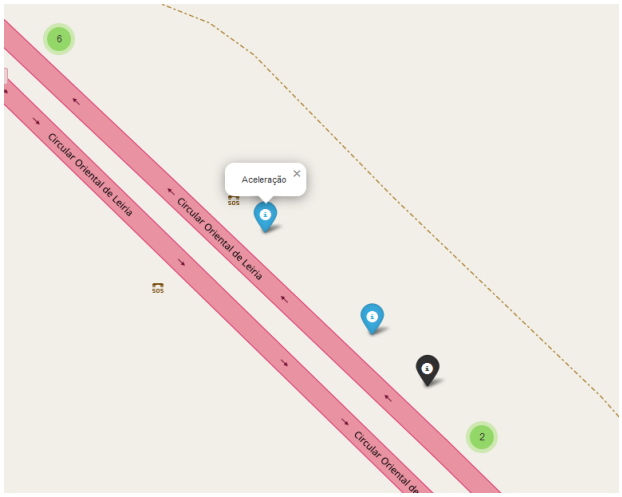

Figura 28 - Gráfico de manobras detalhado do *dataset.*

5.2.6. Separação dos Dados em Treino e Teste

Os dados foram divididos em conjuntos de treino e teste para avaliar a *performance* do modelo. Esta divisão foi feita na seguinte proporção:

- **80%**: Dados para Treino
- **20%**: Dados para Teste

5.2.6.1. Processo de Separação dos Dados

Para a divisão dos dados em conjuntos para treino e teste, foi utilizada a função *split_train_test(data, test_size=0.2)* (ver Listagem 5), que tem como parâmetros de entrada a data e o tamanho que a sequencia de dados de teste irá ter. Este tamanho tem influência direta no tamanho da sequência de treino, pois se o *test_size*=0.2 significa que **20%** dos dados serão utilizados para o modelo realizar os testes, o que faz com que a sequência de treino tenha como tamanho 0.8, ou seja **80%** dos dados (ver Figura 29).

```
1. def split_train_test(data, test_size=0.2):
2.  # Check if test_size is between 0 and 1
3.  if test_size < 0 or test_size > 1:
4.    raise ValueError("test_size must be between 0 and 1.")
5.
6.  # Get the number of samples
7.  num_samples = data.shape[0]
8.
9.  # Calculate the number of samples for each set
10.  train_size = int(num_samples * (1 - test_size))
11.  test_size = num_samples - train_size
12.
16.  # Split the data into training and test sets
17.  train_data = data[:train_size]
18.  test_data = data[train_size:]
19.
20.  return train_data, test_data
```

Listagem 5 - Função *split_train_test*

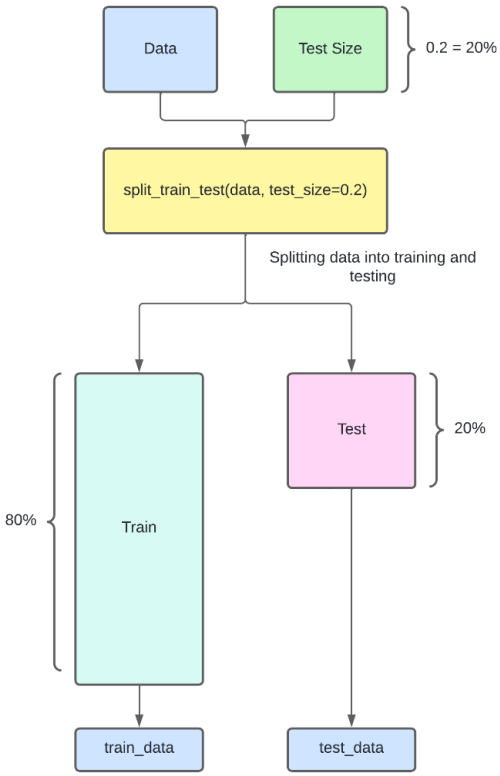

Figura 29 - Diagrama ilustrativo da separação dos dados.

5.2.7. Modelos

Para a criação do modelo, utilizamos uma abordagem baseada em RNN, mais concretamente a arquitetura LSTM.

5.2.7.1. *Stacked* LSTM

- *Sequential Model*
  - Um modelo sequencial que organiza camadas de forma linear.
- *LSTM Layer (64 units, return_sequences=True)*
  - Adiciona uma camada LSTM com 64 unidades. Esta camada é responsável por capturar dependências temporais nos dados sequenciais. Como *return_sequences=True*, a camada retorna a sequência completa de saídas para a próxima camada.
- *Dropout Layer*
  - Desativa uma % das unidades da camada anterior aleatoriamente para prevenir overfitting.
- LSTM *Layer* (64 units, return_sequences=False)
  - Adiciona uma segunda camada LSTM com 64 unidades. Diferente da primeira, esta camada não retorna a sequência completa, mas apenas a última saída, que será passada para as próximas camadas.
- Dense *Layer* (32 units, ReLU activation)
  - Adiciona uma camada densa com 32 unidades. A ativação ReLU é usada para introduzir não-linearidade, permitindo ao modelo capturar padrões complexos nos dados
- Dense *Layer* (12 units, Sigmoid activation)
  - Camada de saída com 12 unidades, com função de ativação *sigmoid*.

```
model_lstm = Sequential()
model_lstm.add(LSTM(64, input_shape=(1, train.shape[2]), return_sequences=True))
model_lstm.add(Dropout(0.2))

model_lstm.add(LSTM(64))
model_lstm.add(Dropout(0.2))

model_lstm.add(Dense(32,activation='relu'))
model_lstm.add(Dense(12,activation='sigmoid'))

# Compile the model
model_lstm.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Listagem 6 - Criação do modelo *Stacked* LSTM Primeira Abordagem

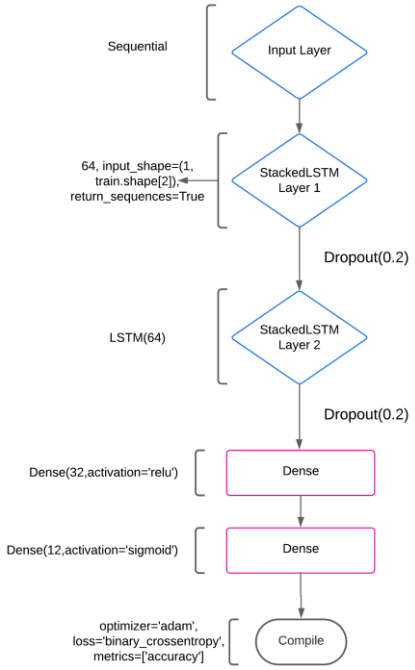

Figura 30 - Diagrama ilustrativo da estrutura do modelo *Stacked* LSTM.

5.2.7.2. *Bidirectional* LSTM

- *Sequential Model*
  - Um modelo sequencial que organiza camadas de forma linear.
- *Bidirectional LSTM Layer (64 units, return_sequences=True)*
  - Adiciona uma camada LSTM *bidirectional* com 64 unidades. Esta camada processa os dados nas direções (para frente e para trás), capturando dependências temporais passadas e futuras. A camada devolve a sequência completa de saídas para a próxima camada.
- *BatchNormalization Layer*
- *Dropout Layer*
- *Bidirectional* LSTM *Layer* (64 units, return_sequences=False)
  - Adiciona outra camada LSTM *bidirectional* com 64 unidades. Diferente da primeira, esta camada não retorna a sequência completa, mas apenas a última saída.
- Dense Layer (12 units, Sigmoid activation)

```
1. # Model configuration
2. model = Sequential()
3.   model.add(Bidirectional(LSTM(64,   return_sequences=True),   input_shape=(train.shape[1],
train.shape[2])))
4. model.add(BatchNormalization())
5. model.add(Dropout(0.5))
6
7. model.add(Bidirectional(LSTM(64, return_sequences=False)))
8. model.add(BatchNormalization())
9. model.add(Dropout(0.2))
10
11. model.add(Dense(64, activation='relu'))
12. model.add(BatchNormalization())
13. model.add(Dropout(0.1))
14
15. model.add(Dense(12, activation='sigmoid'))
16.
16. # Compile the model
18. model.compile(loss='binary_crossentropy',optimizer='adam', metrics=['accuracy'])
```

Listagem 7 - Criação do modelo *Bidirectional* LSTM Primeira Abordagem

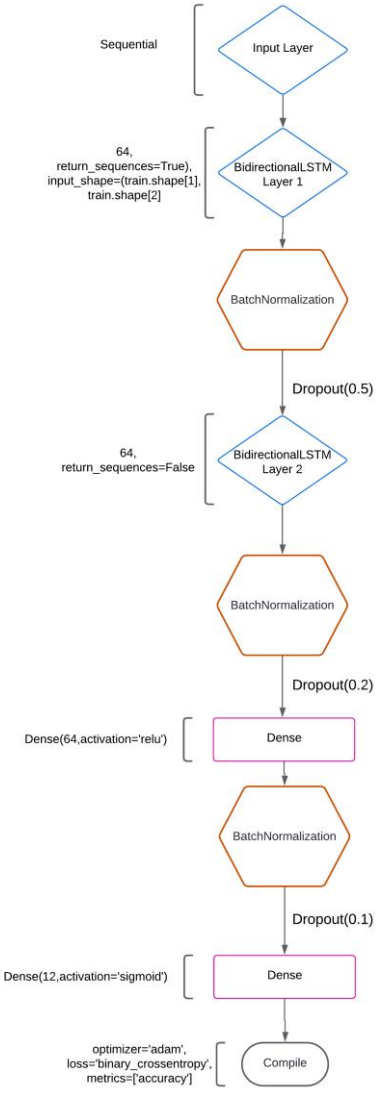

Figura 31 - Diagrama ilustrativo da estrutura do modelo *Bidirectional* LSTM.

5.2.7.3. Convolutional LSTM

- *Sequential Model*
  - Um modelo sequencial que organiza camadas de forma linear.
- *Conv1D Layer*
  - Adiciona uma camada de convolução unidimensional com 64 filtros e tamanho de *kernel 1*. Ativação ReLU é usada para introduzir não-linearidade. A camada espera uma entrada com a forma (*train.shape[1], train.shape[2]*)
- *BatchNormalization Layer*
- *Dropout Layer*
- LSTM *Layer* (256 *units*, *return_sequences=True*)
  - Adiciona uma camada LSTM com 256 unidades, retornando à sequência completa de saída para a próxima camada.
- LSTM *Layer* (128 *units*)
- Dense Layer (12 units, Sigmoid activation)

```
 1. model = Sequential()
 2. model.add(Conv1D(filters=64, kernel_size=1, activation='relu', input_shape=(train.shape[1],
train.shape[2])))
 3. model.add(BatchNormalization())
 4. model.add(Dropout(0.5))
 5. model.add(LSTM(256, return_sequences=True))
 6. model.add(Dropout(0.2))
 7. model.add(LSTM(128))
 8. model.add(Dropout(0.2))
 9. model.add(Dense(12, activation='sigmoid'))
10.
11. model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Listagem 8 - Criação do modelo *Convolutional* LSTM Primeira Abordagem

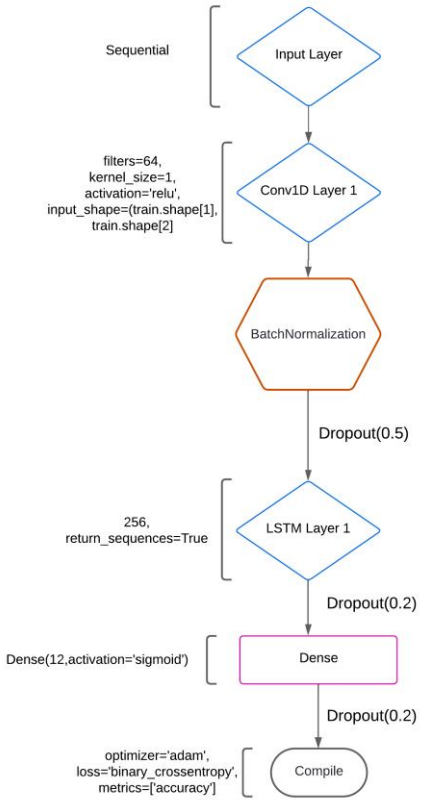

Figura 32 - Diagrama ilustrativo da estrutura do modelo *Convolutional* LSTM.

5.2.8. Compilação e Treino

5.2.8.1. *Stacked* LSTM

```
1. model_lstm.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Listagem 9 - Compilação do modelo *Stacked* LSTM Primeira Abordagem

```
 1. # Define callbacks
 2.
 3. best_model_file = 'runtime_saves/models/Stacked LSTM /model.keras'
 4. best_model = ModelCheckpoint(best_model_file, monitor='val_loss', mode='min', verbose=1,
save_best_only=True)
 5.
 6. early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
 7. reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, verbose=1,
mode='min', min_lr=0.0000)
 8. best_model = ModelCheckpoint(best_model_file, monitor='val_loss', mode='min', verbose=1,
save_best_only=True)
 9.
10. # Train the model
11. history = model_lstm.fit(train, y_train, epochs=30, batch_size=64,
callbacks=[early_stopping,best_model,reduce_lr], validation_data=(test, y_test))
```

Listagem 10 - Treino do modelo *Stacked* LSTM Primeira Abordagem

5.2.8.2. *Bidirectional* LSTM

```
1. model_lstm.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Listagem 11 - Compilação do modelo *Bidirectional* LSTM Primeira Abordagem

```
# best callback for the model
best_model_file = 'runtime_saves/models/BiLSTM/model.keras'
best_model = ModelCheckpoint(best_model_file, monitor='val_loss', mode='min', verbose=1, save_best_only=True)

# early stopping callback
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)

# fit the model
history = model.fit(train, y_train, epochs=30, batch_size=256 , validation_data=(test, y_test), verbose=1, callbacks=[best_model, early_stop])
```

Listagem 12 - Treino do modelo *Bidirectional* LSTM Primeira Abordagem

5.2.8.3. *Convolutional* LSTM

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Listagem 13 - Compilação do modelo *Convolutional* LSTM Primeira Abordagem

```
# best callback for the model
best_model_file = 'runtime_saves/models/ConvLSTM/model.keras'
best_model = ModelCheckpoint(best_model_file, monitor='val_loss', mode='min', verbose=1, save_best_only=True)

# early stopping callback
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)

# fit the model
history = model.fit(train, y_train, epochs=30, batch_size=256 , validation_data=(test, y_test), verbose=1, callbacks=[best_model, early_stop])
```

Listagem 14 - Treino do modelo *Convolutional* LSTM Primeira Abordagem

5.3.Segunda Abordagem – Classificação de Classes

Na segunda abordagem, foi utilizado o IPL-*Dataset* para classificar manobras de condução em diferentes cenários, como aceleração, travagem e interseções. As manobras foram previamente categorizadas como "***Slow***", "***Normal***" ou "***Aggressive***". O processo incluiu a normalização dos dados, o tratamento das séries temporais e a aplicação de técnicas de ML para analisar o comportamento de condução com base em dados de sensores como Acelerómetros, giroscópios e GPS. O objetivo principal foi identificar padrões que pudessem distinguir diferentes estilos de condução.

5.3.1. Descrição e Caracterização dos Dados

Os dados fornecidos no dataset IPL-*Dataset* contem curtas gravações de manobras ou cenários comuns de condução como Aceleração, Travagem e Interseções. As manobras foram categorizadas antes do iniciar a gravação como ***Slow***, ***Normal*** e ***Aggressive***.

5.3.2. Tratamento dos Dados

Os dados foram processados com o objetivo de classificar sequências temporais de diferentes cenários de condução. A abordagem consistiu em várias etapas de pré-

processamento para garantir a qualidade e consistência dos dados antes de serem usados para treinar os modelos (ver Figura 33).

O processo de tratamento dos dados segue os seguintes passos:

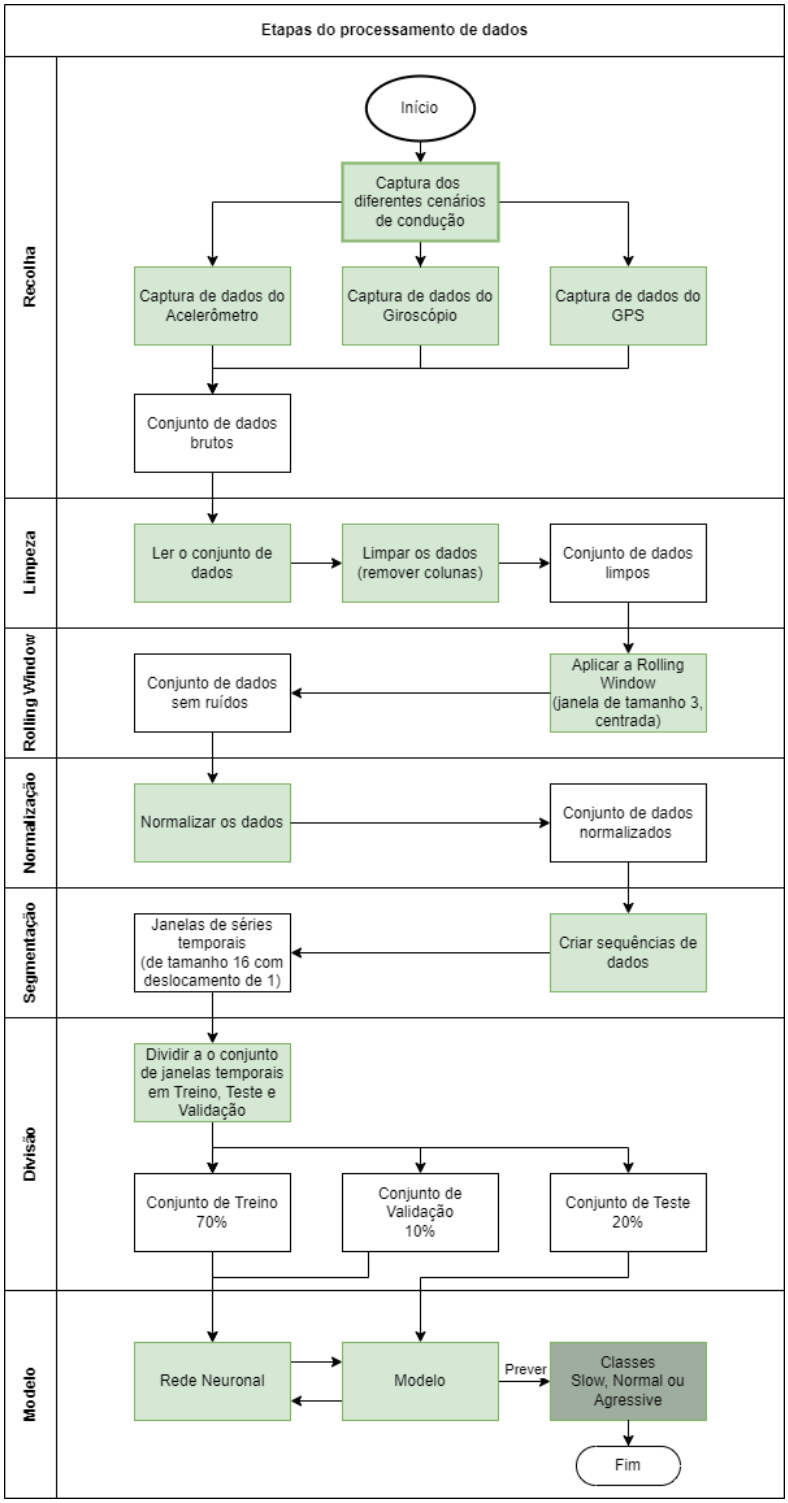

Figura 33 - Etapas de processamento de dados da segunda abordagem.

5.3.2.1. Importação dos Cenários:

Os dados de diferentes cenários (como Aceleração, Travagem e Interseções) foram importados para os dois carros usados no estudo (BMW e Honda).

5.3.2.2. Limpeza e Estruturação dos Dados:

Durante a importação, as colunas irrelevantes para o treino do modelo, como ***timestamp***, ***id***, e ***createdAt*** foram removidas.

5.3.2.3. Aplicação de *Rolling Window*:

Para reduzir o ruído nos dados, foi aplicada uma técnica de ***Rolling Window***, com uma janela de tamanho 3, centrada, nas colunas dos Acelerómetros e giroscópios. Esta técnica suaviza as flutuações bruscas nos dados, resultando em medições mais estáveis. Podemos ver um exemplo de aplicação desta técnica na Tabela 7 e na Tabela 8.

Tabela 7 – Exemplo de dados antes de aplicar *Rolling Window.*

accelerometerX	accelerometerY	accelerometerZ	gyroscopeX	gyroscopeY	gyroscopeZ
0.204224	0.574472	0.015077	0.001833	0.00672	0.003054
0.230342	0.175526	0.122981	-0.00061	0	-0.00244
0.199742	1.303585	-0.18821	0.014661	-0.01344	-0.00305
0.003083	0.753614	0.103433	0.034208	-0.01161	-0.01161
0.202767	0.281722	0.290354	0.023213	-0.01527	-0.03665
…	…	…	…	…	…

Tabela 8 - Exemplo de dados depois de aplicar *Rolling Window.*

accelerometerX	accelerometerY	accelerometerZ	gyroscopeX	gyroscopeY	gyroscopeZ
0.217283	0.374999	0.069029	0.000611	0.00336	0.000305
0.211436	0.684528	-0.01672	0.005294	-0.00224	-0.00081
0.144389	0.744242	0.012736	0.016086	-0.00835	-0.0057
0.135198	0.77964	0.068527	0.024027	-0.01344	-0.0171
0.054488	0.322867	0.121088	0.019344	-0.01018	-0.03014
…	…	…	…	…	…

5.3.2.4. Normalização dos Dados:

Após a suavização dos dados, foi aplicada uma normalização nas colunas do Acelerómetro e giroscópio, escalando os valores para um intervalo entre 0 e 1. Esta etapa é fundamental para melhorar a performance do modelo e aumentar a precisão do mesmo.

5.3.2.5. Codificação de *Labels*:

As etiquetas de classe (***Slow***, ***Normal***, ***Aggressive***) foram codificadas utilizando o ***LabelEncoder*** da biblioteca **sklearn**, transformando as etiquetas categóricas em valores numéricos que podem ser interpretados pelo modelo.

5.3.2.6. Criação de Sequências Temporais:

Os dados então estruturados em sequências temporais com uma janela de tamanho 16, com um deslocamento de 1. A etiqueta associada a cada sequência foi definida como a mais frequente dentro daquela janela (ver Figura 34).

Figura 34 - Exemplo ilustrativo de sequências temporais de dados.

5.3.2.7. Divisão dos Dados:

Os dados foram divididos em conjuntos de treino, teste e validação, com proporções de **70%**, **20%** e **10%**, respetivamente. Essa divisão foi feita de forma estratificada para garantir que a distribuição das classes fosse consistente em todos os conjuntos.

- **70%**: Treino
- **20%**: Teste
- **10%**: Validação

5.3.3. Modelos

5.3.3.1. Stacked LSTM

Descrição do Modelo

O modelo *Stacked* LSTM foi projetado para a classificação de sequências temporais com base nos dados de sensores do IPL-*Dataset*.

Arquitetura da Rede

- **Entrada**: A entrada do modelo foi configurada para aceitar sequências temporais de comprimento 16 e 8 atributos (dados normalizados dos Acelerómetros e giroscópios nos três eixos, mais latitude e longitude).

- **Camadas LSTM Empilhadas**: O modelo utiliza três camadas LSTM empilhadas, cada uma com 32 unidades. Cada uma dessas camadas é seguida por uma camada de ***Batch Normalization*** para estabilizar e acelerar o treino, garantindo que as distribuições dos dados em cada **mini-batch** permaneçam consistentes.
  - A primeira camada LSTM processa a sequência e passa a saída para a próxima camada.
  - A segunda camada LSTM adiciona profundidade ao modelo, processando a saída da camada anterior.
  - A terceira camada LSTM retorna uma sequência de saída final que é processada pela camada densa.

- **Camada de *Dropout***: Após a última camada LSTM, foi adicionada uma camada de ***Dropout*** com uma taxa de **0.4** para reduzir o risco de ***overfitting***, o que envolve desligar aleatoriamente uma fração das unidades de rede durante o treino.

- **Camada de Saída**: Os dados processados pelas camadas LSTM são então passados por uma camada densa com função de ativação **softmax**, que produz as probabilidades para cada uma das três classes: ***Slow***, ***Normal***, ***Aggressive***.

5.3.3.2. Bidirectional LSTM

Descrição do Modelo: Ao contrário das LSTM tradicionais, que processam as sequências de dados numa única direção (do passado para o futuro), o *Bidirectional* LSTM processa a sequência de dados em ambas as direções. Isso permite que o modelo capture tanto as informações passadas quanto as futuras em qualquer ponto da sequência, o que se pode demonstrar muito vantajoso para o nosso problema.

Arquitetura da Rede:

- **Entrada**: Idêntico à *Stacked* LSTM, a entrada do modelo foi configurada para aceitar sequências temporais de comprimento 16 e 8 atributos.
- **Camadas LSTM Bidirecionais**: O modelo utiliza três camadas LSTM bidirecionais, cada uma com 64 unidades. Cada uma dessas camadas é seguida por uma camada de ***Batch Normalization*** e ***Dropout*** para prevenir ***overfitting*** e acelerar o treino.
    - A primeira camada LSTM *bidirectional* processa a sequência e passa a saída para a próxima camada.
    - A segunda camada também é LSTM *bidirectional* e adiciona profundidade ao modelo.
    - A terceira e última camada LSTM *bidirectional* retorna uma sequência de saída final que é processada pela camada densa.
- **Camada de Saída**: Após as camadas LSTM, os dados passam por uma camada densa com função de ativação **softmax**, que produz a probabilidade para cada uma das três classes: ***Slow***, ***Normal***, ***Aggressive***.

5.3.3.3. Compilação e Treino

Os modelos foram compilados utilizando o otimizador ***Adam*** e a função de perda ***Sparse Categorical Entropy***, adequada para problemas de classificação com múltiplas classes. O treino dos modelos foi conduzido com um número máximo de **150 épocas** e um ***batch* de tamanho 16**, utilizando callbacks como ***EarlyStopping***, ***ReduceLROnPlateau*** e ***ModelCheckpoint*** para monitorar e ajustar automaticamente hiperparâmetros.

6. Testes e Resultados

Neste capítulo iremos apresentar através de tabelas comparativas de métricas de avaliação sobre as diferentes arquiteturas desenvolvidas nas duas abordagens. O principal objetivo dos testes é avaliar a precisão e a robustez do modelo LSTM classificador de condução, garantindo que ele possa ser aplicado eficazmente em cenários reais.

6.1. Métricas de Avaliação

As métricas utilizadas para avaliar o desempenho dos modelos incluem:

- TP (*True Positives*) são os verdadeiros positivos,
- TN (*True Negatives*) são os verdadeiros negativos,
- FP (*False Positives*) são os falsos positivos,
- FN (*False Negatives*) são os falsos negativos.

6.1.1.1. *Accuracy*

A *accurary* (exatidão em português) é a métrica mais intuitiva e simples para avaliação de modelos de classificação. É definida como a razão entre o número de previsões corretas e o número total de previsões. A fórmula é dada por:

$$\frac{TP + TN}{TP + TN + FP + FN}$$

6.1.1.2. Error

O *error* (erro em português) é o complemento da métrica *accuracy*. A fórmula é dada por:

$$\frac{FP \ + \ FN}{TP + TN \ + \ FP + \ FN}$$

6.1.1.3. Precision

A *precision* (precisão em português) mede a proporção de verdadeiros positivos entre todas as instâncias classificadas como positivas [48]. É uma métrica importante quando o custo de falsos positivos é alto. A fórmula é dada por:

$$\frac{TP}{TP + FP}$$

6.1.1.4. *Recall* (Sensibilidade)

O *recall* mede a proporção de verdadeiros positivos entre todas as instâncias que são realmente positivas [48]. É uma métrica crucial quando o custo de falsos negativos é alto. A fórmula é dada por:

$$\frac{TP}{TP + FN}$$

6.1.1.5. F1 Score

O *F1 Score* é a média harmónica da precisão e do *recall*, proporcionando um único valor que considera ambas as métricas [48]. É especialmente útil quando há um *trade-off* entre precisão e *recall*. A fórmula é dada por:

$$2 \times \frac{Precision \times Recall}{Precision + Recall}$$

6.1.1.6. Jaccard Score

O *Jaccard Score* mede a similaridade entre o conjunto de rótulos preditos e o conjunto de rótulos verdadeiros [48]. Em problemas de classificação binária, a fórmula é dada por:

$$\frac{TP}{TP + FP + FN}$$

6.1.1.7. Hamming Loss

A *Hamming Loss* mede a taxa de predições incorretas, onde cada predição incorreta conta igualmente [48]. Para classificação binária, a fórmula é dada por:

$$\frac{1}{n}\sum_{i=1}^{n} 1(\hat{y}\imath \neq yi)$$

- N é o número total de amostras.
- $\hat{y}\imath$ é o rótulo previsto para a i-ésima amostra.
- yi é o rótulo verdadeiro para a i-ésima amostra.
- $1(\hat{y}\imath \neq yi)$ é uma função indicadora que vale 1 se $\hat{y}\imath \neq yi$ e 0 caso contrário.

6.1.1.8. *Per-Class Error* (PCE)

O *Per-Class Error* (PCE) mede o erro de classificação para cada classe individualmente, ajudando a identificar como o modelo se comporta para classes específicas [48]. A fórmula é dada por:

$$1 - accuracy_{Class}$$

6.1.1.9. *Mean Per-Class Error* (MPCE)

O *Mean Per-Class Error* (MPCE) é a média dos erros por classe, fornecendo uma visão geral do desempenho do modelo em todas as classes [48]. A fórmula é dada por:

$$\frac{1}{n}\sum_{i=1}^{n} PCE$$

6.1.1.10. *Area Under the Curve* (AUC)

A *Area Under the Curve* (AUC) avalia o desempenho do classificador ao calcular a área sob a curva ROC (*Receiver Operating Characteristic*), que é uma representação gráfica da relação entre a taxa de verdadeiros positivos (TPR) e a taxa de falsos positivos (FPR) [49]. Um valor de AUC de 1 indica uma separação perfeita, enquanto um valor de 0,5 indica uma performance aleatória [48]. A fórmula é dada por:

$$\int_0^1 TPR(FPR)\ d(FPR)]$$

- TPR é a Taxa de Verdadeiros Positivos (*True Positive Rate*);
- FPR é a Taxa de Falsos Positivos (*False Positive Rate*);
- A integral calcula a área sob a curva ROC.

6.2.Primeira Abordagem - Classificação Binária de Manobras

Tabela 9 - Comparação Geral de resultados da Primeira Abordagem.

Metrics	*Proposed* ConvLSTM	BiLSTM	*Stacked* LSTM
Accuracy	97.71%	96.06%	96.38%

Precision	95.40%	84.66%	95.00%
F1 Score	93.02%	87.56%	85.65%
Recall	93.96%	81.66%	89.29%
Hamming Loss	0.20%	0.33%	0.32%
Jaccard Score	88.95%	72.69%	81.95%

6.2.1. Resultados da arquitetura *Stacked* LSTM

Nesta subseção, são apresentados os resultados obtidos com a arquitetura Stacked LSTM (ver Tabela 10). Os resultados são analisados em termos de métricas de desempenho, como precisão, recall, F1-score e AUC. E os resultados são também analisados com recurso à matriz de confusão agregada (ver Figura 35) e às curvas de precisão/*recall (ver* Figura 36*)*

Tabela 10 - Resultados por classe do modelo Stacked LSTM da primeira abordagem.

Manuever	*Precision (%)*	*Recall (%)*	*F1-Score (%)*	*AUC (%)*
Left Turn (Acc X+)	96	91	94	97
Right Turn (Acc X-)	90	97	94	99
Acceleration (Acc Y+)	87	98	92	98
Braking (Acc Y-)	72	41	52	94
Ascent (Acc Z+)	98	62	76	96
Descent (Acc Z-)	100	45	62	93
Left Lateral Tilt (Gyro X+)	87	88	87	93
Right Lateral Tilt (Gyro X-)	91	91	91	97
Forward Tilt (Gyro Y+)	95	88	92	97
Backward Tilt (Gyro Y-)	94	96	95	98
Left Turn (Gyro Z+)	93	94	94	89
Right Turn (Gyro Z-)	88	52	65	87

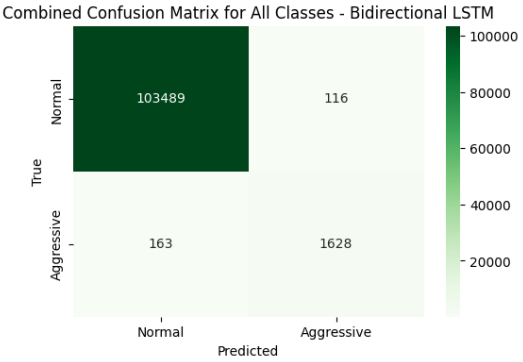

Figura 35 - Matriz de Confusão Agregada - *Stacked* LSTM da primeira abordagem.

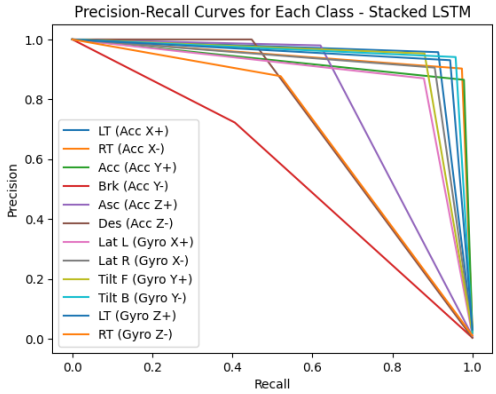

Figura 36 - Curvas de *Precison*/*Recall* - Stacked LSTM da primeira abordagem.

6.2.2. Resultados da arquitetura *Bidirectional* LSTM

Nesta subseção, são apresentados os resultados obtidos com a arquitetura Stacked LSTM (ver Tabela 11). Os resultados são também analisados com recurso à matriz de confusão agregada (ver Figura 37) e às curvas de precisão/*recall (ver* Figura 38*)*

Tabela 11 - Resultados por classe do modelo Bidirectional LSTM da primeira abordagem.

Manuever	*Precision (%)*	*Recall (%)*	*F1-Score (%)*	*AUC (%)*
Left Turn (Acc X+)	98	94	96	97
Right Turn (Acc X-)	97	99	98	99
Acceleration (Acc Y+)	76	96	85	98
Braking (Acc Y-)	88	88	88	94
Ascent (Acc Z+)	95	91	93	96
Descent (Acc Z-)	69	86	77	93
Left Lateral Tilt (Gyro X+)	99	87	92	93
Right Lateral Tilt (Gyro X-)	97	94	96	97
Forward Tilt (Gyro Y+)	97	94	96	97

Backward Tilt (Gyro Y-)	92	95	94	98
Left Turn (Gyro Z+)	99	79	88	89
Right Turn (Gyro Z-)	99	75	85	87

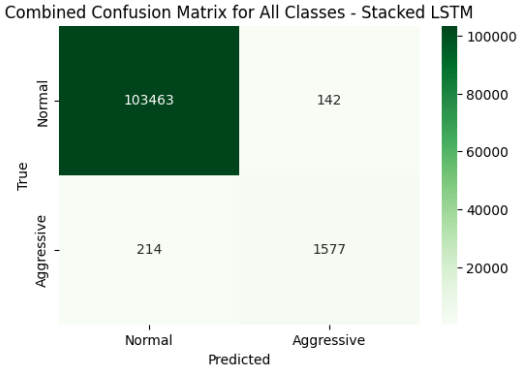

Figura 37 - Matriz de Confusão Agregada - *Bidirectional* LSTM da primeira abordagem.

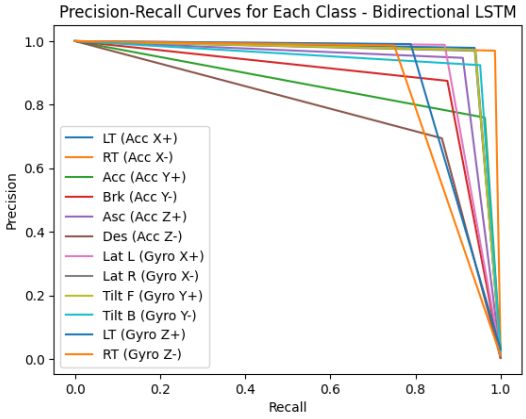

Figura 38 - Curvas de *Precison*/*Recall* - *Bidirectional* LSTM da primeira abordagem.

6.2.1. Resultados da arquitetura *Convolutional* LSTM

Nesta subseção, são apresentados os resultados obtidos com a arquitetura Stacked LSTM (ver Tabela 12). Os resultados são também analisados com recurso à matriz de confusão agregada (ver Figura 39) e às curvas de precisão/*recall (ver* Figura 40*)*

Tabela 12 - Resultados por classe do modelo Convolutional LSTM da primeira abordagem.

Manuever	*Precision (%)*	*Recall (%)*	*F1-Score (%)*	*AUC (%)*
Left Turn (Acc X+)	99	94	97	97
Right Turn (Acc X-)	97	99	98	100
Acceleration (Acc Y+)	98	93	95	96
Braking (Acc Y-)	100	78	88	89

Ascent (Acc Z+)	99	89	93	94
Descent (Acc Z-)	68	90	78	95
Left Lateral Tilt (Gyro X+)	98	94	96	97
Right Lateral Tilt (Gyro X-)	94	99	97	100
Forward Tilt (Gyro Y+)	100	91	95	95
Backward Tilt (Gyro Y-)	98	93	96	97
Left Turn (Gyro Z+)	100	89	94	95
Right Turn (Gyro Z-)	100	65	78	82

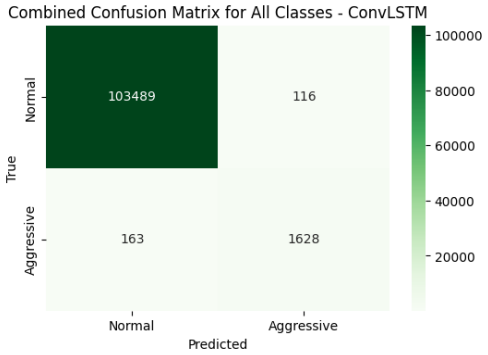

Figura 39 - Matriz de Confusão Agregada - *Convolutional* LSTM da primeira abordagem.

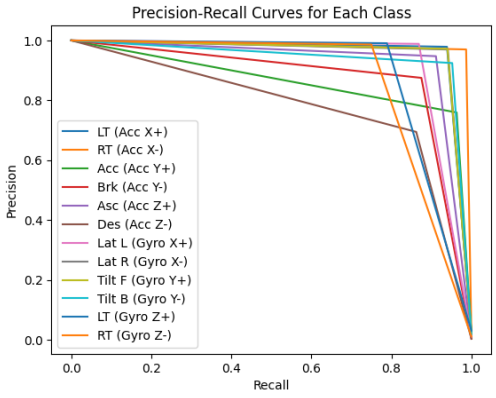

Figura 40 - Curvas de *Precison*/*Recall* - *Convolutional* LSTM da primeira abordagem.

6.2.2. Análise de resultados

Nesta seção, discutimos os resultados obtidos e comparamos o desempenho das diferentes arquiteturas de redes neuronais.

Durante a fase de treinos dos modelos, concentramos todos os esforços para reduzir ao máximo o ***val_loss*** e a ***loss*** visto a ***accuracy*** durante o treino não ser um bom indicador em

problemas de classificação binária. Para uma melhor compreensão dos resultados decidimos representar graficamente o desempenho dos modelos. Com isto podemos observar que:

6.2.2.1. Convolutional LSTM

Por cada época de treino, a ***loss*** e a ***val_loss*** foram diminuindo até chegarem a um valor muito baixo o que indica que indica que não ouve ***overfitting*** pois ambas diminuíram progressivamente como podemos observar na Figura 41. Esta foi a arquitetura a atingir mais rapidamente valores próximos de **0,** logo a partir da **2º época.**

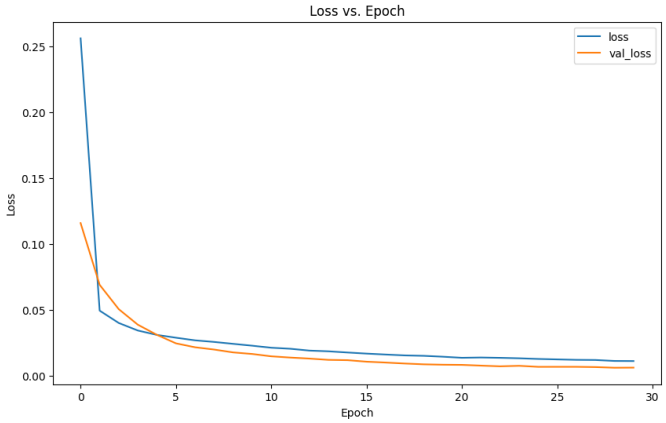

Figura 41 - Comparação da *loss* e *val_loss* para o modelo *Convolutional* LSTM Primeira Abordagem.

Na Figura 42 conseguimos realizar uma comparação entre os valores reais e os valores previstos pelo modelo nos **30** primeiros casos. Com isto conseguimos observar que o modelo para índices onde os valores reais são **0**, ou seja, a maioria dos casos, o modelo consegue prever com elevada precisão valores muito próximos de **0**. Por outro lado, o modelo também consegue prever com elevado grau de proximidade valores próximos de **1**.

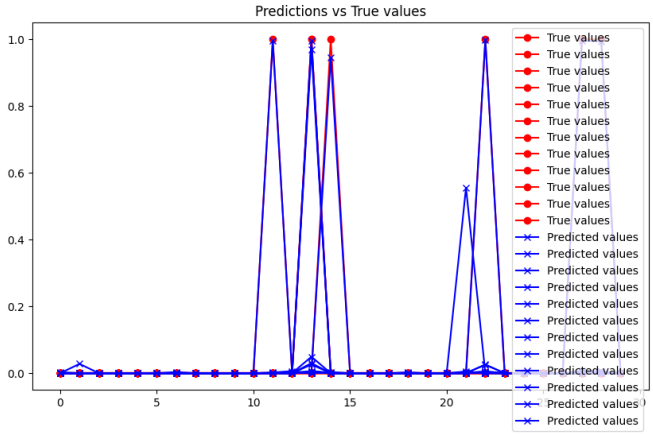

Figura 42 - Comparação dos valores reais vs valores previstos pelo modelo *Convolutional* LSTM primeira abordagem.

6.2.2.2. Stacked LSTM

Neste modelo conseguimos perceber que a diminuição da ***loss*** e a ***val_loss*** foi um bocado menor por cada época de treino e conseguiram chegar a valores perto de **0** um bocado mais lentamente que a ***Convolutional* LSTM**. Também não existiu risco de overfitting como podemos comprovar na Figura 43.

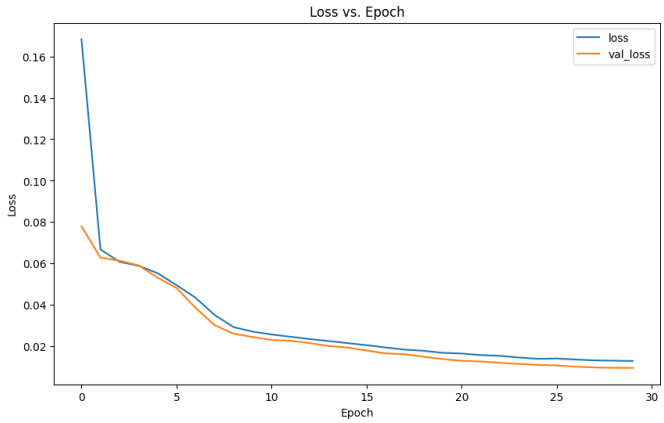

Figura 43 - Comparação da *loss* e *val_loss* para o modelo *Stacked* LSTM primeira abordagem.

Ao contrário do que acontece da arquitetura ***Convolutional***, a ***Stacked LSTM*** teve mais dificuldades em prever os valores nos **30** primeiros casos. Com isto conseguimos observar que o modelo para índices onde os valores reais são **0**, ou seja, a maioria dos casos, acerta em grande parte deles, mas falha numa parte significativa, sendo estes erros muito próximos aos valores reais. Por outro lado, o modelo também consegue prever com um grau significativo de proximidade valores próximos de **1**, mas com alguns erros também (ver Figura 44).

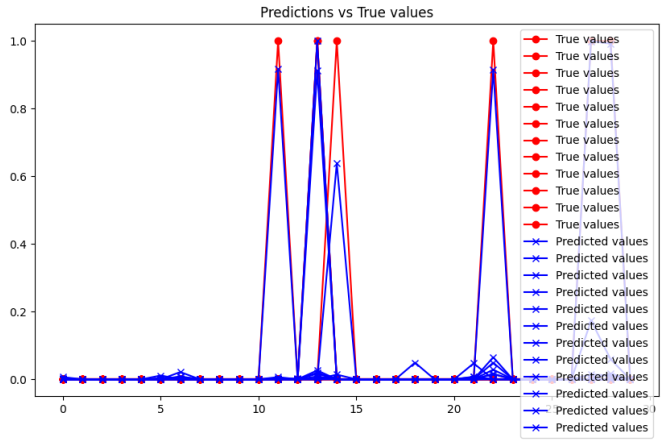

Figura 44 - Comparação dos valores reais vs valores previstos pelo modelo *Stacked* LSTM primeira abordagem.

6.2.2.3. Bidirectional LSTM

A arquitetura ***Bidirectional*** foi mais rápida, comparando com a ***Stacked*** **LSTM,** a atingir valores da ***loss*** e ***val_loss*** próximos de 0. Neste modelo conseguimos perceber que a partir da **3º ronda** os estes valores já estão abaixo do 0.1 o que na ***Stacked*** **LSTM** só ocorre a partir da **24 º ronda.** Também não existiu risco de *overfitting* como podemos comprovar na Figura 45.

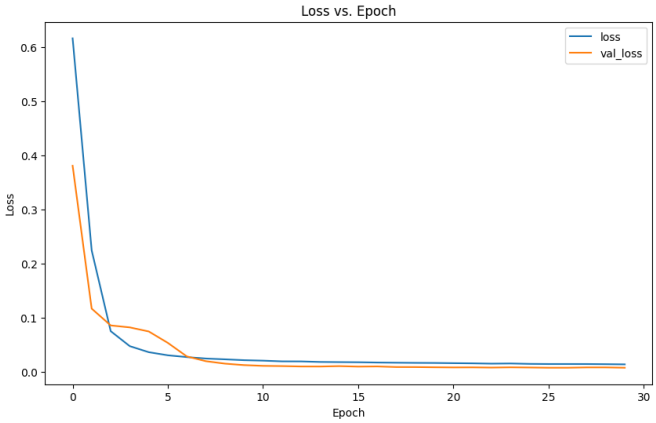

Figura 45 - Comparação da *loss* e *val_loss* para o modelo *Bidirectional* LSTM primeira abordagem.

Em contrapartida a ***Bidirectional*** **LSTM** teve dificuldades em prever os valores nos **30** primeiros casos. Com isto conseguimos observar que o modelo para índices onde os valores reais são **0**, ou seja, a maioria dos casos, acerta em grande parte deles, mas, como na ***Stacked*** **LSTM** falha também numa parte significativa, sendo estes erros muito próximos aos valores reais. Por outro lado, o modelo também consegue prever com um grau significativo de proximidade valores próximos de **1**, mas com alguns erros significativos (ver Figura 46).

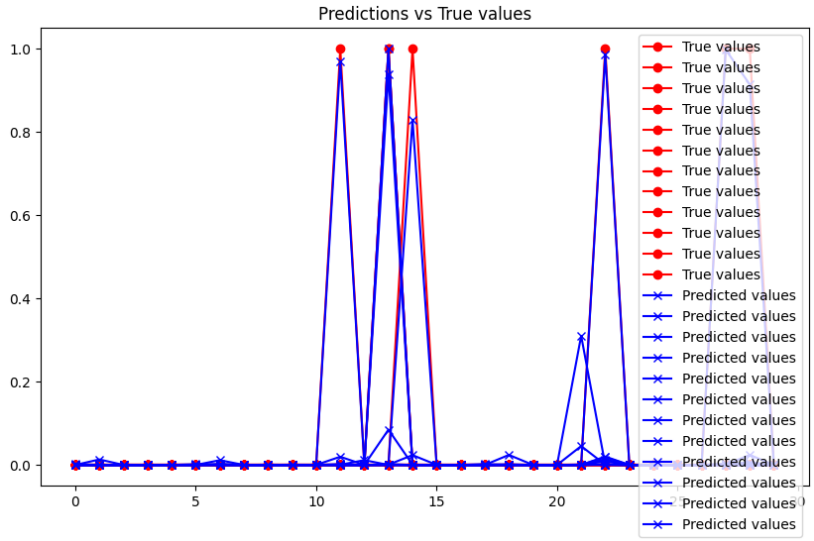

Figura 46 - Comparação dos valores reais vs valores previstos pelo modelo *Bidirectional* LSTM primeira abordagem.

6.3. Segunda Abordagem - Classificação Multiclasse de Estilos de Condução

Aqui apresentamos os resultados da segunda abordagem (ver Tabela 13) onde utilizamos diferentes cenários previamente categorizadas como "***Slow***", "***Normal***" ou "***Aggressive***" para criar e classificar sequências temporais de dados.

Tabela 13 - Comparação Geral de resultados da segunda abordagem.

Metric	BiLSTM	*Stacked*LSTM
Train Accuracy	**99.62%**	97.64%
Test Accuracy	**96.56%**	96.32%
Train Loss	**0.0209**	0.0656
Test Loss	**0.0839**	0.0856
Precision	96.73%	**97.24%**
Recall	96.32%	**97.16%**
F1 *Score*	96.51%	**97.2%**
MPCE	3.68%	**2.83%**

6.3.1. Resultados da arquitetura *Stacked* LSTM

Nesta subseção, são apresentados os resultados obtidos com a arquitetura *Stacked* LSTM (ver Tabela 14). Os resultados são analisados em termos de métricas de desempenho, como precisão, recall, F1-score e erro por classe.

Tabela 14 - Resultados por classe do modelo *Stacked* LSTM da segunda abordagem.

Class	**PCE**	***Accuracy***	***Precison***	***Recall***	**F1-*score***
Slow	2.54%	97.45%	97%	97%	97%
Normal	3.54%	96.45%	97%	96%	97%
Aggressive	**2.47%**	**97.57%**	**98%**	**98%**	**98%**

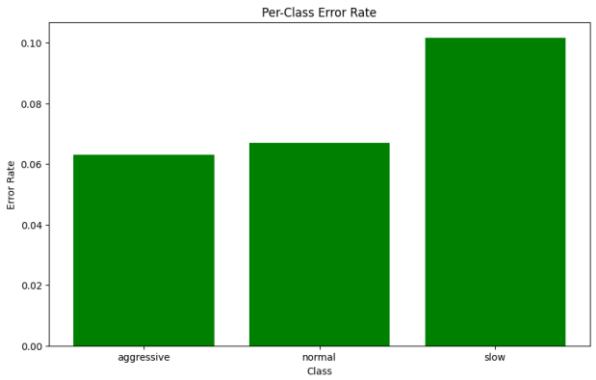

Figura 47 - Erro por classe - *Stacked* LSTM da segunda abordagem.

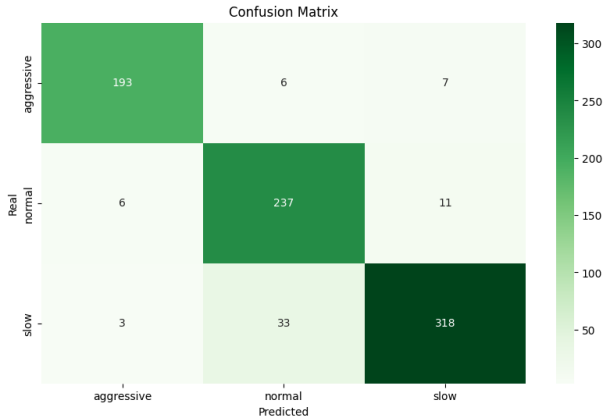

Figura 48 - Matriz de Confusão - *Stacked* LSTM da segunda abordagem.

6.3.2. Resultados da arquitetura *Bidirectional* LSTM

Nesta subseção, são apresentados os resultados da arquitetura *Bidirectional* LSTM (ver Tabela 15).

Tabela 15 - Resultados por classe do modelo Bidirectional LSTM da segunda abordagem.

Class	PCE	*Accuracy*	*Precison*	*Recall*	F1-*score*
Slow	2.26%	97.74%	96%	98%	97%
Normal	3.94%	96.06%	97%	96%	96%
Aggressive	4.85%	95.15%	98%	95%	96%

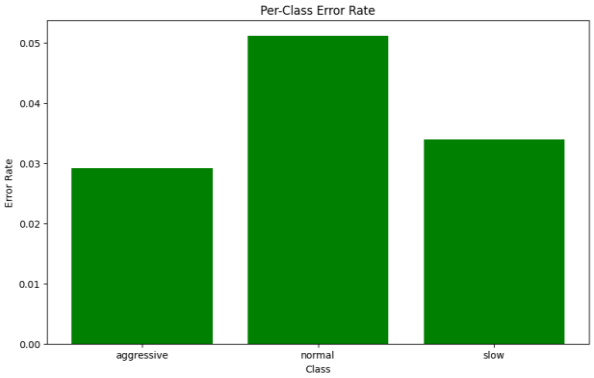

Figura 49 - Erro por classe - *Bidirectional* LSTM da segunda abordagem.

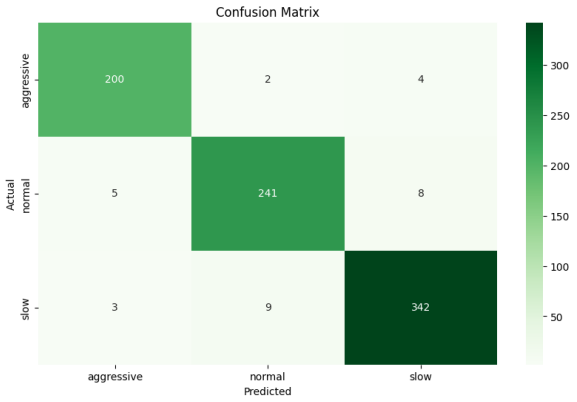

Figura 50 -Matriz de Confusão - *Bidirectional* LSTM da segunda abordagem.

6.3.3. Análise de resultados

A análise dos resultados obtidos para as duas arquiteturas testadas, *Stacked* LSTM e *Bidirectional* LSTM (BiLSTM), revela *insights* importantes sobre o desempenho desses modelos na tarefa de classificação multiclasse de estilos de condução. Segue-se uma discussão detalhada dos resultados, comparando as métricas de desempenho de ambas as abordagens.

6.3.3.1. Desempenho Geral

Ao observar as métricas de *precision*, *recall* **e** *F1-score* nos conjuntos de treino e teste, ambas as arquiteturas apresentam um desempenho robusto, com *accuracy* acima de 96% nos

testes. O BiLSTM apresentou uma ligeira vantagem em termos de *accuracy* no conjunto de teste (96.56% contra 96.32% da Stacked LSTM), enquanto a *Stacked* LSTM teve um desempenho marginalmente melhor em termos de F1-*score* (97.2% contra 96.51% do BiLSTM).

- *Accuracy* de Treino e Teste:
  - O modelo BiLSTM apresentou uma *accuracy* de treino significativamente mais alta (99.62% comparado a 97.64% da *Stacked* LSTM). No entanto, a diferença de *accuracy* no conjunto de teste foi mínima, sugerindo que ambos os modelos são comparavelmente eficazes na generalização para novos dados.
- *Loss*:
  - O BiLSTM também apresentou uma *loss* de treino menor (0.0209 contra 0.0656), o que reforça a hipótese de que o BiLSTM pode ter se ajustado melhor aos dados de treino. No entanto, as *losses* no conjunto de teste são praticamente equivalentes, com 0.0839 para o BiLSTM e 0.0856 para a *Stacked* LSTM, indicando que ambos os modelos têm desempenho semelhante na predição em dados não vistos.

6.3.3.2. Análise das Métricas por Classe

As métricas de desempenho por classe (Tabela 14 e Tabela 15) mostram que ambas as arquiteturas mantêm um desempenho consistente entre as diferentes classes.

***Stacked* LSTM:**

- As métricas de *precision*, *recall* e *F1-score* estão consistentemente em torno de 97% a 98% para todas as classes. A classe "Aggressive" apresentou o melhor desempenho, com uma *precision*, *recall* e *F1-score* de 98%.
- O modelo apresentou um erro por classe relativamente baixo em todas as classes, com valores variando entre 2.47% e 3.54%. A classe "Normal" teve o maior erro (3.54%), o que sugere que este padrão de condução é ligeiramente mais difícil de classificar corretamente para o modelo.

Bidirectional LSTM:

- A classe "Slow" foi onde o BiLSTM teve o seu melhor desempenho, com um *recall* de 98% e um PCE de 2.26%. A classe "Aggressive", no entanto, teve um desempenho inferior em relação ao Stacked LSTM, com um PCE mais alto (4.85%) e *recall* de 95%.
- Este modelo apresentou um PCE mais elevado para a classe "Aggressive" (4.85%), o que sugere uma maior dificuldade em identificar corretamente este

estilo de condução. Por outro lado, teve um PCE ligeiramente menor para a classe "Slow" (2.26%), indicando que é mais eficaz na deteção de condução lenta.

6.3.4. Considerações Finais

Ambas as arquiteturas apresentam resultados excelentes para a tarefa de classificação de estilos de condução. O *Bidirectional* LSTM destacou-se por ter uma menor *loss* durante o treino e uma maior *accuracy* global, o que pode indicar uma maior capacidade de aprendizagem e generalização. No entanto, a *Stacked* LSTM apresentou um *F1-score* ligeiramente superior, o que sugere que ela pode ser mais equilibrada na classificação entre as diferentes classes.

As diferenças entre os dois modelos são pequenas, o que significa que qualquer um deles pode ser escolhido com base em necessidades ou preferências específicas. Ambos apresentam um desempenho robusto e consistente, sendo que tanto o *Bidirectional* LSTM quanto a *Stacked* LSTM são opções viáveis para a tarefa de classificação de estilos de condução.

6.4. Ambiente de testes

Para a realização dos testes, utilizou-se a seguinte configuração de hardware e software:

Configuração de sistema:

- Processador: Intel Core i5 4690k
- Placa de Vídeo: NVIDIA GTX 1060 6GB
- RAM: 16GB DDR3

Versão do Software Utilizado:

- Sistema Operativo: Ubuntu 22.04
- Python: Versão 3.11.4
    - tensorflow: 2.15.0
    - keras: 2.15.0
- CUDA: 12.4
- cuDNN: 8.9.4.25

7. Artigo Científico

Para complementar o nosso projeto, foi-nos proposto pelos orientadores redigirmos um artigo científico para documentar e compartilhar a nossa abordagem e os resultados que obtemos com a implementação. Aceitando esta proposta, elaboramos o artigo intitulado de "An LSTM-Based Approach for Driving Behaviour Classification". Preparamos este artigo com o intuito de fornecer uma visão abrangente da nossa abordagem ao problema, destacando a preparação dos dados e a criação e treino dos modelos.

Acreditamos que este artigo não só documenta o processo de desenvolvimento, mas também contribui para a literatura existente sobre o tema uma vez se diferenciando de técnicas e métodos utilizados noutros trabalhos, mais concretamente na preparação dos dados. O artigo completo está disponível em anexo:

Artigo_Projeto_Informatico_194_LSTM.pdf

Anexo 1 - Artigo desenvolvido

8. Conclusão e Trabalho Futuro

Concluindo, consideramos que os objetivos delineamos para o desenvolvimento deste projeto foram cumpridos através da criação e treino de várias arquiteturas LSTM para fins comparativos. As arquiteturas desenvolvidas mostraram-se extremamente eficazes na análise e classificação das sequências de dados provenientes dos sensores utilizados para capturar os dados do *dataset.* A investigação focou-se principalmente no desenvolvimento de duas abordagens diferentes capazes de analisar e classificar diferentes padrões de condução tais como acelerações, travagens e curvas.

O uso das RNN, em particular a sua variante LSTM permitiu capturar de forma eficiente as dependências temporais presentes nos dados resultando em modelos capazes de identificar e classificar diferentes tipos de condução com um elevado grau de precisão o que se traduz numa implementação bem-sucedida.

Adicionalmente, o projeto proporcionou uma contribuição para o estado da arte na classificação de padrões de condução através de IA produzindo conhecimento que poderá servir de base para futuras investigações. A metodologia adotada, abarcou desde a análise ao problema proposto até ao desenvolvimento de validação dos modelos construídos, assegurando a robustez e fiabilidade dos resultados obtidos.

Por fim, consideramos que os resultados obtidos demonstram não só a eficácia das técnicas aplicadas como também o potencial para uma aplicação pratica da solução desenvolvida em contextos reais, tais como a monitorização de condutores, a análise comportamental para seguradoras ou até mesmo o desenvolvimento de sistemas de condução autónoma.

8.1. Trabalho Futuro

Apesar dos resultados alcançados, o projeto abre portas para um leque de oportunidades servindo como base para outros projetos futuros, que poderão ainda mais potenciar a aplicação da solução desenvolvida entre eles:

- Expansão do *dataset*
    - Uma expansão e diversificação dos dados utilizados para alimentar o modelo poderá aumentar a capacidade de generalização do mesmo e assim uma permitir ter uma maior adaptabilidade a diferentes contextos.
- Integração com Tecnologias de Veículos Autónomos
    - Realizar uma integração utilizando veículos autónomos analisando assim implementação da solução em contextos reais.
- Estudos de Impacto e Usabilidade
    - A realização de estudos sobre o impacto e usabilidade da solução desenvolvida, analisando a sua eficácia na redução de comportamentos imprudentes e promovendo comportamentos seguros.
- Aprimoramento dos Modelos de IA
    - Investir no aprimoramento dos modelos utilizando outras técnicas e abordagens que possam oferecer outro tipo de ganhos em termos de eficiência computacional e precisão.

Bibliografia

[1] B. Vieira e P. Aparício, “Aplicação para Rastreio de Viaturas,” Leiria, 2023.

[2] R. Anyoha, “The History of Artificial Intelligence,” 28 August 2017. [Online]. Available: https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/.

[3] S. A. S. A. Alowais, “Revolutionizing healthcare: the role of artificial intelligence in clinical practice,” *BMC Med Educ*, 2023.

[4] S. S. S. Divya Garikapati, “Autonomous Vehicles: Evolution of Artificial Intelligence and the Current Industry Landscape,” *MDPI*, 2024.

[5] IBM, “What is ML?,” [Online]. Available: https://www.ibm.com/topics/machine-learning.

[6] IBM, “ What is deep learning?,” [Online]. Available: https://www.ibm.com/topics/deep-learning.

[7] Nvidia, “Deep Learning,” [Online]. Available: https://developer.nvidia.com/deep-learning.

[8] IBM, “O que é uma rede neural?,” [Online]. Available: https://www.ibm.com/br-pt/topics/neural-networks.

[9] S. Teles, “Processamento de Linguagem Natural com Deep Learning: Uma nova Era,” 4 August 2020. [Online]. Available: https://medium.com/data-hackers/a-revolu%C3%A7%C3%A3o-no-processamento-de-linguagem-natural-com-deep-learning-eba175f64c01.

[10] M. Ramos, “O que é Redes Neurais (Neural Networks)? - Glossário de Automação,” [Online]. Available: https://glossario.maiconramos.com/glossario/o-que-e-redes-neurais-neural-networks/.

[11] AWS, “O que é RNN? — Explicação sobre redes neurais recorrentes,” [Online]. Available: https://aws.amazon.com/pt/what-is/recurrent-neural-network/.

[12] S. Ashwani, “A Gentle Introduction to Recurrent Neural Networks(RNN),” 9 Junho 2022. [Online]. Available: https://swagata1506.medium.com/a-gentle-introduction-to-recurrent-neural-networks-rnn-54d223598105.

[13] M. B. Durna, “Neural Networks in NLP: RNN, LSTM, and GRU,” 12 Janeiro 2024. [Online]. Available: https://medium.com/@mervebdurna/nlp-with-deep-learning-neural-networks-rnns-lstms-and-gru-3de7289bb4f8.

[14] A. Sharma, “AI In Finance: Time Series Forecasting with Recurrent Neural Networks,” 22 Janeiro 2024. [Online]. Available: https://www.signitysolutions.com/tech-insights/time-series-forecasting-with-rnn.

[15] P. Baheti, “The Complete Guide to Recurrent Neural Networks,” 29 Julho 2022. [Online]. Available: https://www.v7labs.com/blog/recurrent-neural-networks-guide.

[16] J. S. Sepp Hochreiter, “Long Short-term Memory,” *Neural computation,* vol. 9, pp. 1735-80, 1997.

[17] maryangelabermudez, “ Nessa rede de deep learning a saída de um neurônio é aplicada como entrada no próprio neurônio,” [Online]. Available: https://brainly.com.br/tarefa/56827023.

[18] GeeksforGeeks, “What is LSTM - Long Short Term Memory?,” [Online]. Available: https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/.

[19] Á. G. A. F. d. C. C. W. d. S. &. F. M. Z. Modesto, “ Mapeamento tecnológico da aplicação de redes neurais para eficiência energética de sistemas de bombeamento.,” *Revista de Tecnologia Aplicada,* p. 82–95, 2023.

[20] C. V. Nicholson, “A Beginner's Guide to LSTMs and Recurrent Neural Networks,” [Online]. Available: https://wiki.pathmind.com/lstm.

[21] C. Olah, “Understanding LSTM Networks,” 27 Agosto 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[22] shivammiglani09, “Bidirectional Recurrent Neural Network,” 8 Junho 2023. [Online]. Available: https://www.geeksforgeeks.org/bidirectional-recurrent-neural-network/.

[23] I. K. Ihianle, A. O. Nwajana, S. H. Ebenuwa, R. I. Otuka, K. Owa e M. O. Orisatoki, “A deep learning approach for human activities recognition from multimodal sensing devices,” 2020. [Online].

[24] Y. a. L. Y. Bengio, “Convolutional Networks for Images, Speech, and Time-Series,” 1997.

[25] M. a. P. P. D. a. S. J. a. D. B. a. A. L. a. O. L. Ferreira, “Avaliação do Uso Redes Neurais Convolucionais para Identificação de Lesões Cariosas Dentárias,” pp. 473-478, 2023.

[26] avichalbharti, “What is a 1D Convolutional Layer in Deep Learning,” 19 Fevereiro 2024. [Online]. Available: https://www.geeksforgeeks.org/what-is-a-1d-convolutional-layer-in-deep-learning/.

[27] L. C. Chen, J. T. Sheu, Y. J. Chuang e Y. Tsao, “Predicting the Travel Distance of Patients to Access Healthcare Using Deep Neural Networks,” 2022. [Online]. Available: /pmc/articles/PMC8809644/.

[28] IBM, “How do convolutional neural networks work?,” [Online]. Available: https://www.ibm.com/topics/convolutional-neural-networks.

[29] Y. a. Y. C. T. a. K. S. a. T. S. K. a. C. C. a. Y. T. a. A. A. a. A. K. a. R. A. Mohammed Alsumaidaee, “Detection of Corona Faults in Switchgear by Using 1D-CNN, LSTM, and 1D-CNN-LSTM Methods,” *Sensors*, vol. 23, 2023.

[30] AWS, “O que é Python? – Explicação sobre a linguagem Python,” [Online]. Available: https://aws.amazon.com/pt/what-is/python/.

[31] GeeksforGeeks, “What is Python? it’s Uses and Applications,” [Online]. Available: https://www.geeksforgeeks.org/what-is-python/.

[32] Python, “os - Python documentation,” [Online]. Available: https://docs.python.org/3/library/os.html.

[33] Python, “math - Python documentation,” [Online]. Available: https://docs.python.org/3/library/math.html.

[34] NumPy, “NumPy documentation,” [Online]. Available: https://numpy.org/doc/stable/.

[35] pandas, “pandas documentation,” [Online]. Available: https://pandas.pydata.org/docs/index.html.

[36] Matplotlib, “Matplotlib documentation,” [Online]. Available: https://matplotlib.org/stable/index.html.

[37] seaborn, “seaborn documentation,” [Online]. Available: https://seaborn.pydata.org/.

[38] TensorFlow, “About TensorFlow,” [Online]. Available: https://www.tensorflow.org/about.

[39] Keras, “About Keras,” [Online]. Available: https://keras.io/about/.

[40] scikit-learn, “scikit-learn documentation,” [Online]. Available: https://scikit-learn.org/stable/.

[41] Folium, “Folium documentation,” [Online]. Available: https://python-visualization.github.io/folium/latest/.

[42] Lucidchart, “O que é diagramação inteligente?,” [Online]. Available: https://www.lucidchart.com/blog/pt/diagramas-inteligentes.

[43] S. d. Microsoft, “Introdução ao Microsoft Teams,” [Online]. Available: https://support.microsoft.com/pt-pt/office/introdu%C3%A7%C3%A3o-ao-microsoft-teams-b98d533f-118e-4bae-bf44-3df2470c2b12.

[44] Visual Studio Code , “Visual Studio Code Frequently Asked Questions,” [Online]. Available: https://code.visualstudio.com/Docs/supporting/faq.

[45] K. Saleh, M. Hossny e S. Nahavandi, “Driving behavior classification based on sensor data fusion using LSTM recurrent neural networks,” 7 2018. [Online].

[46] M. A. Khodairy e G. Abosamra, “Driving Behavior Classification Based on Oversampled Signals of Smartphone Embedded Sensors Using an Optimized Stacked-LSTM Neural Networks,” 2021. [Online].

[47] E. Carvalho, B. V. Ferreira, J. Ferreira, C. De Souza, H. V. Carvalho, Y. Suhara, A. S. Pentland e G. Pessin, “Exploiting the use of recurrent neural networks for driver behavior profiling,” 6 2017. [Online].

[48] M. Sokolova e G. Lapalme, A systematic analysis of performance measures for classification tasks, vol. 45, Pergamon, 2009, pp. 427-437.

Anexos