

**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



**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

Inserir aqui os agradecimentos. Trata-se de um elemento **facultativo**.

Texto dos agradecimentos. Texto dos agradecimentos. Texto dos agradecimentos. Texto dos agradecimentos. Texto dos agradecimentos. Texto dos agradecimentos. Texto dos agradecimentos. Texto dos agradecimentos. Texto dos agradecimentos. Texto dos agradecimentos. Texto dos agradecimentos.

Texto dos agradecimentos. Texto dos agradecimentos. Texto dos agradecimentos. Texto dos agradecimentos. Texto dos agradecimentos. Texto dos agradecimentos. Texto dos agradecimentos. Texto dos agradecimentos. Texto dos agradecimentos. Texto dos agradecimentos. Texto dos agradecimentos.

# Resumo

Este trabalho aborda a utilização de Inteligência Artificial, especificamente Redes Neurais do tipo *Long Short-Term Memory* (LSTM), para a classificação de comportamentos de condução. A motivação deste projeto surge com a necessidade de desenvolver métodos capazes de identificar e reagir a diferentes padrões de condução devido ao avanço dos sistemas de condução autónoma. O objetivo principal é desenvolver um modelo capaz de identificar padrões de condução e classificá-los utilizando dados de sensores de dispositivos móveis.

A metodologia empregue inclui a aquisição e pré-processamento de dados de sensores, o desenvolvimento e treino de um modelo de Redes Neurais Recorrentes LSTM, e a validação e avaliação do desempenho do modelo.

Os resultados mostram que o modelo desenvolvido consegue classificar de forma eficaz os diferentes tipos de comportamento de condução, apresentando uma alta taxa de acurácia. As conclusões indicam que a aplicação de IA na classificação de comportamentos de condução pode contribuir significativamente para a segurança viária e a otimização de sistemas de condução autônoma.

**Palavras-chave:** Inteligência Artificial, Redes Neuronais, LSTM, RNN, Condução, Classificação

# Abstract

This work addresses the use of Artificial Intelligence, specifically Long Short-Term Memory (LSTM) Neural Networks, for the classification of driving behaviors. The main objective is to develop a model capable of identifying aggressive and non-aggressive driving patterns using data from mobile device sensors. The intention of this project arises from the need to develop methods capable of identifying and reacting to different driving patterns due to the advancement of autonomous driving systems.

The methodology employed includes the acquisition and preprocessing of sensor data, the development and training of an LSTM Recurrent Neural Network model, and the validation and evaluation of the model's performance.

The results show that the developed model can effectively classify different types of driving behavior, presenting a high accuracy rate. The conclusions indicate that the application of AI in driving behavior classification can significantly contribute to road safety and the optimization of autonomous driving systems.

**Keywords:** Artificial Intelligence, Neural Networks, LSTM, RNN, Driving, Classification

Índice

[Dedicatória ii](#_Toc175166722)

[Agradecimentos iii](#_Toc175166723)

[Resumo iv](#_Toc175166724)

[Abstract v](#_Toc175166725)

[Lista de Figuras x](#_Toc175166726)

[Lista de Listagens xii](#_Toc175166727)

[Lista de Tabelas xiii](#_Toc175166728)

[Lista de siglas e acrónimos xiv](#_Toc175166729)

[1. Introdução 1](#_Toc175166730)

[1.1. Estrutura do trabalho 1](#_Toc175166731)

[1.2. Metodologia De Trabalho 3](#_Toc175166732)

[1.2.1. Reuniões 3](#_Toc175166733)

[1.2.2. Desenvolvimento 3](#_Toc175166734)

[2. Fundamentação Científica 4](#_Toc175166735)

[2.1. Inteligência Artificial 4](#_Toc175166736)

[2.2. Redes Neuronais (RN) 5](#_Toc175166737)

[2.3. Redes Neurais Recorrentes (RNN) 5](#_Toc175166738)

[2.3.1. Tipos de RNN 6](#_Toc175166739)

[2.3.2. Estrutura e Funcionamento das RNN 7](#_Toc175166740)

[2.3.3. Vantagens e Limitações das RNN 7](#_Toc175166741)

[2.4. Long Short-Term Memory 10](#_Toc175166742)

[2.4.1. Arquitetura de uma LSTM: Estrutura e Funcionamento 10](#_Toc175166743)

[2.4.2. Parâmetros comuns de LSTM 13](#_Toc175166744)

[2.5. LSTM VS RNN 14](#_Toc175166745)

[2.6. Arquitetura Bidirecional 15](#_Toc175166746)

[2.6.1. Estrutura e Funcionamento 15](#_Toc175166747)

[2.6.2. Vantagens e Desvantangens 16](#_Toc175166748)

[2.7. Redes Neuronais Convolucionais 16](#_Toc175166749)

[2.8. Redes Convolucionais Unidimensionais 17](#_Toc175166750)

[2.8.1. Estrutura e Funcionamento 17](#_Toc175166751)

[2.8.2. Vantagens e Desvantagens 18](#_Toc175166752)

[2.9. Análise de Soluções 19](#_Toc175166753)

[3. Tecnologias e Ferramentas Utilizadas 20](#_Toc175166754)

[3.1.1. Linguagem *Python* 20](#_Toc175166755)

[3.1.2. Bibliotecas 20](#_Toc175166756)

[3.1.3. Plataformas 23](#_Toc175166757)

[3.2. Trabalhos Relacionados 24](#_Toc175166758)

[3.2.1. Driving Behavior Classification Based on Sensor Data Fusion Using LSTM Recurrent Neural Networks 24](#_Toc175166759)

[3.2.2. Driving Behavior Classification Based on Oversampled Signals of Smartphone Embedded Sensors Using an Optimized Stacked-LSTM Neural Networks 26](#_Toc175166760)

[4. Análise 30](#_Toc175166761)

[4.1. Análise dos sensores de dispositivos móveis 30](#_Toc175166762)

[4.1.1. Definição e Função 30](#_Toc175166763)

[4.1.2. Tipos de Sensores Utilizados 30](#_Toc175166764)

[4.2. Análise da Aplicação de Recolha dos Dados 31](#_Toc175166765)

[4.3. Análise dos datasets 31](#_Toc175166766)

[4.3.1. IPL-Dataset 31](#_Toc175166767)

[4.3.2. UAH*-DRIVESET* 32](#_Toc175166768)

[5. Requisitos 33](#_Toc175166769)

[5.1. Requisitos Funcionais 33](#_Toc175166770)

[5.1.1. Aquisição de Dados 33](#_Toc175166771)

[5.1.2. Processamento de Dados 33](#_Toc175166772)

[5.1.3. Treino do Modelo 33](#_Toc175166773)

[5.1.4. Classificação 33](#_Toc175166774)

[5.1.5. Relatórios e Análises 34](#_Toc175166775)

[5.2. Requisitos Não Funcionais 34](#_Toc175166776)

[5.2.1. Desempenho 34](#_Toc175166777)

[5.2.2. Escalabilidade 34](#_Toc175166778)

[5.2.3. Manutenibilidade 34](#_Toc175166779)

[5.3. Arquitetura da Solução 34](#_Toc175166780)

[6. Arquiteturas Propostas 37](#_Toc175166781)

[6.1. *Stacked* LSTM 37](#_Toc175166782)

[6.1.1. Introdução a *Stacked* LSTM 37](#_Toc175166783)

[6.1.2. Estrutura da Stacked LSTM 37](#_Toc175166784)

[6.2. Bidirectional LSTM 38](#_Toc175166785)

[6.2.1. Introdução a Bidirectional LSTM 38](#_Toc175166786)

[6.2.2. Estrutura da Bidirectional LSTM 38](#_Toc175166787)

[6.3. Convolutional LSTM 38](#_Toc175166788)

[6.3.1. Introdução a Convolutional LSTM 38](#_Toc175166789)

[6.3.2. Estrutura da Convolutional LSTM 39](#_Toc175166790)

[7. Desenvolvimento 40](#_Toc175166791)

[7.1. Implementação 40](#_Toc175166792)

[7.2. Primeira Abordagem - Classificação de Binária 41](#_Toc175166793)

[7.2.1. Descrição e Caracterização dos Dados 41](#_Toc175166794)

[7.2.2. Tratamento dos Dados 41](#_Toc175166795)

[7.2.3. Classificação dos Dados 44](#_Toc175166796)

[7.2.4. Normalização dos Dados 46](#_Toc175166797)

[7.2.5. Representação Visual dos Dados 49](#_Toc175166798)

[7.2.6. Separação dos Dados em Treino e Teste 50](#_Toc175166799)

[7.2.7. Criação dos Modelos 51](#_Toc175166800)

[7.2.8. Compilação e Treino 56](#_Toc175166801)

[7.3. Segunda Abordagem – Classificação de Classes 58](#_Toc175166802)

[7.3.1. Descrição e Caracterização dos Dados 58](#_Toc175166803)

[7.3.2. Tratamento dos Dados 58](#_Toc175166804)

[7.3.3. Modelos 62](#_Toc175166805)

[8. Testes e Resultados 64](#_Toc175166806)

[8.1. Objetivos dos Testes 64](#_Toc175166807)

[8.2. Métricas de Avaliação 64](#_Toc175166808)

[8.2.1. Acurácia 64](#_Toc175166809)

[8.2.2. Precisão 64](#_Toc175166810)

[8.2.3. Recall (Sensibilidade) 65](#_Toc175166811)

[8.2.4. F1-Score 65](#_Toc175166812)

[8.2.5. Jaccard Score 65](#_Toc175166813)

[8.2.6. Hamming Loss 65](#_Toc175166814)

[8.3. Resultados 66](#_Toc175166815)

[8.3.1. Primeira Abordagem - Classificação de Binária 66](#_Toc175166816)

[8.3.1. Segunda Abordagem - Classificação de Classes 73](#_Toc175166817)

[8.4. Considerações Finais 76](#_Toc175166818)

[9. Artigo Científico 77](#_Toc175166819)

[9.1. Objetivos do Artigo 77](#_Toc175166820)

[9.2. Metodologia 77](#_Toc175166821)

[9.3. Resultados e Discussão 77](#_Toc175166822)

[9.4. Conclusões e Contributos Científicos 77](#_Toc175166823)

[9.5. Considerações Finais 77](#_Toc175166824)

[10. Conclusão e Trabalho Futuro 78](#_Toc175166825)

[Bibliografia 79](#_Toc175166826)

[Anexos 83](#_Toc175166827)

[Glossário 84](#_Toc175166828)

# Lista de Figuras

[Figura 1 - Diagrama Ilustrativo de uma RNN 6](#_Toc175166829)

[Figura 2 - Diagrama Ilustrativo de uma LSTM 10](#_Toc175166830)

[Figura 3 - Diagrama Ilustrativo da Forget Gate 11](#_Toc175166831)

[Figura 4 - Diagrama Ilustrativo da Input Gate 12](#_Toc175166832)

[Figura 5 - Diagrama Ilustrativo da Output Gate 13](#_Toc175166833)

[Figura 6 - Diagrama Representativo de uma LSTM Bidirecional. FONTE: A Deep Learning Approach for Human Activities Recognition From Multimodal Sensing Devices (2020) 16](#_Toc175166834)

[Figura 7 - Estágios e Camadas de uma Rede Neural Convolucional. FONTE: Adaptado de Kategaru (2020) 17](#_Toc175166835)

[Figura 8 - Diagrama Ilustrativo de uma Implementação CNN 1D usando Max Pooling. FONTE: Predicting the Travel Distance of Patients to Access Healthcare using Deep Neural Networks (2021) 18](#_Toc175166836)

[Figura 9 - Software Utilizado para Capturar os Dados do UAH-driveset. FONTE: Driving behavior classification based on sensor data fusion using LSTM recurrent neural networks (2017) 25](#_Toc175166837)

[Figura 10 - Gráfico Ilustrativo do Resampling dos Dados. FONTE: Driving behavior classification based on sensor data fusion using LSTM recurrent neural networks (2017) 26](#_Toc175166838)

[Figura 11 - Resultados da Matriz de Confusão da Implementação Proposta em Comparação com a Classificação de Comportamento de Condução Descrita pelo UAH-DriveSet. FONTE: Driving behavior classification based on sensor data fusion using LSTM recurrent neural networks (2017) 26](#_Toc175166839)

[Figura 12 - Arquitetura do protótipo 35](#_Toc175166840)

[Figura 13 - Diagrama Ilustrativo do Tratamento dos Dados 43](#_Toc175166841)

[Figura 14 - Diagrama ilustrativo da Classificação dos Dados 45](#_Toc175166842)

[**Figura 15 - Diagrama ilustrativo da Função *Max Of Vectors* Aplicada ao Acelerómetro** 47](#_Toc175166843)

[Figura 16 - Diagrama ilustrativo da Função *Max Of Vectors* Aplicada ao Giroscópio 47](#_Toc175166844)

[Figura 17 - Diagrama ilustrativo da Normalização dos Dados 48](#_Toc175166845)

[Figura 18 - Gráfico de manobras do dataset 49](#_Toc175166846)

[Figura 19 - Gráfico de manobras, com zoom, do dataset 49](#_Toc175166847)

[Figura 20 - Diagrama Ilustrativo da Separação dos Dados 51](#_Toc175166848)

[Figura 21 - Diagrama ilustrativo da estrutura do modelo Stacked LSTM 52](#_Toc175166849)

[Figura 22 - Diagrama ilustrativo da estrutura do modelo Bidirectional LSTM 54](#_Toc175166850)

[Figura 23 - Diagrama ilustrativo da estrutura do modelo ConvolutionalLSTM 56](#_Toc175166851)

[Figura 24 - Etapas de processamento de dados (Segunda Abordagem) 59](#_Toc175166852)

[Figura 26 - Comparação da loss e val\_loss para o modelo Convolutional LSTM Primeira Abordagem 67](#_Toc175166853)

[Figura 27 - Comparação dos valores reais vs valores previstos pelo modelo Convolutional LSTM Primeira Abordagem 68](#_Toc175166854)

[Figura 28 - Comparação da loss e val\_loss para o modelo Stacked LSTM Primeira Abordagem 69](#_Toc175166855)

[Figura 29 - Comparação dos valores reais vs valores previstos pelo modelo Stacked LSTM Primeira Abordagem 70](#_Toc175166856)

[Figura 30 - Comparação da loss e val\_loss para o modelo Bidirecional LSTM Primeira Abordagem 71](#_Toc175166857)

[Figura 31 - Comparação dos valores reais vs valores previstos pelo modelo Bidirecional LSTM Primeira Abordagem 72](#_Toc175166858)

[Figura 32 - Erro por classe Stacked LSTM (Segunda abordagem) 74](#_Toc175166859)

[Figura 33 - Matriz de Confusão Stacked LSTM (Segunda abordagem) 74](#_Toc175166860)

[Figura 34 - Erro por classe Bidirecional LSTM (Segunda abordagem) 75](#_Toc175166861)

[Figura 35 -Matriz de Confusão Bidirecional LSTM (Segunda abordagem) 76](#_Toc175166862)

# Lista de Listagens

[Listagem 1 - Função *separate\_positives\_negatives* 42](#_Toc175166863)

[Listagem 2 - Função y\_classification 45](#_Toc175166864)

[Listagem 3 - Função max\_of\_vectors 46](#_Toc175166865)

[Listagem 4 - Função normalize\_between\_0\_and\_max\_v2 46](#_Toc175166866)

[Listagem 5 - Função split\_train\_test 50](#_Toc175166867)

[Listagem 6 - Criação do Modelo Stacked LSTM Primeira Abordagem 52](#_Toc175166868)

[Listagem 7 - Criação do Modelo Bidirectional LSTM Primeira Abordagem 53](#_Toc175166869)

[Listagem 8 - Criação do Modelo Convolutional LSTM Primeira Abordagem 55](#_Toc175166870)

[Listagem 9 - Compilação do Modelo Stacked LSTM Primeira Abordagem 56](#_Toc175166871)

[Listagem 10 - Treino do Modelo Stacked LSTM Primeira Abordagem 56](#_Toc175166872)

[Listagem 11 - Compilação do Modelo Bidirectional LSTM Primeira Abordagem 57](#_Toc175166873)

[Listagem 12 - Treino do Modelo Bidirectional LSTM Primeira Abordagem 57](#_Toc175166874)

[Listagem 13 - Compilação do Modelo Convolutional LSTM Primeira Abordagem 57](#_Toc175166875)

[Listagem 14 - Treino do Modelo Convolutional LSTM Primeira Abordagem 57](#_Toc175166876)

# Lista de Tabelas

[Tabela 1 - Tabela representativa das reuniões e dos tópicos abordados 3](#_Toc175166877)

[Tabela 2 - Tabela Comparativa entre LSTM e RNN 14](#_Toc175166878)

[Tabela 3 - Tabela comparativa entre as diferentes arquiteturas 19](#_Toc175166879)

[Tabela 4 - Glossário de Cores Utilizadas nos Diagramas Ilustrativos 40](#_Toc175166880)

[Tabela 5 - Dados antes de aplicar Rolling Window 60](#_Toc175166881)

[Tabela 6 - Dados depois de aplicar Rolling Window 60](#_Toc175166882)

[Tabela 7 - Comparação de métricas entre os modelos Primeira Abordagem 66](#_Toc175166883)

[Tabela 8 - Comparação de resultados dos modelos da Segunda abordagem 73](#_Toc175166884)

[Tabela 9 - Resultados por classe do modelo StackedLSTM (Segunda abordagem) 73](#_Toc175166885)

[Tabela 10 - Resultados por classe do modelo Bidirecional LSTM (Segunda abordagem) 75](#_Toc175166886)

# Lista de siglas e acrónimos

|  |  |
| --- | --- |
| CNN | Convolutional Neural Networks |
| ESTG | Escola Superior de Tecnologia e Gestão |
| IA | Inteligência Artificial |
| LSTM | Long Short-Term Memory |
| LLM | Large Language Model |
| JSON | JavaScript Object Notation |
| RNN | Recurrent Neural Network |
|  |  |
|  |  |

# Introdução

O presente projeto, intitulado “AI Driving Classification”, propõe-se a explorar, analisar e classificar a condução de condutores através de uma rede neuronal. Este projeto surge da necessidade de compreender os padrões de comportamento ao volante e as implicações que têm na segurança rodoviária.

O objetivo central deste projeto é a classificação da condução com base em dados adquiridos através de uma aplicação. Por meio da análise destes dados, procura-se identificar padrões de comportamento do condutor, tais como tipo de aceleração, tipo de travagem, entre outros indicadores relevantes para a classificação.

A análise da condução é de extrema importância em diversos contextos, desde a segurança rodoviária até ao desenvolvimento de sistemas de assistência à condução. Compreender os padrões de condução pode contribuir significativamente para a prevenção de acidentes rodoviários.

Os objetivos gerais deste projeto consistem em desenvolver um modelo de classificação de condução o mais preciso e confiável possível, capaz de identificar diferentes estilos e padrões direção.

## Estrutura do trabalho

O relatório está organizado em 13 capítulos, cada um abordando uma parte fundamental no desenvolvimento deste projeto. Os capítulos seguem a seguinte ordem:

* Capítulo 1 – Introdução: Apresenta os objetivos e a estrutura do trabalho.
* Capítulo 2 – Metodologia De Trabalho: Descreve como o projeto foi desenvolvido e as reuniões semanais com os orientadores.
* Capítulo 3 – Fundamentação Científica: Aborda os conceitos relacionadas com o projeto.
* Capítulo 4 – Estado da Arte: Aborda as tecnologias relacionadas com o projeto.
* Capítulo 5 – Trabalhos Relacionados: Aqui fazemos a revisão da literatura existente sobre o tema do projeto.
* Capítulo 6 – Análise: Análise aos *datasets* e aplicação de recolha de dados utilizados no desenvolvimento deste projeto.
* Capítulo 7 – Planeamento: Descrição do planeamento e dos requisitos do projeto.
* Capítulo 8 – Arquiteturas Propostas: Análise detalhada das arquiteturas propostas.
* Capítulo 9 – Desenvolvimento: Descrição detalhada do processo de desenvolvimento do projeto.
* Capítulo 8 – Testes e Resultados: Análise e discussão dos resultados obtidos.
* Capítulo 9 – Artigo Científico: Aborda o artigo realizado no âmbito do projeto.
* Capítulo 10 – Conclusões e Trabalho Futuro: Conclusão do trabalho realizado, abordando o c

## Metodologia De Trabalho

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.

### 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 \_\_ reuniões no total, tendo como tópicos abordados os seguintes:

Tabela 1 - Tabela representativa das reuniões e dos tópicos abordados

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Fundamentação Científica

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

## Inteligência Artificial

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, somente 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.

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.

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 neurais profundas e algoritmos de visão computacional, possibilitam que os veículos autônomos identifiquem sinais de trânsito, pedestres, outros veículos e obstáculos, agindo de forma similar a um ser humano 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.

## Redes Neuronais (RN)

As redes neurais representam uma abordagem poderosa e versátil no campo da IA, inspirada no funcionamento do cérebro humano.

Redes neurais, ou sistemas neurais 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 neural.

A arquitetura de uma rede neural geralmente inclui uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída [1]. 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 neurais, cada uma com arquiteturas e aplicações específicas [2].

As redes neurais 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 diagnóstico médico.

Apesar dos avanços significativos, as redes neurais ainda enfrentam desafios como o sobreajuste, a interpretabilidade e a eficiência computacional. No entanto, com o desenvolvimento contínuo de novas arquiteturas e técnicas de treino, as redes neurais estão cada vez mais a tornar-se uma ferramenta poderosa para resolver problemas complexos em diversas áreas.

## Redes Neurais Recorrentes (RNN)

Uma rede neural 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 1) 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 [3]. Cada vez mais as RNN estão a ser substituídas por IA e LLM, que são muito mais eficientes no processamento sequencial de dados.

|  |
| --- |
| Uma imagem com diagrama, file, círculo  Descrição gerada automaticamente  Figura 1 - Diagrama ilustrativo de uma RNN |

### Tipos de RNN

* ***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 [3].

* ***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.**

### 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:

* ​ é o estado oculto no tempo *t*.
* é a entrada no tempo *t*.
* ​ e ​ são pesos de entrada e recorrentes, respetivamente.
* ​ é o *bias* do estado oculto.
* é a saída no tempo *t*.
* é o peso de saída.
* ​ é o *bias* da saída.
* e 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.

### Vantagens e Limitações das RNN

##### Vantagens

**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 uma etapa de cada vez, 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 manterem um 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.
  + **Previsão de Séries Temporais**: Previsão de vendas, análise financeira, previsão do tempo.
  + **Reconhecimento de Padrões**: Reconhecimento de fala, classificação de gestos em vídeos, deteção de anomalias.
* **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.

##### Limitações

**Problema do Gradiente Desaparece/Explode**

O problema do gradiente desaparece/explode 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 de retropropagação (*backpropagation*) é usado para ajustar os pesos da rede. Este processo envolve o cálculo do gradiente do erro em relação 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.
* *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.

**Solução para Resolver as Limitações**

Para resolver a dificuldade de capturar dependências de longo prazo, diversas abordagens e melhorias nas arquiteturas de redes neurais 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.

## Long Short-Term Memory

LSTM (ver Figura 2) é uma versão melhorada da RNN projetada por Hochreiter & Schmidhuber. O LSTM é adequado 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 é um *contêiner* 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 [4]. As LSTM também podem ser usadas em combinação com outras arquiteturas de rede neural, como Redes Neurais Convolucionais (CNN) para análise de imagens e vídeos.

|  |
| --- |
| A diagram of a flowchart  Description automatically generated  Figura 2 - Diagrama ilustrativo de uma LSTM |

### 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*.

##### *Forget Gate*

A informação que já não é útil no estado da célula é removida com a porta de esquecimento. Dois inputs, (entrada no tempo específico) e (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.

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:

onde:

* representa a matriz de pesos associada à *Forget Gate* (ver Figura 3) .
* denota a concatenação da entrada atual e do estado oculto anterior.
* é o bias com a *Forget Gate*.
* é a função de ativação sigmoide

|  |
| --- |
| A diagram of a diagram of a function  Description automatically generated  Figura 3 - Diagrama ilustrativo da *Forget Gate* |

##### *Input Gate*

A adição de informações úteis ao estado da célula é feita pela *Input Gate* (ver Figura 4) . Primeiro, a informação é regulada usando a função sigmoide que filtra os valores a serem lembrados usando as entradas e . Depois, um vetor é criado usando a função tanh que fornece uma saída de -1 a +1, que contém todos os valores possíveis de e [5]. Por fim, os valores do vetor e os valores regulados são multiplicados para obter as informações úteis. A equação para a *Input Gate* é:

Multiplicamos o estado anterior por ft não utilizando a informação que optamos anteriormente por ignorar. A seguir, incluímos it ∗ Ct. Isto representa os valores candidatos atualizados, ajustados pelo valor que escolhemos para atualizar cada *state value*.

onde:

* ⊙ denota multiplicação elemento a elemento.
* é a função de ativação tanh.

|  |
| --- |
| A diagram of a diagram of a flowchart  Description automatically generated  Figura 4 - 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 5) . Primeiro, um vetor é gerado aplicando a função na célula. Depois, a informação é regulada através da função sigmoide e filtrada pelos valores a serem lembrados através das entradas e Por fim, os valores do vetor e os valores regulados são multiplicados para serem enviados como *output* e *input* para a próxima célula. A equação para a *Output Gate* é:

|  |
| --- |
| A diagram of a flowchart  Description automatically generated  Figura 5 - Diagrama ilustrativo da *Output Gate* |

### Parâmetros comuns de LSTM

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

***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*.

## LSTM VS RNN

As RNN são redes neurais 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 possuem 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.

Tabela 2 - Tabela comparativa entre LSTM e RNN

|  |  |  |
| --- | --- | --- |
| **Funcionalidade** | **LSTM** | **RNN** |
| **Memoria** | Tem uma unidade de memória especifica que possibilita a aprendizagem de sequência de dados | Não tem uma 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 memoria | Mais fácil de treinar |
| ***Long-term dependency learning*** | Sim | Limitado |
| **Capacidade de aprender dados sequenciais** | Sim | Sim |

## Arquitetura Bidirecional

A arquitetura bidirecional tem sido amplamente escolhidas para desempenhar tarefas que requerem uma 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.

### Estrutura e Funcionamento

As LSTM Bidirecionais (ver Figura 6), 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.



Figura 6 - Diagrama Representativo de uma LSTM Bidirecional. FONTE: A Deep Learning Approach for Human Activities Recognition From Multimodal Sensing Devices (2020)

### Vantagens e Desvantangens

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 um impacto considerável na complexidade computacional por serem muito mais complexas que as LSTM tradicionais. Também tem uma maior quantidade de parâmetros o que poderá levar ao aumento do consumo de memoria. 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.

## Redes Neuronais Convolucionais

As *Convolutional Neural Network* (CNN) são um tipo de redes neuronais que foram propostas pelo pesquisador francês Yann Lecun que se destacam pela eficácia obtida na análise de dados que estejam no formato *grade****.*** Tal como o nome indica, *convolution*, significa uma função matemática que deriva da integração de outras duas funções completamente distintas. A arquitetura das CNN (ver Figura 7) deriva das *Artificial Neural Networks* (ANN)*.* Estes tipos de redes neuronais costumam trabalhar sobre imagens e sinais de áudio, mas, no entanto, também existem outras vertentes que aplicam este conceito a dados de texto, séries temporais de dados e sequências de dados.



Figura 7 - Estágios e Camadas de uma Rede Neural Convolucional. FONTE: Adaptado de Kategaru (2020)

## 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 sequencias unidimensionais tais como texto e sinais de áudio.

### Estrutura e Funcionamento

Uma CNN pode ser dividida em três etapas principais (ver Figura 8): *Convolution* com função de ativação, *Pooling* e *Fully-connected*.

Na primeira camada, *Convolution*, é processada a operação que realiza a convulção dos dados de entrada. Ira ser 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 que foi 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 que é 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 sequencias unidimensionais e o *polling* precisa duas dimensões, podemos fazer um *reshape* do *input shape* para solucionar este problema.



Figura 8 - Diagrama ilustrativo de uma implementação CNN 1D usando *Max Pooling*. FONTE: Predicting the Travel Distance of Patients to Access Healthcare using Deep Neural Networks (2021)

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.

### 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 e 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 reduzir, evitando assim um potencias *overfitting* da rede. Mas a principal vantagem que se destaca entre as outras é a capacidade desta arquitetura se combinar com outras, como LSTM, para que aprimorar os resultados fazendo com que a escalabilidade seja possível.

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 sequencias que seja claro, as Conv 1D não irão produzir resultados precisos.

## Análise de Soluções

A Tabela 3 apresenta um resumo comparativo das diferentes arquiteturas.

Tabela 3 - Tabela comparativa entre as diferentes arquiteturas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Característica** | **LSTM** | **RNN** | **Conv1D** | **Bi-LSTM** |
| **Memória** | Sim | Não | Não | Sim |
| **Direcionalidade** | Unidirecional /Bidirecional | Unidirecional | Unidirecional | Bidirecional |
| **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 |

# Tecnologias e Ferramentas Utilizadas

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.

### 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 [6]. Resumidamente, *Python* é um conjunto de instruções que fornecemos sob a forma de um programa ao nosso computador para realizar qualquer tarefa especifica. É uma linguagem de programação que possui 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 [7].

### Bibliotecas

De seguida são apresentadas as bibliotecas em *Python* utilizadas no desenvolvimento do projeto.

##### *os*

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.

##### *math*

*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*.

##### *NumPy*

O *NumPy* é uma biblioteca fundamental para computação científica em *Python* [8]. Proporciona suporte para *arrays* multidimensionais e funções matemáticas de alto nível para operarem nestes *arrays*, sendo essencial para operações numéricas eficientes.

##### *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.

##### *Matplotlib*

O *Matplotlib* é uma biblioteca de visualização de dados que permite a criação de gráficos estáticos, animados e interativos em *Python* [9]. É amplamente utilizada para gerar visualizações detalhadas e personalizadas de dados.

##### *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 [10]. Facilita a visualização de dados complexos de maneira percetível e estética.

##### *TensorFlow*

O *TensorFlow* é uma biblioteca de software de código aberto desenvolvida pelo Google, utilizada para construir e treinar modelos de ML e DL [11]. É reconhecida pela flexibilidade e escalabilidade, permitindo a criação de uma variedade de modelos complexos em uma ampla gama de plataformas, desde dispositivos móveis até grandes clusters de servidores.

##### *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 neurais artificiais. É compatível com o *TensorFlow* e oferece flexibilidade para desenvolver uma variedade de arquiteturas de redes neuronais.

* ***keras.models.Sequential, load\_model***: Classes e funções para criar e carregar modelos sequenciais.
* ***keras.layers***: Inclui várias camadas usadas em redes neurais, 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.

##### *scikit-learn*

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

* ***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*.

##### *Folium*

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

* ***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.

### Plataformas

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

##### *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 repositória que ambos os programadores tinham acesso. Desta forma, foi possível ter sempre o código atualizado e sincronizado, facilitando assim o processo de desenvolvimento.

##### *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 [12]. Foi utilizado o *Lucidchart* para criar os diagramas representativos das arquiteturas que desenvolvemos.

##### *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 [13]. Foi através desta plataforma que foram realizadas algumas das nossas reuniões com os nossos orientadores.

##### *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* [14]. Foi neste editor que desenvolvemos o código para a nossa solução.

## Trabalhos Relacionados

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

### Driving Behavior Classification Based on Sensor Data Fusion Using LSTM Recurrent Neural Networks

Este projeto 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.

##### 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.

##### 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 Inercia:
  + 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.

##### 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*.

##### *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 9).

Uma imagem com texto, file, Software de multimédia, captura de ecrã

Descrição gerada automaticamente

Figura 9 - *Software* utilizado para capturar os dados do UAH-*driveset*. FONTE: Driving behavior classification based on sensor data fusion using LSTM recurrent neural networks (2017)

##### 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 10).

Uma imagem com texto, Gráfico, file, captura de ecrã

Descrição gerada automaticamente

Figura 10 - Gráfico ilustrativo do *resampling* dos dados. FONTE: Driving behavior classification based on sensor data fusion using LSTM recurrent neural networks (2017)

##### Resultados Obtidos

Esta implementação obteve resultados bastante concisos visto 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 11).

Uma imagem com texto, captura de ecrã, número, Tipo de letra

Descrição gerada automaticamente

Figura 11 - Resultados da matriz de confusão da implementação proposta em comparação com a classificação de comportamento de condução descrita pelo UAH-*DriveSet*. FONTE: Driving behavior classification based on sensor data fusion using LSTM recurrent neural networks (2017)

### Driving Behavior Classification Based on Oversampled Signals of Smartphone Embedded Sensors Using an Optimized Stacked-LSTM Neural Networks

##### Metodologia Proposta

A metodologia proposta neste artigo 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).

##### 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 neurônios 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.

##### 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.

##### Dataset Utilizado

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

##### 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 (ver Figura 12).

A diagram of a flowchart

Description automatically generated

Figura 12 - Diagrama ilustrativo das várias etapas da implementação

##### Resultados Obtidos

Os resultados obtidos (ver Figura 13) 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.

A white rectangular box with black text

Description automatically generated

Figura 13 - Tabela comparativa dos resultados obtidos

# 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.

## Análise dos sensores de dispositivos móveis

Os dispositivos moveis modernos, como *smarthphone* 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 classificação de condução. No contexto do nosso projeto sobre classificação de condução, utilizaremos o acelerómetro, giroscópio e GPS para recolher dados que ajudam a classificar padrões de condução.

### Definição e Função

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 em um único *chip*, o que permite que sejam compactos e eficientes em termos de energia.

### Tipos de Sensores Utilizados

Neste projeto, utilizamos três tipos principais de sensores:

* **Acelerómetro**
  + 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.
* **Giroscópio**
  + 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.
* **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.

## Análise da Aplicação de Recolha dos Dados

Para recolher os dados do IPL-*Dataset* foi utilizada uma aplicação que foi desenvolvida como parte de um Projeto Informático 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.

## Análise dos *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.

### IPL-*Dataset*

O IPL-*Dataset* é o dataset gerado pela App anteriormente descrita. O *dataset* possui informações capturadas pelos sensores durante viagens que foram feitas pelos estudantes que desenvolveram o projeto. Este *dataset* pode ser armazenado em formatos JSON ou CSV.

##### Estrutura dos Dados

* + **Aceleração (**m/s2): Eixos X, Y e Z.
  + **Giroscópio (°/s):** Eixos *Roll*, *Pitch*, *Yaw*.
  + **GPS:** Latitude, Longitude.
  + ***Speed*:** Velocidade atual em km/h.
  + ***Timestamp*:** Data e hora associada aos dados dos sensores.

### 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.

##### Estrutura dos Dados

* + **Aceleração (**m/s2): Eixos X, Y e Z.
  + **Giroscópio (°/s):** Eixos *Roll*, *Pitch*, *Yaw*.
  + **GPS:** Latitude, Longitude, Velocidade (km/h), Altitude, Precisão Vertical e Horizontal, *Course*.
  + **Câmera:** Deteção de faixa, Deteção de veículos.

# Requisitos

Neste capítulo, 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.

## Requisitos Funcionais

### Aquisição de Dados

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

### Processamento de Dados

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

### 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.

### Classificação

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

### 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.

## Requisitos Não Funcionais

### Desempenho

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

### Escalabilidade

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

### Manutenibilidade

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

## Arquitetura da Solução

A arquitetura que desenvolvemos para a nossa solução pode ser descrita visualmente na Figura 14.

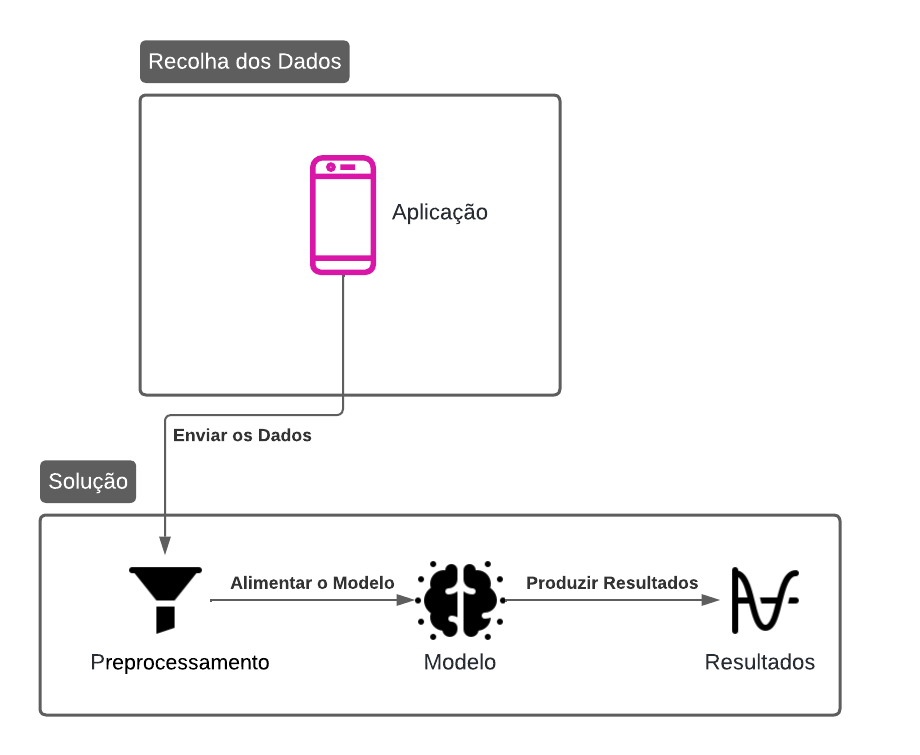


Figura 14 - Arquitetura do protótipo

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, com recurso 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 neural, 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.

# Arquiteturas Propostas

Para abordar a tarefa de classificação de condução com base nos dados dos sensores de smartphones, propomos o uso de várias arquiteturas de redes neurais LSTM, cada uma com suas vantagens específicas para lidar com dados temporais. As arquiteturas propostas são: *Stacked* LSTM, *Bidirectional* LSTM e *Convolutional* LSTM. Todas 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.

## *Stacked* LSTM

### Introdução a *Stacked* LSTM

Uma 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.

### Estrutura da Stacked LSTM

1. Entrada: Dados preprocessados
2. Camadas LSTM
   1. Primeira Camada: Recebe a sequência de entrada e aprende as representações iniciais dos dados temporais.
   2. Camadas intermédias: Cada camada subsequente recebe a saída da camada anterior, permitindo modelar padrões mais complexos e abstrações de longo prazo.
   3. 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.

## *Bidirectional* LSTM

### Introdução a *Bidirectional* LSTM

A *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.

### Estrutura da *Bidirectional* LSTM

1. Entrada: Dados preprocessados.
2. Camadas *Bidirectional* LSTM:
   1. Camada Direcional Frontal: Processa a sequência de entrada na ordem original, aprendendo representações temporais avançadas.
   2. Camada Direcional Reversa: Processa a sequência na ordem inversa, permitindo que o modelo capture padrões que podem depender de informações futuras.
   3. 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 neurônios corresponde ao número de classes que o modelo precisa prever.

## *Convolutional* LSTM

### Introdução a *Convolutional* LSTM

A *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.

### Estrutura da *Convolutional* LSTM

1. Entrada: Dados preprocessados com estrutura espacial-temporal.
2. Camadas *Convolucionais* (Conv1D):
   1. Primeira Camada Conv1D: Aplica filtros convolucionais para extrair características locais dos dados temporais.
   2. Camadas Convolucionais Adicionais: Podem ser adicionadas para capturar características mais complexas e abstrações espaciais.
   3. Camadas LSTM: Após a extração das características espaciais, as camadas LSTM processam as representações temporais.
   4. Primeira Camada LSTM: Modela dependências temporais a curto prazo.
   5. Camadas LSTM Intermédias: Cada camada subsequente captura padrões temporais mais complexos.
   6. Última Camada LSTM: Produz a representação final das dependências temporais.
3. Camada de Saída: Camada densa cujo número de neurônios corresponde ao número de classes que o modelo precisa prever, com ativação *sigmoid*.

# Desenvolvimento

Este capítulo aborda o desenvolvimento teórico e pratico do projeto, assim como as principais decisões tomadas em função do mesmo.

Como o referido anteriormente, este projeto assenta em duas grandes componentes, a componente de Pesquisa e a componente de Implementação.

## Implementação

A implementação deste projeto passa por várias etapas crucias 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:

Tabela 4 - 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. |

## Primeira Abordagem - Classificação de Binária

### Descrição e Caracterização dos Dados

Os dados fornecidos têm por base um *dataset* em formato CSV que contém várias características relacionadas com a condução. Estas características serão utilizadas para identificar diferentes tipos de manobras e comportamentos durante a condução.

##### Estrutura dos Dados

* **Tempo**: *Timestamp* da recolha dos dados
* **Velocidade**: Medida em km/h
* **Aceleração**: Medida em m/s2
* **Latitude e Longitude**: Coordenadas GPS

##### Sensores Utilizados

* **Acelerómetros**: Eixos X, Y, Z
* **Giroscópio**: Eixos X, Y, Z

### 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

##### Processo de Tratamento de Dados

O processo de Tratamento (ver Figura 15) 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 que é composto por 12 elementos no total, 6 elementos positivos e os 6 elementos negativos correspondentes aos dados processados pelos sensores.

1. def separate\_positives\_negatives(data):

2.   # Ensure the input is converted to a NumPy array for easier manipulation

3.   data = np.array(data)

4.

5.   # Create two empty arrays to store positive and negative values

6.   positives = np.zeros\_like(data)

7.   negatives = np.zeros\_like(data)

8.

9.   # Use boolean indexing to separate positive and negative values

10.   positives[data > 0] = data[data > 0]

11.   negatives[data < 0] = -data[data < 0]

12.

13.   # Combine the positive and negative values into a single 2D array

14.   return (positives, negatives)

Listagem 1 - Função *separate\_positives\_negatives*



Figura 15 - Diagrama ilustrativo do tratamento dos dados

### 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***.**

##### 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.

##### Processo de Classificação dos Dados

O processo de classificação dos dados (ver Figura 16) 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 thresholdfornecido como parâmetro de entrada, ou seja,*max\_value \* threshold*.

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.

9. return classification

Listagem 2 - Função *y\_classification*

|  |
| --- |
| Figura 16 - Diagrama ilustrativo da classificação dos dados |

### 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]

##### Processo de Normalização dos Dados

O processo de normalização (ver Figura 19) 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 17**);
* Para o giroscópio, em que os parâmetros de entrada são: **‘***gyrPositiveX’***,** *gyrNegativeX***,** *gyrPositiveY***,** *gyrNegativeY***,** *gyrPositiveZ* **e**

*gyrNegativeZ* (ver Figura 18)

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 17 - Diagrama ilustrativo da função *max of vectors* aplicada ao giroscópio



Figura 18 - Diagrama ilustrativo da função *max of vectors* aplicada ao giroscópio



Figura 19 - Diagrama ilustrativo da normalização dos dados

### 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 20 e Figura Figura 21 representam estas visualizações dos conjuntos de dados:

A map with orange dots

Description automatically generated

Figura 20 - Gráfico de manobras do *dataset*

A map of a road with a location pin

Description automatically generated

Figura 21 - Gráfico de manobras, com *zoom*, do *dataset*

### 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

##### 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*=0.2significa 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 22).

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 22 - Diagrama ilustrativo da separação dos dados

### Criação dos Modelos

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

##### *Stacked* 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]*)
* *Dropout* *Layer*
  + Desativa uma % das unidades da camada anterior aleatoriamente para prevenir overfitting.
* 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 é usada para capturar dependências temporais em dados sequenciais.
* LSTM *Layer* (128 *units*)

1. model\_lstm = Sequential()

2. model\_lstm.add(LSTM(64, input\_shape=(1, train.shape[2]), return\_sequences=True))

3. model\_lstm.add(Dropout(0.2))

4.

5. model\_lstm.add(LSTM(64))

6. model\_lstm.add(Dropout(0.2))

7.

8. model\_lstm.add(Dense(32,activation='relu'))

9. model\_lstm.add(Dense(12,activation='sigmoid'))

10.

11. # Compile the model

12. model\_lstm.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

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

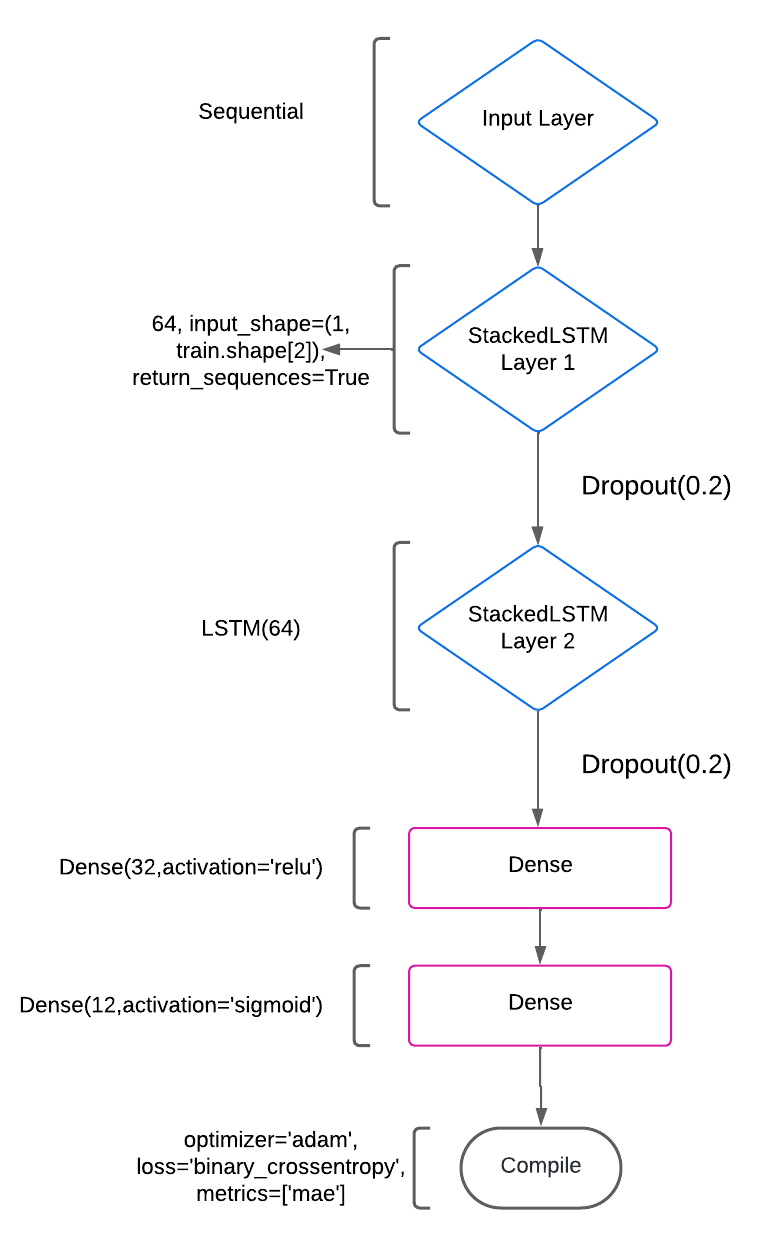


Figura 23 - Diagrama ilustrativo da estrutura do modelo *Stacked* LSTM

##### *Bidirectional* 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*
  + Normaliza a ativação da camada anterior para acelerar o treino e melhorar a estabilidade da rede.
* *Dropout Layer*
  + Desativa uma % das unidades da camada anterior aleatoriamente para prevenir overfitting.
* LSTM *Layer* (256 *units*, *return\_sequences=True*)
  + Adiciona uma camada LSTM com 256 unidades, retornando a sequência completa de saída para a próxima camada. LSTM é usada para capturar dependências temporais em dados sequenciais.
* LSTM *Layer* (128 *units*)

1. # Model configuration

2. dropout1 = 0.5

3. dropout2 = 0.2

4. dropout3 = 0.1

5. initial\_learning\_rate = 0.001

6.

7. # Define the model

8. model = Sequential()

9. model.add(Bidirectional(LSTM(64, return\_sequences=True), input\_shape=(train.shape[1], train.shape[2])))

10. model.add(BatchNormalization())

11. model.add(Dropout(dropout1))

12. model.add(Bidirectional(LSTM(64, return\_sequences=False)))

13. model.add(BatchNormalization())

14. model.add(Dropout(dropout2))

15. model.add(Dense(64, activation='relu'))

16. model.add(BatchNormalization())

17. model.add(Dropout(dropout3))

18. model.add(Dense(12, activation='sigmoid'))

19.

20.

21.

22. # Compile the model

23. model.compile(loss='binary\_crossentropy',optimizer='adam', metrics=['accuracy'])

24.

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

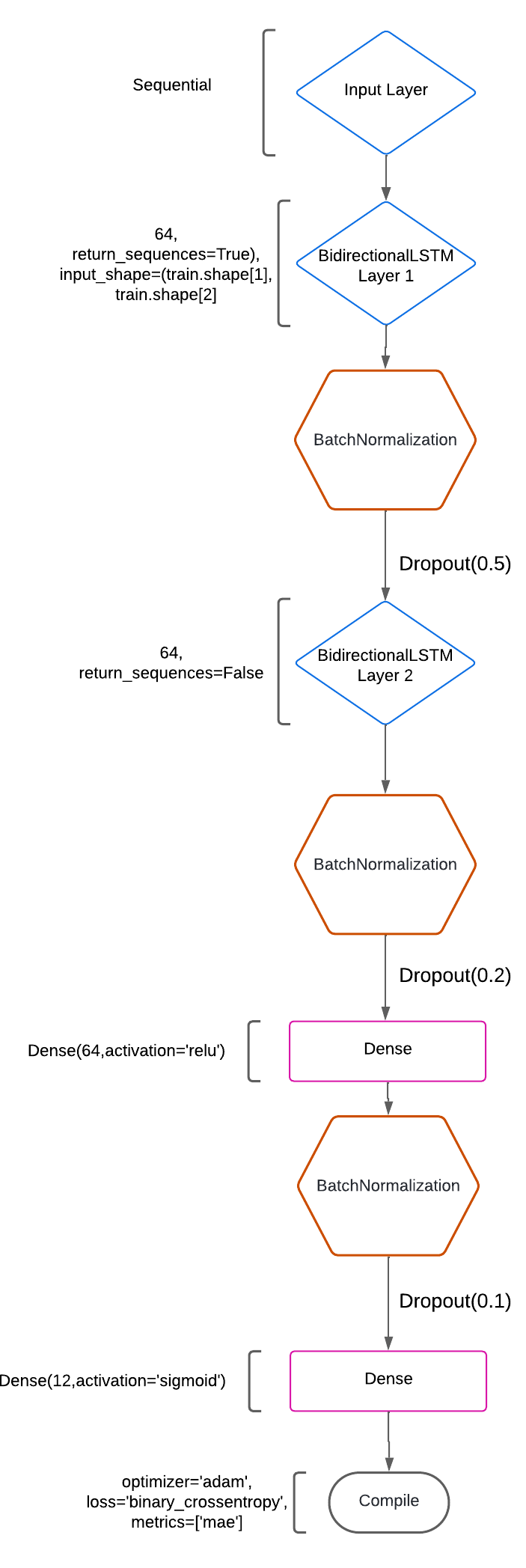


Figura 24 - Diagrama ilustrativo da estrutura do modelo *Bidirectional* LSTM

##### *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*
  + Normaliza a ativação da camada anterior para acelerar o treinamento e melhorar a estabilidade da rede.
* *Dropout Layer*
  + Desativa uma % das unidades da camada anterior aleatoriamente para prevenir *overfitting*.
* 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 é usada para capturar dependências temporais em dados sequenciais.
* LSTM *Layer* (128 *units*)

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

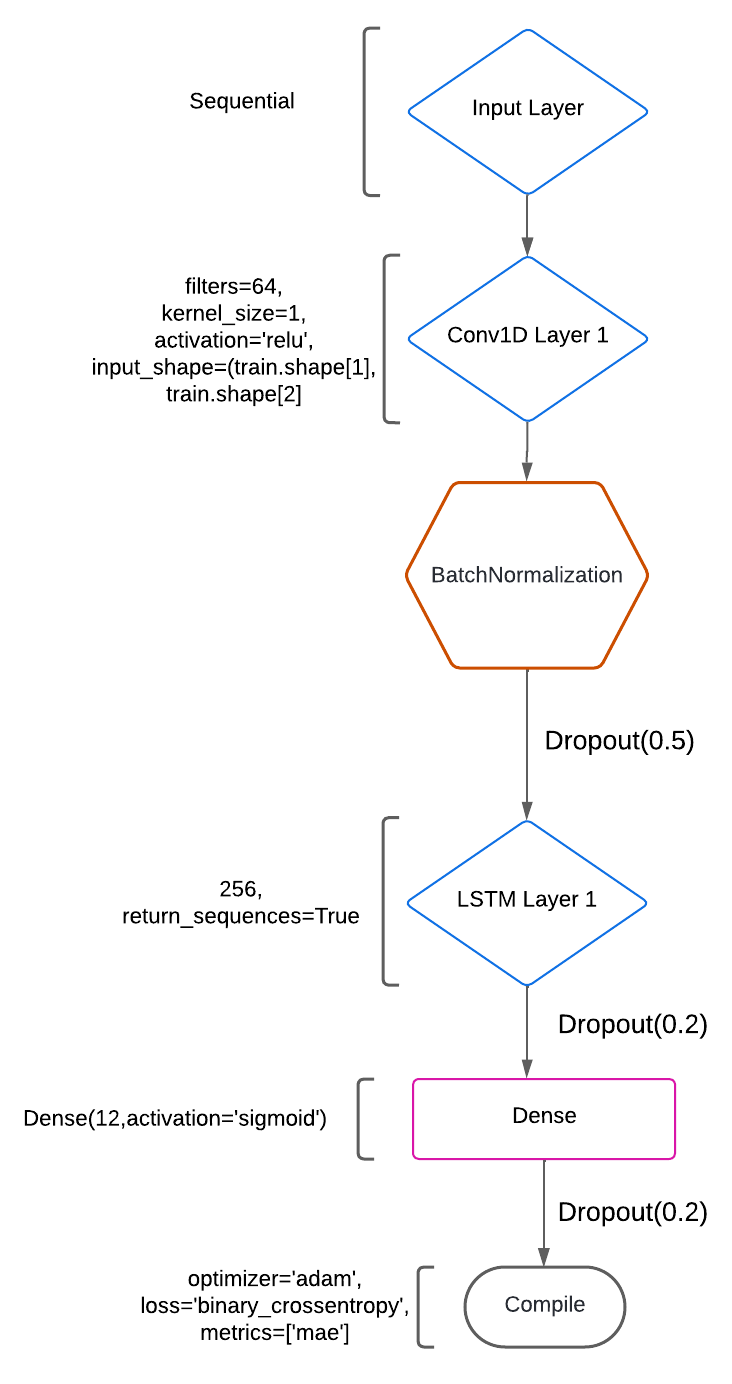


Figura 25 - Diagrama ilustrativo da estrutura do modelo ConvolutionalLSTM

### Compilação e Treino

##### *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.

11.

12. # Train the model

13. 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

##### *Bidirectional* LSTM

1. model\_lstm.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

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

1. # best callback for the model

2. best\_model\_file = 'runtime\_saves/models/BiLSTM/model.keras'

3. best\_model = ModelCheckpoint(best\_model\_file, monitor='val\_loss', mode='min', verbose=1, save\_best\_only=True)

4.

5. # early stopping callback

6. early\_stop = EarlyStopping(monitor='val\_loss', mode='min', verbose=1, patience=10)

7.

8. # fit the model

9. 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

##### *Convolutional* LSTM

1. model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

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

1. # best callback for the model

2. best\_model\_file = 'runtime\_saves/models/ConvLSTM/model.keras'

3. best\_model = ModelCheckpoint(best\_model\_file, monitor='val\_loss', mode='min', verbose=1, save\_best\_only=True)

4.

5. # early stopping callback

6. early\_stop = EarlyStopping(monitor='val\_loss', mode='min', verbose=1, patience=10)

7.

8. # fit the model

9. 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

## 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 acelerômetros, giroscópios e GPS. O objetivo principal foi identificar padrões que pudessem distinguir diferentes estilos de condução.

### 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***.

##### Sensores Utilizados

* Acelerómetros
* Giroscópio
* GPS

##### Estrutura dos Dados

* **Acelerômetro:** Medidas de aceleração nos eixos X, Y, Z (em m/s²)
* **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

### 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 26).

##### Processo de Tratamento de Dados

O processo de tratamento dos dados segue os seguintes passos:

A diagram of a company

Description automatically generated

Figura 26 - Etapas de processamento de dados (Segunda Abordagem)

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).

1. **Limpeza e Estruturação dos Dados**:

Durante a importação, as colunas irrelevantes como ***timestamp***, ***id***, e ***createdAt*** foram removidas.

1. **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 acelerômetros e giroscópios. Esta técnica suaviza as flutuações bruscas nos dados, resultando em medições mais estáveis.

Tabela 5 - 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 6 - 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 |
| … | … | … | … | … | … |

1. **Normalização dos Dados**:

Após a suavização dos dados, foi aplicada uma normalização nas colunas do acelerômetro 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.

1. **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.

1. **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

1. **Codificação de Etiquetas**:

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.

### Modelos

##### *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 acelerômetros 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***.

##### *Bidirectional* LSTM

**Descrição do Modelo**

Ao contrário das LSTMs tradicionais, que processam as sequências de dados em uma 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 bidirecional processa a sequência e passa a saída para a próxima camada.
  + A segunda camada também é LSTM bidirecional e adiciona profundidade ao modelo.
  + A terceira e última camada LSTM bidirecional 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***.

##### 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.

# Testes e Resultados

Neste capítulo iremos apresentar através de tabelas comparativas de métricas de avaliação sobre as diferentes arquiteturas desenvolvidas nas duas abordagens.

## Objetivos dos Testes

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.

## Métricas de Avaliação

As métricas utilizadas para avaliar o desempenho dos modelos incluem:

### **Acurácia**

A acurácia 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:

* 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.

### **Precisão**

A precisão mede a proporção de verdadeiros positivos entre todas as instâncias que foram classificadas como positivas. É uma métrica importante quando o custo de falsos positivos é alto. A fórmula é dada por:

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

### *Recall* (Sensibilidade)

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

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

### F1-Score

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

### *Jaccard* *Score*

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

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

### *Hamming* *Loss*

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

* N é o número total de amostras.
* é o rótulo previsto para a i-ésima amostra.
* é o rótulo verdadeiro para a i-ésima amostra.
* é uma função indicadora que vale 1 se e 0 caso contrário.

## Resultados

### Primeira Abordagem - Classificação de Binária

Tabela 7 - Comparação de métricas entre os modelos 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% |

##### Análise de resultados

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

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:

**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 27. Esta foi a arquitetura a atingir mais rapidamente valores próximos de **0,** logo a partir da **2º época.**

A graph with blue and orange lines

Description automatically generated

Figura 27 - Comparação da *loss* e *val\_loss* para o modelo *Convolutional* LSTM Primeira Abordagem

Na Figura 28 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**.

A graph with blue and red lines

Description automatically generated

Figura 28 - Comparação dos valores reais vs valores previstos pelo modelo *Convolutional* LSTM Primeira Abordagem

**Stacked LSTM**

Neste modelo conseguimos perceber que a diminuição da ***loss*** e a ***val\_loss*** foi um bocado menorpor 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 29.

A graph with a line

Description automatically generated

Figura 29 - 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 30).

A graph of blue and red lines

Description automatically generated

Figura 30 - Comparação dos valores reais vs valores previstos pelo modelo *Stacked* LSTM Primeira Abordagem

**Bidirectional LSTM**

A arquitetura ***Bidirecional*** 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 31.

A graph with blue and orange lines

Description automatically generated

Figura 31 - Comparação da *loss* e *val\_loss* para o modelo Bidirecional LSTM Primeira Abordagem

Em contrapartida a **Bidirecional 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 32).



Figura 32 - Comparação dos valores reais vs valores previstos pelo modelo Bidirecional LSTM Primeira Abordagem

### Segunda Abordagem - Classificação de Classes

|  |  |  |
| --- | --- | --- |
| **Metric** | **BiLSTM** | ***Stacked* LSTM** |
| ***Train Accuracy*** | **99.48%** | 97.91% |
| ***Test Accuracy*** | 96.20% | 91.89% |
| ***Train Loss*** | 0.0171 | 0.0765 |
| ***Test Loss*** | 0.1003 | 0.2373 |
| ***Precision*** | 84.66% | 92.02% |
| ***Recall*** | 81.66% | 92.28% |
| ***F1 Score*** | 87.56% | 92.07% |
| ***MPCE*** | 0.0381 | 0.0772 |

Tabela 8 - Comparação de resultados dos modelos da Segunda abordagem

***Stacked* LSTM**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Class*** | ***PCE*** | ***Accuracy*** | ***Precison*** | ***Recall*** | ***F1-score*** |
| ***Slow*** | 10.2% | 89.8% | 95% | 90% | 92% |
| ***Normal*** | 6.7% | 93.3% | 86% | 93% | 89% |
| ***Aggressive*** | 6.3% | 93.7% | 96% | 94% | 95% |

Tabela 9 - Resultados por classe do modelo *Stacked* LSTM (Segunda abordagem)

A green bar graph with white text

Description automatically generated

Figura 33 - Erro por classe *Stacked* LSTM (Segunda abordagem)

A green squares with white text

Description automatically generated

Figura 34 - Matriz de Confusão *Stacked* LSTM (Segunda abordagem)

**Bidirecional LSTM**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Class*** | ***PCE*** | ***Accuracy*** | ***Precison*** | ***Recall*** | ***F1-score*** |
| ***Slow*** | 0.0339 | 96.6% | 97% | 97% | 97% |
| ***Normal*** | 0.0512 | 94.9% | 96% | 95% | 95% |
| ***Aggressive*** | 0.0291 | 97.1% | 96% | 97% | 97% |

Tabela 10 - Resultados por classe do modelo Bidirecional LSTM (Segunda abordagem)

A graph showing a number of green rectangular bars

Description automatically generated

Figura 35 - Erro por classe Bidirecional LSTM (Segunda abordagem)

A green squares with white text

Description automatically generated

Figura 36 -Matriz de Confusão Bidirecional LSTM (Segunda abordagem)

##### Análise de resultados

Nesta seção, discutimos os resultados obtidos e comparamos o desempenho das diferentes arquiteturas de redes neurais. Observamos que:

## Considerações Finais

# Artigo Científico

Para complementar o nosso projeto, foi-nos proposto pelos orientadores redigirmos um artigo científico a descrever a nossa abordagem ao problema. Foi entao elaborado um artigo intitulado de “An LSTM-Based Approach for Driving Behaviour Classification”. O objetivo principal do artigo foi explicar a forma como implementamos o nosso projeto.



## Objetivos do Artigo

Como referido anteriormente, o principal objetivo do artigo é a descrição da nossa abordagem e os resultados que conseguimos obter. Alem disso o artigo visou contribuir com novas metodologias na área de classificação de problemas de condução de modo a enriqueces a literatura já existente.

## Metodologia

Para a elaboração do artigo, seguimos uma estrutura rigorosa, baseada em:

**Revisão Bibliográfica**: Levantamento da literatura relevante já existente para contrastar com o nosso artigo.

**Metodologia Experimental**: Explicação detalhada dos métodos e técnicas aplicadas para alcançar os resultados discutidos.

**Análise de Resultados**: Discussão sobre a qualidade dos resultados obtidos.

**Conclusões**: Síntese das descobertas mais importantes e sugestões para trabalhos futuros.

## Resultados e Discussão

## Conclusões e Contributos Científicos

## Considerações Finais

# 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

# Bibliografia

[1] *Processamento de Linguagem Natural com Deep Learning: Uma nova Era | by Savio Teles | Data Hackers | Medium*. (n.d.). Retrieved August 13, 2024, from <https://medium.com/data-hackers/a-revolu%C3%A7%C3%A3o-no-processamento-de-linguagem-natural-com-deep-learning-eba175f64c01>

[2] *O que é Redes Neurais (Neural Networks)? - Glossário de Automação*. (n.d.). Retrieved August 13, 2024, from https://glossario.maiconramos.com/glossario/o-que-e-redes-neurais-neural-networks/

[3] *O que é RNN? — Explicação sobre redes neurais recorrentes — AWS*. (n.d.). Retrieved August 13, 2024, from https://aws.amazon.com/pt/what-is/recurrent-neural-network/

[4] *Nessa rede de deep learning a saída de um neurônio é aplicada como entrada no próprio neurônio e/ou em - brainly.com.br*. (n.d.). Retrieved August 13, 2024, from https://brainly.com.br/tarefa/56827023

[5] Modesto, Á. G., Almeida, F. da C., Corrêa, W. da S., & Fortes, M. Z. (2023). Mapeamento tecnológico da aplicação de redes neurais para eficiência energética de sistemas de bombeamento. Revista de Tecnologia Aplicada, 82–95. <https://doi.org/10.48005/2237-3713RTA2022V11N3P8295>

[6] *O que é Python? – Explicação sobre a linguagem Python – AWS*. (n.d.). Retrieved August 6, 2024, from <https://aws.amazon.com/pt/what-is/python/>

[7] *What is Python? it’s Uses and Applications - GeeksforGeeks*. (n.d.). Retrieved August 6, 2024, from <https://www.geeksforgeeks.org/what-is-python/>

[8] *NumPy: A Biblioteca Fundamental para Computação Científica em Python*. (n.d.). Retrieved August 15, 2024, from <https://www.inovaproj.com.br/engenharia/matem%C3%A1tica-computacional/artigos/numpy-a-biblioteca-fundamental-para-computa%C3%A7%C3%A3o-cient%C3%ADfica-em-python>

[9] Matplotlib na prática: códigos úteis e comandos essenciais - Didática Tech. (n.d.). Retrieved August 15, 2024, from <https://didatica.tech/matplotlib-na-pratica-codigos-uteis-e-comandos-essenciais/>

[10] Principais bibliotecas do Python para Ciência de Dados. (n.d.). Retrieved August 15, 2024, from <https://www.dio.me/articles/principais-bibliotecas-do-python-para-ciencia-de-dados>

[11] OpenCV e TensorFlow, o que são e o melhor modo de aplicar eles | by Diogo Goes Zanetti | Medium. (n.d.). Retrieved August 15, 2024, from <https://omegadiogo.medium.com/opencv-e-tensorflow-o-que-s%C3%A3o-e-o-melhor-modo-de-aplicar-eles-71db48cc7f43>

[12] *O que é diagramação inteligente?| Lucidchart*. (n.d.). Retrieved August 7, 2024, from <https://www.lucidchart.com/blog/pt/diagramas-inteligentes>

[13] *Introdução ao Microsoft Teams - Suporte da Microsoft*. (n.d.). Retrieved August 7, 2024, from <https://support.microsoft.com/pt-pt/office/introdu%C3%A7%C3%A3o-ao-microsoft-teams-b98d533f-118e-4bae-bf44-3df2470c2b12>

[14] *Visual Studio Code Frequently Asked Questions*. (n.d.). Retrieved August 7, 2024, from <https://code.visualstudio.com/Docs/supporting/faq>

[15] Xing, Y., Lv, C., Wang, H., Cao, D., & Velenis, E. (2020). An ensemble deep learning approach for driver lane change intention inference. *Transportation Research Part C: Emerging Technologies*, *115*, 102615. https://doi.org/10.1016/J.TRC.2020.102615

[16] Deo, N., & Trivedi, M. M. (n.d.). *Multi-Modal Trajectory Prediction of Surrounding Vehicles with Maneuver based LSTM*.

[17] Darsono, A. M., Mat Yazi, N. H., Ja’Afar, A. S., Othman, M. A., & Ahmad, M. I. (2024). Utilizing LSTM Networks for the Prediction of Driver Behavior. *Przeglad Elektrotechniczny*, *2024*(4), 182–185. https://doi.org/10.15199/48.2024.04.34

[18] Shahverdy, M., Fathy, M., Berangi, R., & Sabokrou, M. (2020). Driver behavior detection and classification using deep convolutional neural networks. *Expert Systems with Applications*, *149*, 113240. https://doi.org/10.1016/J.ESWA.2020.113240

[19] Cai, Y., Zhao, R., Wang, H., Chen, L., Lian, Y., & Zhong, Y. (2023). CNN-LSTM Driving Style Classification Model Based on Driver Operation Time Series Data. *IEEE Access*, *11*, 16203–16212. https://doi.org/10.1109/ACCESS.2023.3245146

[20] *LSTM Full Form - Long Short-Term Memory - GeeksforGeeks*. (n.d.). Retrieved July 17, 2024, from https://www.geeksforgeeks.org/lstm-full-form-long-short-term-memory/

[21] *What is LSTM - Long Short Term Memory? - GeeksforGeeks*. (n.d.). Retrieved July 17, 2024, from <https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/>

# Anexos

Elemento a figurar, **quando aplicável**.

# Glossário

Elemento a figurar, **quando aplicável**.