



Site Web Meteo com Inteligência Artificial

Licenciatura em Engenharia Informática

José Pedro Da Silva Ferreira

Rúben Francisco Antunes

Leiria, Junho de 2025

Site Web Meteo com Inteligência Artificial

Licenciatura em Engenharia Informática

José Pedro Da Silva Ferreira

Rúben Francisco Antunes

Trabalho de Projeto da unidade curricular de Projeto Informático realizado sob a orientação
do Professor Doutor João Da Silva Pereira, do Professor Doutor Paulo Manuel Almeida
Costa e do Professor Doutor Rui Vasco Guerra Baptista Monteiro

Leiria, Junho de 2025

Agradecimentos

A concretização deste projeto não teria sido possível sem o apoio e contributo de diversas pessoas, às quais deixamos o nosso mais sincero agradecimento.

Em primeiro lugar, gostaríamos de agradecer aos professores orientadores, Professor Doutor João Da Silva Pereira, Professor Doutor Paulo Manuel Almeida Costa e Professor Doutor Rui Vasco Guerra Baptista Monteiro, pelo acompanhamento, orientação científica e disponibilidade demonstrada ao longo de todas as fases deste trabalho. A sua dedicação, conhecimento e exigência foram fundamentais para a concretização deste projeto.

Estendemos também o nosso agradecimento a todos os professores do curso de Engenharia Informática, que ao longo destes anos contribuíram de forma significativa para a nossa formação académica, transmitindo não apenas conhecimento técnico, mas também espírito crítico, rigor e profissionalismo.

Aos nossos colegas e amigos, agradecemos o apoio constante, a partilha de experiências, as conversas motivadoras e os momentos de descontração que tornaram este percurso mais leve e enriquecedor.

Por fim, um agradecimento muito especial às nossas famílias, pelo amor incondicional, paciência, compreensão e incentivo permanente. O vosso apoio foi essencial em todos os momentos, especialmente nos mais desafiantes.

A todos, o nosso muito obrigado.

Resumo

A previsão meteorológica desempenha um papel fundamental na vida quotidiana das populações, influenciando áreas tão diversas como a agricultura, os transportes, o turismo e a gestão de recursos hídricos. Em Portugal, esta responsabilidade recai sobre o Instituto Português do Mar e da Atmosfera (IPMA), uma organização governamental que fornece previsões regulares baseadas em dados recolhidos por satélites, radares e estações meteorológicas.

No contexto atual, torna-se cada vez mais relevante desenvolver métodos de previsão que apresentem uma maior taxa de acerto e uma fiabilidade superior, sobretudo perante fenómenos meteorológicos extremos ou alterações climáticas. Foi neste sentido que desenvolvemos um projeto centrado na utilização de redes neurais para previsão de chuva em Portugal Continental, procurando tirar partido das mais recentes abordagens da inteligência artificial (IA).

O nosso trabalho focou-se na comparação entre dois tipos de modelos de *Deep Learning*: *Convolutional Neural Networks* (CNN) e *Liquid Neural Networks* (LNN), tendo sido exploradas duas variantes de LNN: *Liquid Time-Constant* (LTC) e *Closed-Form Continuous-Time* (CfC). No treino da CNN foram apenas utilizados dados estáticos – uma imagem de radar para cada valor de precipitação – já na LNN foram comparados os treinos com dados estáticos e os treinos com séries de dados temporais – sequência de imagens de radar para cada valor de precipitação – com o objetivo de modelar a evolução temporal dos fenómenos de precipitação.

Os resultados obtidos evidenciaram que a abordagem mais eficaz foi a utilização de uma LNN baseada em LTC e treinada com séries de dados temporais. Este modelo alcançou, na sua configuração final, níveis de *validation accuracy* de 97.4%, 96.8% e 88.8% para previsões com 1, 2 e 3 horas de antecedência, respetivamente.

Apesar destes resultados promissores, enfrentámos diversos desafios ao longo do desenvolvimento do projeto. Destacamos, em particular, o problema de *overfitting*, que comprometeu a capacidade de generalização de alguns modelos, bem como as limitações do *dataset* utilizado, tanto em termos da sua dimensão como da sua diversidade.

Ainda assim, a análise global dos resultados demonstra a eficácia das abordagens implementadas, sendo especialmente notória a mais-valia da integração de informação temporal nas LNN. A utilização deste tipo de redes revela-se, assim, uma estratégia promissora para o aprimoramento das previsões meteorológicas baseadas em inteligência artificial.

Palavras-chave: Previsão Meteorológica; Inteligência Artificial; *Deep Learning*; CNN; LNN; LTC; CfC

Abstract

Weather forecasting plays a fundamental role in people's daily lives, influencing areas as diverse as agriculture, transportation, tourism and water management. In Portugal, this responsibility falls to the Portuguese Institute for Sea and Atmosphere (IPMA), a governmental organisation that provides regular forecasts based on data collected by satellites, radars and weather stations.

In the current context, it is becoming increasingly important to develop forecasting methods that have a higher accuracy rate and greater reliability, especially in the face of extreme weather events or climate change. It was with this in mind that we developed a project focused on the use of neural networks to forecast rainfall in mainland Portugal, seeking to take advantage of the latest approaches in artificial intelligence (AI).

Our work focused on the comparison between two types of Deep Learning models: Convolutional Neural Networks (CNN) and Liquid Neural Networks (LNN), exploring two variants of LNN: Liquid Time-Constant (LTC) and Closed-Form Continuous-Time (CfC). In the CNN training, only static data was used – a radar image for each precipitation value – while in the LNN, training with static data and training with time-series data – a sequence of radar images for each precipitation value – were compared with the aim of modelling the temporal evolution of precipitation phenomena.

The results obtained showed that the most effective approach was the use of an LNN based on LTC and trained with time-series data. This model achieved, in its final configuration, validation accuracy levels of 97.4%, 96.8% and 88.8% for forecasts with 1, 2 and 3 hours in advance, respectively.

Despite these promising results, we faced several challenges throughout the development of the project. We highlight, in particular, the problem of overfitting, which compromised the generalization capabilities of some models, as well as the limitations of the dataset used, both in terms of size and diversity.

Nevertheless, the overall analysis of the results demonstrates the effectiveness of the implemented approaches, with the added value of integrating temporal information into LNNs being particularly notable. The use of this type of network thus proves to be a promising strategy for improving weather forecasts based on artificial intelligence.

Keywords: Weather Forecasting; Artificial Intelligence; Deep Learning; CNN; LNN; LTC; CfC

Índice

Agradecimentos.....	ii
Resumo.....	iv
Abstract	vi
Lista de Figuras.....	xi
Lista de Tabelas	xiv
Lista de Siglas e Acrónimos	xvi
1. Introdução.....	1
2. Enquadramento teórico	3
2.1. Redes Neuronais	3
2.2. Deep Learning	4
2.3. <i>Convolutional Neural Networks</i>	5
2.3.1. <i>Convolutional Layer</i>	7
2.3.2. <i>Pooling Layer</i>	8
2.3.3. <i>Normalization Layer</i>	9
2.3.4. <i>Dropout Layer</i>	10
2.3.5. <i>Flatten Layer</i>	11
2.3.6. <i>Fully-connected Layer</i>	11
2.4. Liquid Neural Networks	13
2.5. Data Augmentation	14
2.5.1. Dados aumentados (<i>augmented</i>) vs. Dados sintéticos.....	15
2.6. <i>Oversampling e Undersampling</i>	16
2.7. <i>Overfitting e Underfitting</i>	17
3. Descrição do processo	20
3.1. Ferramentas utilizadas	20

3.1.1.	<i>Jupyter Notebook</i>	20
3.1.2.	<i>Flask</i>	21
3.1.3.	<i>Vue.js</i>	21
3.1.4.	<i>TensorFlow</i>	22
3.1.5.	<i>GitHub</i>	23
3.1.6.	<i>Excel</i>	23
3.2.	<i>Dataset utilizado</i>	24
3.2.1.	Pré-processamento das imagens	26
3.2.2.	Normalização dos dados.....	27
3.2.1.	<i>Oversampling e Undersampling</i>	28
3.2.2.	<i>Data Augmentation</i>	29
3.3.	Construção das redes neurais	31
3.3.1.	Arquitetura CNN original.....	31
3.3.2.	Arquitetura LNN baseada em LTC	33
3.3.3.	Arquitetura LNN baseada em CfC	35
3.3.4.	Arquitetura híbrida: CNN com LTC (sem camada <i>fully-connected</i>)	36
3.3.5.	Arquitetura híbrida: CNN com CfC (sem camada <i>fully-connected</i>)	38
3.3.6.	Arquitetura híbrida: CNN com LTC (com camada <i>fully-connected</i>).....	39
3.3.7.	Arquitetura híbrida: CNN com CfC (com camada <i>fully-connected</i>).....	40
3.4.	Treino e avaliação dos modelos	41
3.4.1.	Primeira abordagem.....	42
3.4.2.	Segunda abordagem.....	45
3.4.3.	Terceira abordagem	46
3.4.4.	Quarta abordagem.....	47
3.4.5.	Quinta abordagem.....	50
3.4.6.	Sexta abordagem.....	51
3.4.7.	Abordagem final	54
4.	Aplicação real do modelo	56
4.1.	API em <i>Flask</i>	56
4.1.1.	<i>Endpoint /radar-images</i>	57
4.1.2.	<i>Endpoint /predict-rain</i>	58

4.2.	Web Meteo	59
4.2.1.	Clock.vue.....	60
4.2.2.	MapImage.vue.....	62
4.2.3.	PredictionsTable.vue	64
4.3.	Instalação e uso.....	67
4.3.1.	Requisitos	67
4.3.2.	Instruções	67
5.	Análise de resultados.....	69
5.1.	Resultados com dados estáticos.....	70
5.2.	Resultados com dados temporais	74
5.3.	Resultados na abordagem final.....	77
6.	Conclusão	81
	Bibliografia.....	82

Lista de Figuras

Figura 1 - Componentes de um perceptrão	3
Figura 2 - Simples ilustração de <i>backpropagation</i>	4
Figura 3 - Estrutura de redes neurais profundas	4
Figura 4 - Representação de uma imagem como matriz de píxeis	6
Figura 5 - Estrutura de uma <i>Convolutional Neural Network</i>	6
Figura 6 - Representação da operação de convolução	7
Figura 7 - Movimento do <i>kernel</i>	8
Figura 8 - Tipos de <i>pooling</i>	9
Figura 9 - Dados normalizados (vermelho) vs Dados não normalizados (azul)	10
Figura 10 - <i>Dropout</i> aplicado a uma típica rede neuronal	10
Figura 11 - Representação do processo de <i>flattening</i>	11
Figura 12 - <i>Fully-connected Layer</i> para problemas de classificação binária	12
Figura 13 - <i>Fully-connected Layer</i> para problemas de classificação multi-classe	12
Figura 14 - Estrutura de uma <i>Liquid Neural Network</i>	14
Figura 15 - Técnicas de <i>Image Augmentation</i>	15
Figura 16 - Exemplo de <i>overfitting</i>	17
Figura 17 - Exemplo de <i>underfitting</i>	18
Figura 18 - Exemplo de um <i>Jupyter Notebook</i>	20
Figura 19 - Exemplo de uma <i>app</i> em <i>Flask</i>	21
Figura 20 - Exemplo de uma <i>app</i> em <i>Vue.js</i>	22
Figura 21 - Código para obter dados do IPMA	25
Figura 22 - <i>Script</i> para automatizar a recolha de dados.....	26
Figura 23 - Função para remover os píxeis pretos da imagem.....	26
Figura 24 - Código com o incremento de um <i>counter dictionary</i>	29
Figura 25 - Função para gerar imagem substituta	29
Figura 26 - Configuração do <i>Data Augmentation</i>	30
Figura 27 - Treino do modelo com <i>Data Augmentation</i> em tempo real.....	30
Figura 28 - Código da arquitetura da CNN	32

Figura 29 - Código da arquitetura da LNN baseada em LTC com dados estáticos	33
Figura 30 - Código da arquitetura da LNN baseada em LTC com dados temporais	34
Figura 31 - Código da arquitetura da LNN baseada em CfC com dados estáticos.....	35
Figura 32 - Código da arquitetura da LNN baseada em CfC com dados temporais	35
Figura 33 - Código da arquitetura híbrida CNN + LTC com dados estáticos (sem camada <i>fully-connected</i>)	36
Figura 34 - Código da arquitetura híbrida CNN + LTC com dados temporais (sem camada <i>fully-connected</i>)	37
Figura 35 - Código da arquitetura híbrida CNN + CfC com dados temporais (sem camada <i>fully-connected</i>)	38
Figura 36 - Código da arquitetura híbrida CNN + CfC com dados estáticos (sem camada <i>fully-connected</i>)	38
Figura 37 - Código da arquitetura híbrida CNN + LTC com dados estáticos (com camada <i>fully-connected</i>)	39
Figura 38 - Código da arquitetura híbrida CNN + CfC com dados estáticos (com camada <i>fully-connected</i>)	40
Figura 39 - Código do <i>endpoint</i> /radar-images	57
Figura 40 - Código do <i>endpoint</i> /predict-rain	58
Figura 41 - <i>Website</i> Web Meteo	59
Figura 42 - Código do <i>template</i> do Clock.vue	60
Figura 43 - Código do <i>script</i> do Clock.vue	61
Figura 44 - Código do <i>template</i> do MapImage.vue	62
Figura 45 - Código do <i>script</i> do MapImage.vue	63
Figura 46 - Código do <i>template</i> do PredictionsTable.vue	64
Figura 47 - Código do <i>script</i> do PredictionsTable.vue	66
Figura 48 - Gráfico comparativo da <i>accuracy</i> em dados estáticos	71
Figura 49 - – Gráfico comparativo da <i>validation accuracy</i> em dados estáticos	71
Figura 50 - Gráfico comparativo da <i>validation loss</i> em dados estáticos	72
Figura 51 - Gráfico comparativo da <i>loss</i> em dados estáticos	72
Figura 52 - Gráfico comparativo da <i>accuracy</i> em dados temporais	74
Figura 53 - Gráfico comparativo da <i>loss</i> em dados temporais	75
Figura 54 - Gráfico comparativo da <i>validation accuracy</i> em dados temporais	75
Figura 55 - Gráfico comparativo da <i>validation loss</i> em dados temporais	76

Figura 56 - Gráfico comparativo da <i>validation accuracy</i> no modelo final.....	78
Figura 57 - Gráfico comparativo da <i>accuracy</i> no modelo final	78
Figura 58 - Gráfico comparativo da <i>validation loss</i> no modelo final.....	79
Figura 59 - Gráfico comparativo da <i>loss</i> no modelo final	79

Lista de Tabelas

Tabela 1 - Critérios de emissão de avisos meteorológicos do IPMA	27
Tabela 2 - Critérios de classificação de precipitação do IPMA	28
Tabela 3 - Tabela de amostras obtidas	28
Tabela 4 - Resultados do teste: CNN com 50 <i>epochs</i> e <i>data augmentation</i>	42
Tabela 5 - Resultados do teste: CNN com 100 <i>epochs</i> e <i>data augmentation</i>	43
Tabela 6 - Resultados do teste: CNN com 200 <i>epochs</i> e <i>data augmentation</i>	44
Tabela 7 - Resultados do teste: CNN com 100 <i>epochs</i> sem <i>data augmentation</i>	44
Tabela 8 - Resultados do teste: LTC com dados estáticos e 100 <i>epochs</i>	45
Tabela 9 - Resultados do teste: CfC com dados estáticos e 100 <i>epochs</i>	46
Tabela 10 - Resultados do teste: CNN + LTC (sem camada <i>fully-connected</i>) com dados estáticos e 100 <i>epochs</i>	47
Tabela 11 - Resultados do teste: CNN + CfC (sem camada <i>fully-connected</i>) com dados estáticos e 100 <i>epochs</i>	47
Tabela 12 - Resultados do teste: CNN com dados estáticos, 100 <i>epochs</i> e método PSJ	48
Tabela 13 - Resultados do teste: CNN + LTC (com camada <i>fully-connected</i>) com dados estáticos, 100 <i>epochs</i> e <i>Glorot Uniform</i>	48
Tabela 14 - Resultados do teste: CNN + LTC (com camada <i>fully-connected</i>) com dados estáticos, 100 <i>epochs</i> e PSJ	49
Tabela 15 - Resultados do teste: CNN + CfC (com camada <i>fully-connected</i>) com dados estáticos, 100 <i>epochs</i> e <i>Glorot Uniform</i>	49
Tabela 16 - Resultados do teste: CNN + CfC (com camada <i>fully-connected</i>) com dados estáticos, 100 <i>epochs</i> e PSJ	50
Tabela 17 - Resultados do teste: LTC com dados temporais e 100 <i>epochs</i>	51
Tabela 18 - Resultados do teste: CfC com dados temporais e 100 <i>epochs</i>	51
Tabela 19 -- Resultados do teste: CNN + LTC (sem camada <i>fully-connected</i>) com dados temporais e 100 <i>epochs</i>	52
Tabela 20 - Resultados do teste: CNN + LTC (sem camada <i>fully-connected</i>) com dados temporais e 200 <i>epochs</i>	52
Tabela 21 - Resultados do teste: CNN + CfC (sem camada <i>fully-connected</i>) com dados temporais e 100 <i>epochs</i>	53
Tabela 22 - Resultados do teste: CNN + CfC (sem camada <i>fully-connected</i>) com dados temporais e 200 <i>epochs</i>	53

Tabela 23 - Resultados do teste: CNN + LTC (sem camada <i>fully-connected</i>) com dados temporais e 100 <i>epochs</i> para previsões de 1h (<i>load best weights</i>)	54
Tabela 24 - Resultados do teste: CNN + LTC (sem camada <i>fully-connected</i>) com dados temporais e 100 <i>epochs</i> para previsões de 2h (<i>load best weights</i>)	55
Tabela 25 - Resultados do teste: CNN + LTC (sem camada <i>fully-connected</i>) com dados temporais e 100 <i>epochs</i> para previsões de 3h (<i>load best weights</i>)	55
Tabela 26 - – Resumo dos resultados obtidos nos testes com dados estáticos.....	70
Tabela 27 - Resumo dos resultados obtidos nos testes com dados temporais.....	74
Tabela 28 - Resumo dos resultados obtidos nos testes com o modelo final	77

Lista de Siglas e Acrónimos

ANN	<i>Artificial Neural Network</i>
API	<i>Application Programming Interface</i>
CfC	<i>Closed-Form Continuous-Time</i>
CNN	<i>Convolutional Neural Network</i>
CPU	<i>Central Processing Units</i>
CSS	<i>Cascading Style Sheets</i>
GAN	<i>Generative Adversarial Network</i>
GPU	<i>Graphics Processing Units</i>
HTML	<i>Hypertext Markup Language</i>
IA	Inteligência Artificial
IPMA	Instituto Português do Mar e da Atmosfera
LTC	<i>Liquid Time-Constant</i>
LNN	<i>Liquid Neural Network</i>
MIT	Massachusetts Institute of Technology
NCP	<i>Neural Circuit Policies</i>
ODE	<i>Ordinary Differential Equation</i>
PSJ	<i>Perfect Sequences Joined</i>
ReLU	<i>Rectified Linear Unit</i>
SGD	<i>Stochastic Gradient Descent</i>
TPU	<i>Tensor Processing Units</i>

1. Introdução

A previsão meteorológica baseada em dados espaciais e temporais continua a ser um desafio no domínio da inteligência artificial (IA), sobretudo quando se procura modelar fenómenos altamente dinâmicos e não lineares como a precipitação.

Nos últimos anos, as *Artificial Neural Networks* (ANN) têm vindo a demonstrar um desempenho promissor na modelação de séries temporais e dados visuais [1], impulsionando avanços significativos em áreas como a climatologia, a robótica e a neurociência computacional. Em particular, a *Liquid Neural Network* (LNN), uma arquitetura recentemente proposta pelo Massachusetts Institute of Technology (MIT) [2], tem-se destacado pela sua capacidade de adaptação e generalização com um número relativamente reduzido de parâmetros, tornando-a uma alternativa atrativa para cenários em que os dados são ruidosos, escassos ou complexos.

O presente trabalho tem como objetivo aplicar e avaliar o desempenho das diferentes variantes de LNN disponibilizadas pela biblioteca *Neural Circuit Policies* (NCP) [3], desenvolvida pelo MIT, em dados locais de meteorologia na forma de imagens de radar [4]. Estes dados apresentam características espaço-temporais intrinsecamente complexas, sendo por isso um caso de estudo adequado para explorar o potencial da *Liquid Time-Constant* (LTC) e da *Closed-Form Continuous-Time* (CfC) em tarefas de previsão e deteção de padrões meteorológicos.

Além da avaliação isolada da LNN, propõe-se também a sua integração com *Convolutional Neural Networks* (CNN) tradicionais, com o intuito de investigar se as operações de convolução espacial – reconhecidas pela sua eficácia na extração de características relevantes em imagens – contribuem para uma melhoria do desempenho global das LNN. Esta abordagem híbrida procura conjugar a capacidade adaptativa das NCP com a robustez das CNN na análise visual, explorando sinergias que possam resultar em sistemas de previsão meteorológica mais precisos e eficientes.

Para suportar o treino eficaz dos modelos, recorreu-se a técnicas de processamento de dados e imagens, nomeadamente a normalização, transformação e reestruturação de sequências temporais. Considerando as limitações do conjunto de dados, foram também

implementadas estratégias de *Data Augmentation*, com o objetivo de aumentar a diversidade dos exemplos de treino e reduzir a dependência de padrões específicos. Para mitigar os efeitos de desequilíbrios nas classes de saída, foram aplicadas também técnicas de *oversampling* e *undersampling*, de modo a garantir uma distribuição mais uniforme das amostras ao longo das diferentes intensidades de precipitação.

Adicionalmente, procurou-se ainda otimizar o desempenho dos modelos logo nas fases iniciais de treino, testando o método proprietário *Perfect Sequences Joined* (PSJ) para inicialização de pesos, desenvolvido pelo Professor Doutor João Da Silva Pereira [5]. No entanto, não foi possível obter resultados conclusivos.

Este estudo pretende, assim, contribuir para o corpo de conhecimento atual sobre redes neurais não convencionais, avaliando o seu potencial em contextos do mundo real, nomeadamente na previsão de fenómenos atmosféricos a partir de dados visuais. Os resultados obtidos poderão fornecer indicações valiosas sobre as capacidades e limitações destas arquiteturas, bem como sobre estratégias de integração com modelos clássicos de *computer vision*.

2. Enquadramento teórico

2.1. Redes Neuronais

As *Artificial Neural Networks* (ANN) são modelos computacionais inspirados nas redes de neurónios do cérebro humano, sendo compostas por unidades de processamento denominadas neurónios artificiais. Estas redes têm como principal objetivo aprender a partir de dados, permitindo modelar relações complexas e não lineares entre variáveis, algo que as torna particularmente eficazes em tarefas como classificação, regressão, reconhecimento de padrões e análise de séries temporais [6].

Cada neurónio recebe como entrada uma soma ponderada dos valores provenientes da camada anterior, à qual é aplicada uma função de ativação (Figura 1) [7]. As funções de ativação mais comuns incluem *Rectified Linear Unit* (ReLU), *Sigmoid* e *Tanh*, e a sua principal função é introduzir não-linearidade no modelo, permitindo que a rede aprenda representações complexas dos dados.

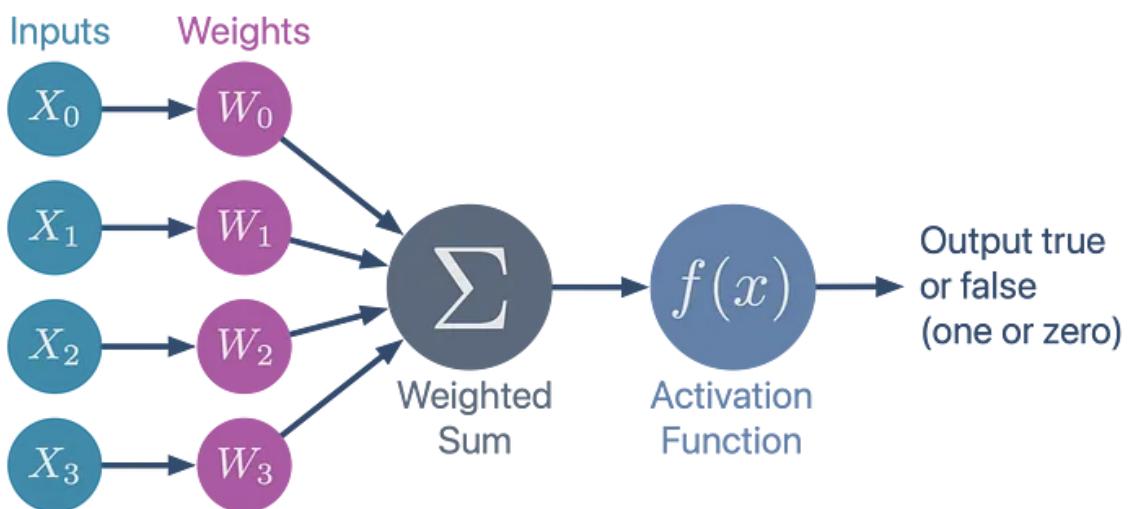


Figura 1 - Componentes de um percepção

O processo de aprendizagem ocorre por meio de um processo iterativo denominado *backpropagation*, no qual a rede ajusta os seus parâmetros (os pesos das ligações entre os neurónios) com base no erro entre a saída prevista e a saída desejada (Figura 2) [8]. Esse erro é propagado de volta pelas camadas da rede, e os pesos são atualizados com o objetivo de minimizar, utilizando algoritmos de otimização como o *Stochastic Gradient Descent* (SGD) ou o *Adam*.

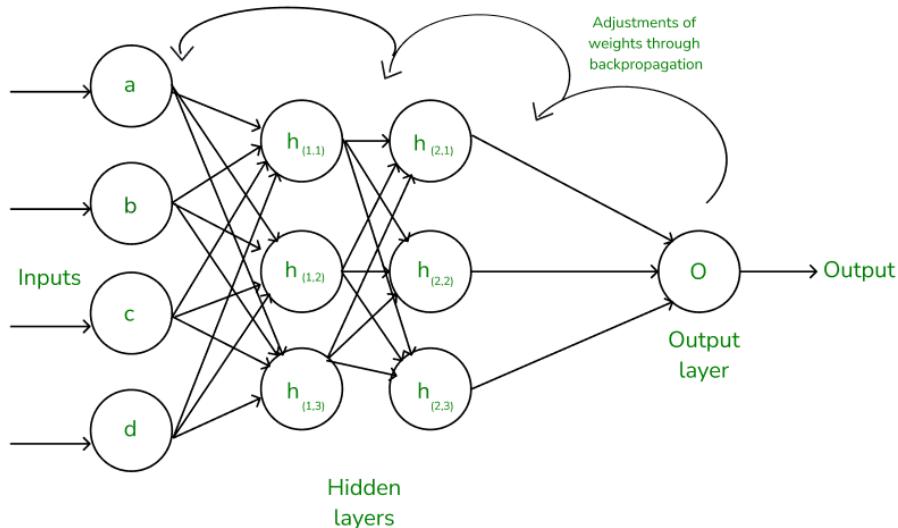


Figura 2 - Simples ilustração de backpropagation

2.2. Deep Learning

O termo *Deep Learning* refere-se a uma subárea das ANN, caracterizada pelo uso de redes neurais profundas, ou seja, redes compostas por múltiplas camadas ocultas entre a camada de entrada e a camada de saída (Figura 3) [9]. Estas redes são designadas como “profundas” devido à sua arquitetura com elevado número de camadas, o que permite a modelação de funções altamente complexas e a aprendizagem de representações hierárquicas de dados. Esta profundidade estrutural é uma das principais razões pelas quais o *Deep Learning* tem demonstrado um desempenho superior em diversas tarefas.

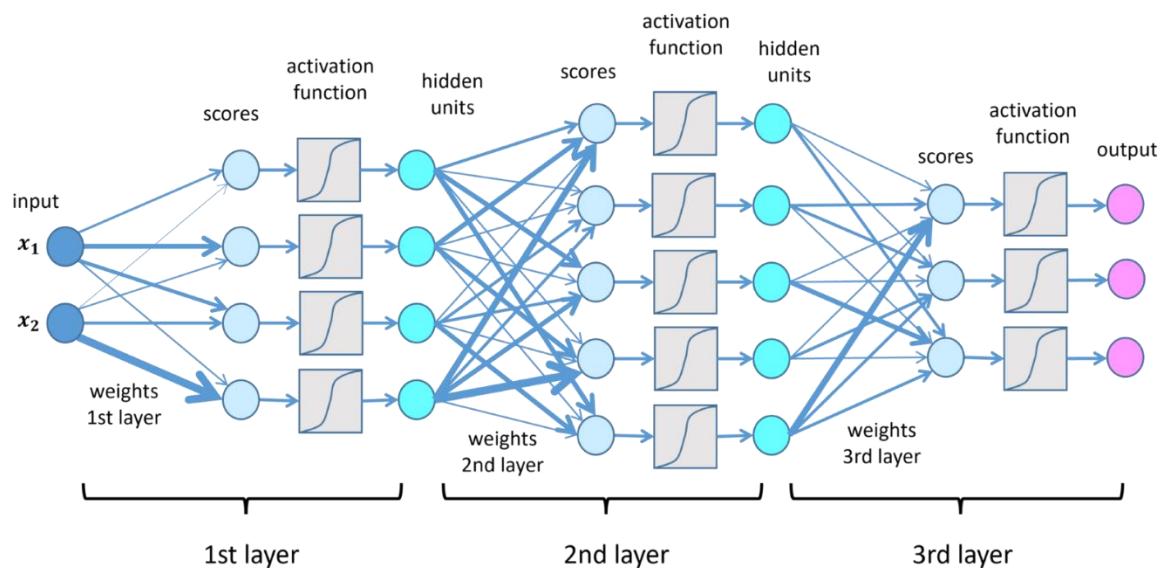


Figura 3 - Estrutura de redes neurais profundas

No entanto, o treino de redes profundas apresenta desafios significativos, decorrentes do elevado número de parâmetros e da complexidade computacional envolvida. Entre esses desafios destaca-se o *vanishing gradient*, que ocorre quando os gradientes nas camadas mais profundas se tornam extremamente pequenos, dificultando a aprendizagem [10]. Para mitigar estas limitações, foram desenvolvidos algoritmos de otimização mais avançados, que permitem ajustar os pesos de forma mais eficiente, acelerando a convergência e melhorando o desempenho da rede.

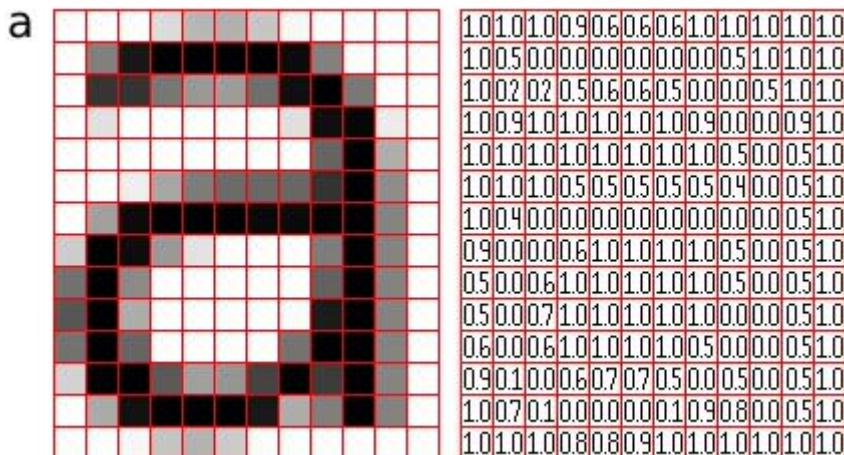
A implementação de técnicas de *Deep Learning* tem levado a avanços notáveis em múltiplas áreas, como o reconhecimento de imagem, a tradução automática, os sistemas de recomendação e a deteção de fraudes. Além disso, tecnologias emergentes, como os veículos autónomos, também recorrem extensivamente a esta abordagem para processar e interpretar dados sensoriais em tempo real.

Ao explorar o potencial das redes profundas, é possível desenvolver sistemas inteligentes capazes de executar tarefas complexas anteriormente reservadas à intervenção humana. O *Deep Learning* tem, assim, desempenhado um papel fundamental na revolução da automação e na evolução de soluções analíticas em diversos setores.

2.3. Convolutional Neural Networks

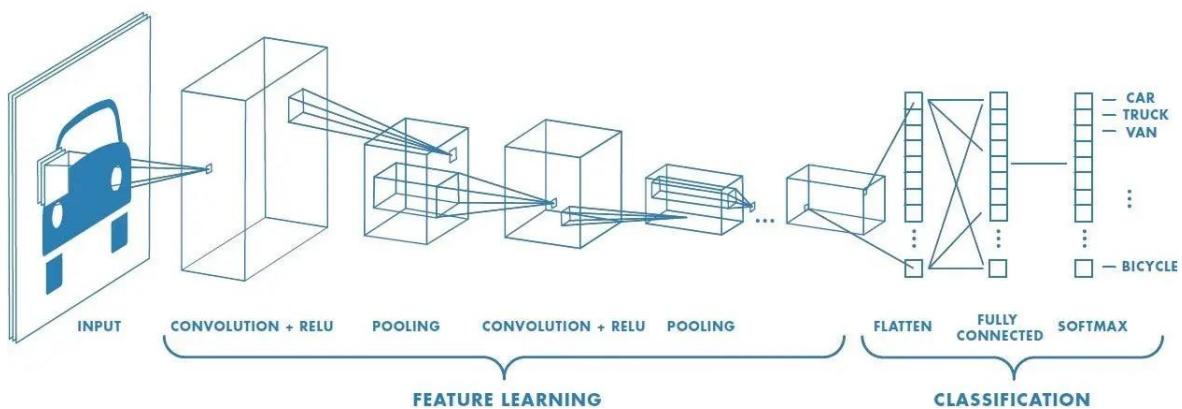
Uma *Convolutional Neural Network*, também conhecida como CNN ou *ConvNet*, é uma classe de redes neurais artificiais especialmente concebida para o processamento de dados com estrutura matricial, como imagens [11]. A sua arquitetura foi inspirada no funcionamento do córtex visual dos animais, sendo particularmente eficaz em tarefas de reconhecimento e classificação de padrões visuais.

Uma imagem digital pode ser representada como uma matriz de valores numéricos, onde cada elemento (píxel) indica a intensidade e, dependendo do canal, a cor da imagem (Figura 4) [12]. Esta estrutura torna-se ideal para ser processada por *convolutional layers*, que constituem a principal inovação das CNN.

**Figura 4** - Representação de uma imagem como matriz de pixels

As *convolutional layers* aplicam filtros (ou *kernels*) que percorrem os dados de entrada para extrair características locais, como contornos, texturas e padrões espaciais. Cada filtro é aprendido automaticamente durante o processo de treino, permitindo à rede identificar progressivamente características mais abstratas e complexas.

Frequentemente, as *convolutional layers* são seguidas por *pooling layers*, que reduzem a dimensionalidade espacial dos dados, preservando as informações mais relevantes e contribuindo para a redução do número de parâmetros e da complexidade computacional (Figura 5) [13].

**Figura 5** - Estrutura de uma *Convolutional Neural Network*

Esta combinação de extração eficiente de características locais e redução da dimensionalidade permite às CNN aprender representações hierárquicas dos dados de forma eficaz, sendo amplamente utilizadas em aplicações como reconhecimento facial, análise de imagens médicas e condução autónoma.

Cada camada da arquitetura de uma CNN possui um papel específico no processo de extração e transformação dos dados. Nas secções seguintes, são descritas detalhadamente as camadas mais comuns utilizadas neste tipo de rede.

2.3.1. Convolutional Layer

A *Convolutional Layer*, ou *Conv2D*, é um dos componentes fundamentais de uma CNN. É comum iniciar uma CNN com esta camada, pois permite reduzir significativamente a complexidade dos dados de entrada, passando para as camadas seguintes apenas as características mais pertinentes.

O principal parâmetro de uma *Convolutional Layer* é o filtro, também conhecido como *kernel*. Este é uma matriz de pesos, geralmente de dimensões reduzidas (como 2x2, 3x3 ou 5x5), que percorre a matriz de entrada (por exemplo, uma imagem), realizando uma operação de convolução [14]. Esta operação consiste em multiplicar os valores dos píxeis cobertos pelo filtro pelos valores do próprio filtro e somar os resultados, gerando assim um novo valor que será inserido no mapa de características (*feature map*), que constitui o *output* da camada (Figura 6) [15].

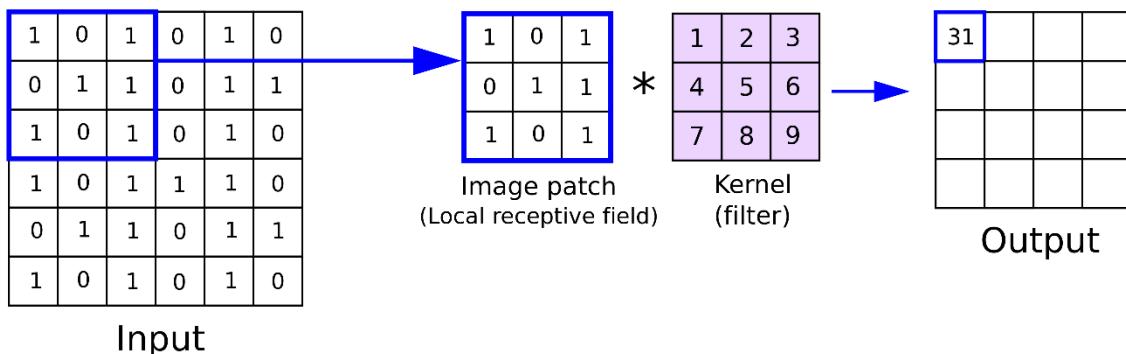


Figura 6 - Representação da operação de convolução

O processo é repetido em diferentes posições da imagem à medida que o filtro desliza (processo conhecido como *stride*), podendo ser utilizado *padding* para manter as dimensões de saída (Figura 7) [13].

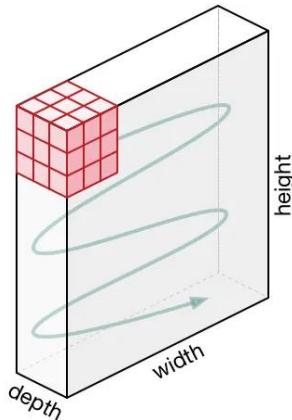


Figura 7 - Movimento do kernel

Após a convolução, é habitualmente aplicada uma função de ativação – como a ReLU – ao mapa de características resultante, introduzindo não-linearidade no modelo e permitindo realçar ainda mais os padrões relevantes detetados pelo filtro.

Esta camada permite, assim, identificar automaticamente características importantes da imagem, como contornos, texturas ou bordas, contribuindo para a construção de representações mais abstratas nos níveis superiores da rede.

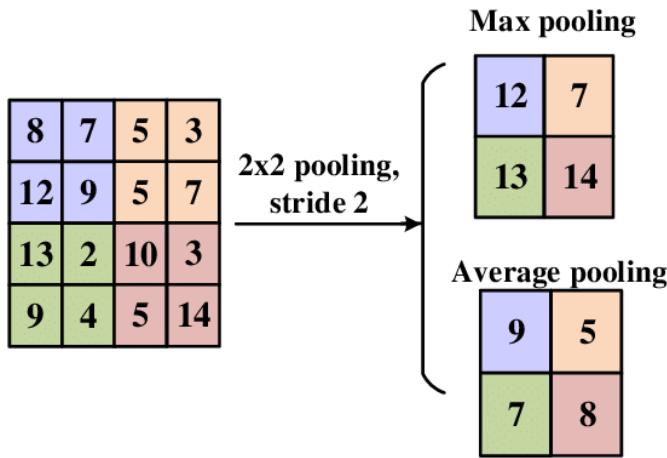
2.3.2. *Pooling Layer*

A *Pooling Layer* é geralmente colocada após uma *Convolutional Layer* e tem como objetivo reduzir as dimensões espaciais do mapa de características, preservando as informações mais relevantes. Esta redução contribui para um menor custo computacional e para a diminuição do risco de *overfitting*, ao mesmo tempo que proporciona alguma invariância à translação dos dados de entrada [14].

Tal como na convolução, aplica-se uma janela deslizante, definida por um parâmetro chamado *pool size*, mas, ao contrário das *convolutional layers*, esta operação não envolve pesos treináveis.

Existem dois tipos mais comuns de *pooling* (Figura 8) [16]:

- *Max Pooling*: seleciona o valor máximo dentro da região coberta pela janela, conservando a característica mais prominente daquela região.
- *Average Pooling*: calcula a média dos valores da região coberta, resultando numa representação suavizada dos dados.

**Figura 8 - Tipos de pooling**

A aplicação sucessiva de *pooling* ao longo da rede permite uma extração mais abstrata e robusta de características. No entanto, é importante reconhecer que este processo também implica uma perda de detalhe, podendo eliminar informações sutis que poderiam ser relevantes para certas tarefas [17].

2.3.3. *Normalization Layer*

A *Normalization Layer* tem como função normalizar as ativações da rede neuronal durante o treino, com o objetivo de estabilizar o processo de aprendizagem, acelerar a convergência e melhorar a generalização do modelo. Este tipo de camada tornou-se especialmente popular com a introdução da *Batch Normalization*, amplamente utilizada em arquiteturas modernas.

Na *Batch Normalization*, os valores das ativações são normalizados de forma que apresentem média próxima de zero e variância unitária (Figura 9) [18], considerando cada *mini-batch* de treino. Em seguida, dois parâmetros treináveis – γ (escala) e β (deslocamento) – são aplicados para restaurar a capacidade da rede de representar distribuições arbitrárias. Esta técnica ajuda a mitigar o problema do *vanishing gradient* e permite usar taxas de aprendizagem mais elevadas.

Em suma, a normalização contribui para tornar o treino mais eficiente e robusto, podendo também atuar como uma forma de regularização que reduz o risco de *overfitting*.

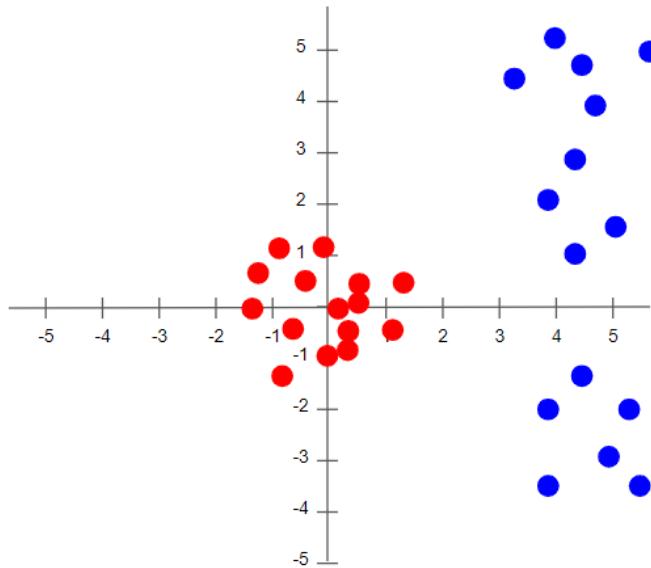


Figura 9 - Dados normalizados (vermelho) vs Dados não normalizados (azul)

2.3.4. Dropout Layer

A *Dropout Layer* é uma técnica de regularização utilizada para reduzir o risco de *overfitting* durante o treino de redes neurais. Esta camada atua desativando aleatoriamente, em cada iteração, uma percentagem predefinida de neurónios (normalmente entre 20% e 50%) (Figura 10) [19]. Isso impede que a rede se torne excessivamente dependente de determinadas ativações, forçando-a a aprender representações mais robustas e distribuídas.

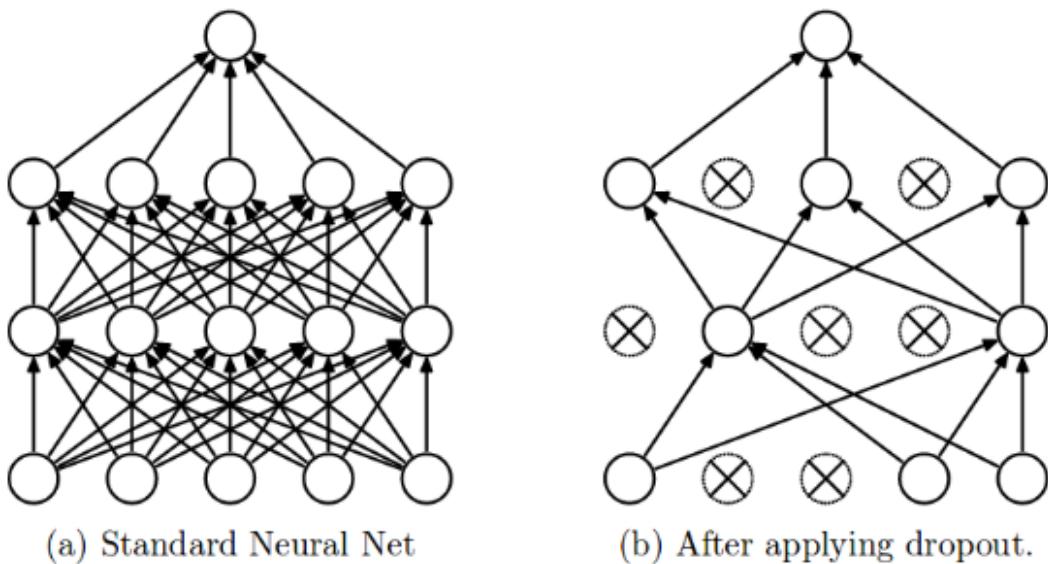


Figura 10 - *Dropout* aplicado a uma típica rede neuronal

Durante o treino, os neurónios desativados não contribuem para a propagação direta nem para o cálculo do gradiente. Já durante a fase de validação, todos os neurónios permanecem ativos, e os seus valores são escalados adequadamente para compensar a ausência de *dropout*.

Esta técnica é frequentemente aplicada entre camadas *fully-connected* e, em alguns casos, também após *convolutional layers*. O seu uso tem demonstrado melhorias significativas na generalização de modelos, especialmente em redes profundas.

2.3.5. Flatten Layer

A *Flatten Layer* é uma camada de transição responsável por converter a saída multidimensional das *convolutional* e *pooling layers* – geralmente representada como um tensor 3D – num vetor unidimensional (1D) (Figura 11) [20]. Esta transformação é necessária para que os dados possam ser processados por camada densamente conectadas (*fully-connected layers*), que exigem vetores de entrada [21].

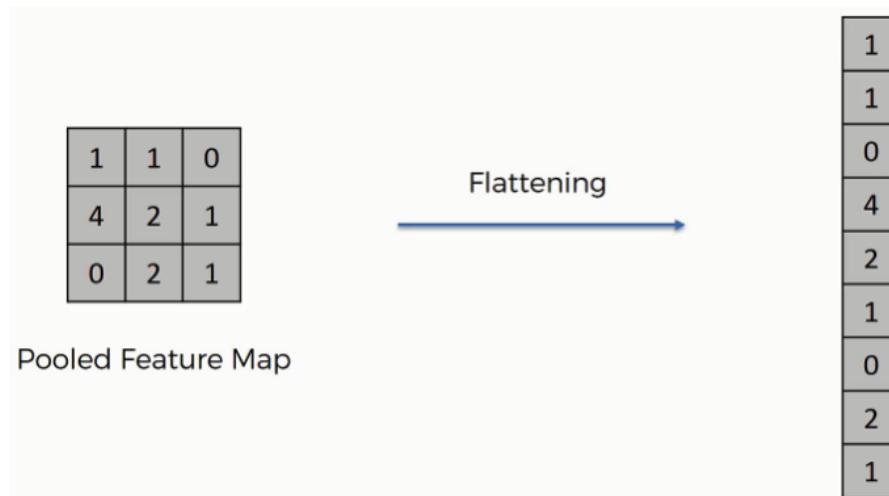


Figura 11 - Representação do processo de *flattening*

A função desta camada é, portanto, “linearizar” o mapa de características, reorganizando todos os valores num único vetor contínuo. Esta operação não altera os valores nem realiza qualquer tipo de cálculo, servindounicamente para adequar a forma dos dados ao formato exigido pelas camadas posteriores, tipicamente associadas à tomada de decisão, como a classificação final.

2.3.6. Fully-connected Layer

A *Fully-connected Layer* é geralmente posicionada nas fases finais de uma CNN e tem como função principal realizar a interpretação das características extraídas pelas camadas anteriores, conduzindo à decisão final do modelo. Esta camada estabelece ligações entre todos os neurónios da camada anterior e todos os seus próprios neurónios, funcionando como uma rede densa clássica [22].

A estrutura desta camada pode variar conforme o objetivo da rede. Em tarefas de classificação binária, a *Fully-connected Layer* geralmente contém um único neurónio final com função de ativação *Sigmoid*. A saída será um valor contínuo entre 0 e 1, representando a probabilidade de o *input* pertencer à classe-alvo (Figura 12) [23].

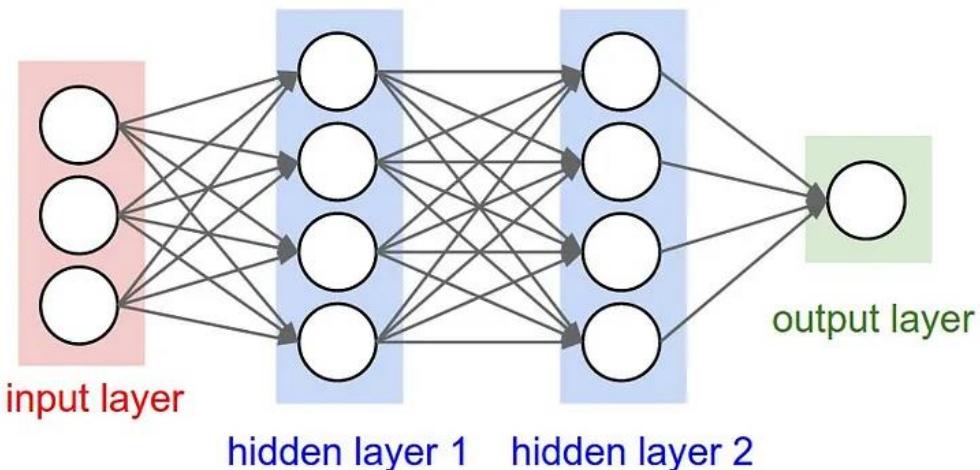


Figura 12 - Fully-connected Layer para problemas de classificação binária

Em tarefas de classificação multi-classe, a camada final contém um número de neurónios igual ao número de classes possíveis. Cada neurónio está associado a uma classe e a função de ativação mais comum neste contexto é a *Softmax*, que transforma os valores de saída numa distribuição de probabilidades, assegurando que a soma total seja igual a 1. Desta forma, é possível identificar a classe mais provável de forma clara e eficiente (Figura 13) [24].

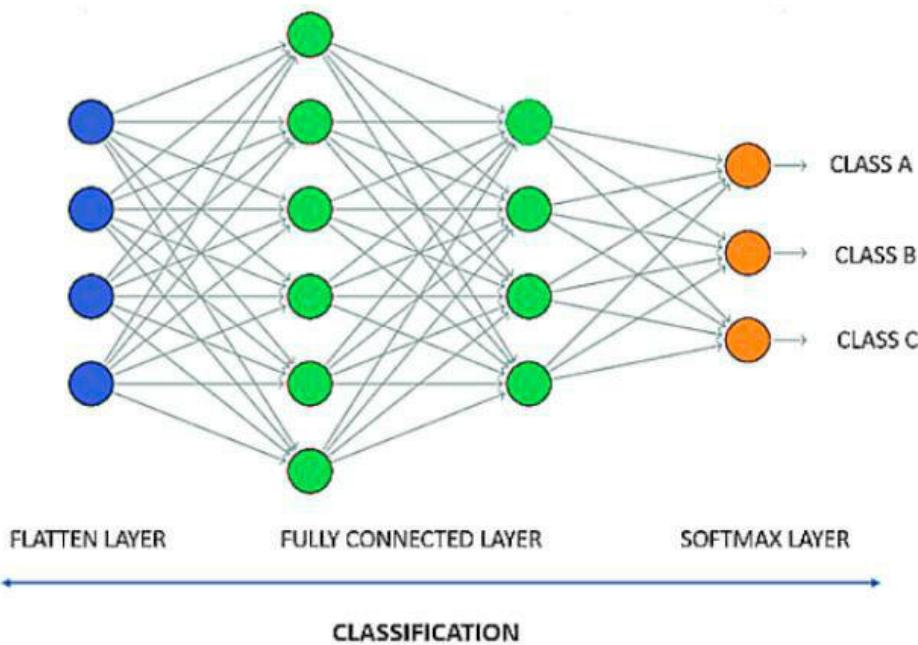


Figura 13 - Fully-connected Layer para problemas de classificação multi-classe

2.4. Liquid Neural Networks

As *Liquid Neural Networks* (LNN) representam um avanço recente no campo do *Deep Learning*, com inspiração em princípios da neurociência biológica. Estas redes foram desenvolvidas para colmatar algumas limitações das redes neuronais tradicionais, em especial no que diz respeito à generalização a partir de poucos dados, capacidade de adaptação em tempo real e maior eficiência computacional.

Concebidas no MIT, as LNN assentam numa estrutura dinâmica onde cada neurónio segue uma *Ordinary Differential Equation* (ODE) [25], modelando o seu comportamento ao longo do tempo. A principal característica diferenciadora destas redes é a plasticidade contínua: em vez de utilizarem funções de ativação fixas, os neurónios líquidos ajustam-se dinamicamente de forma não-linear, reagindo ao histórico dos dados de entrada. Isto confere-lhes uma flexibilidade notável para aprender padrões temporais complexos.

A biblioteca *Neural Circuit Policies* (NCP) desenvolvida no MIT [26], fornece implementações prática deste tipo de redes, incluindo duas variantes principais:

- *Liquid Time-Constant (LTC) Networks*: introduzem um conceito inovador, onde cada neurónio possui uma constante de tempo aprendível, que regula a rapidez com que este responde aos estímulos recebidos. Tal abordagem permite que a rede module dinamicamente a sua memória temporal, facilitando a captação de dependências de curto e longo prazo. Esta característica torna as LTC eficazes em ambientes com variação rápida no tempo, como séries meteorológicas ou robótica dinâmica [27].
- *Closed-Form Continuous-Time (CfC) Networks*: oferecem uma alternativa matemática mais eficiente às LTC, ao resolverem analiticamente as equações diferenciais dos neurónios. Isto elimina a necessidade de métodos numéricos iterativos, como *Euler* ou *Runge-Kutta*, reduzindo o custo computacional e aumentando a estabilidade. Os neurónios CfC são capazes de integrar sinais temporais de forma contínua e precisa, o que os torna adequados para tarefas com dados de alta resolução temporal [28].

As LNN demonstram elevado desempenho em tarefas com dados temporais ruidosos ou escassos, tais como controlo robótico, navegação autónoma, e mais recentemente, previsão meteorológica com recurso a séries temporais.

O comportamento adaptativo destas redes torna-as particularmente promissoras em contextos onde os dados se alteram rapidamente, ou onde os modelos necessitam de tomar decisões em tempo real. No entanto, dado o seu caráter recente, ainda há muito por explorar em termos de compreensão teórica, limitações práticas e integração eficaz com outras arquiteturas, como as CNN.

Em termos estruturais, uma rede líquida pura tende a ser mais simples do que uma CNN, dado que a componente essencial é uma camada especializada conhecida como reservatório líquido, que encapsula a dinâmica dos neurónios líquidos (Figura 14) [29]. Esta camada pode ser combinada com outras, como camadas densas, para formar arquiteturas híbridas mais completas.

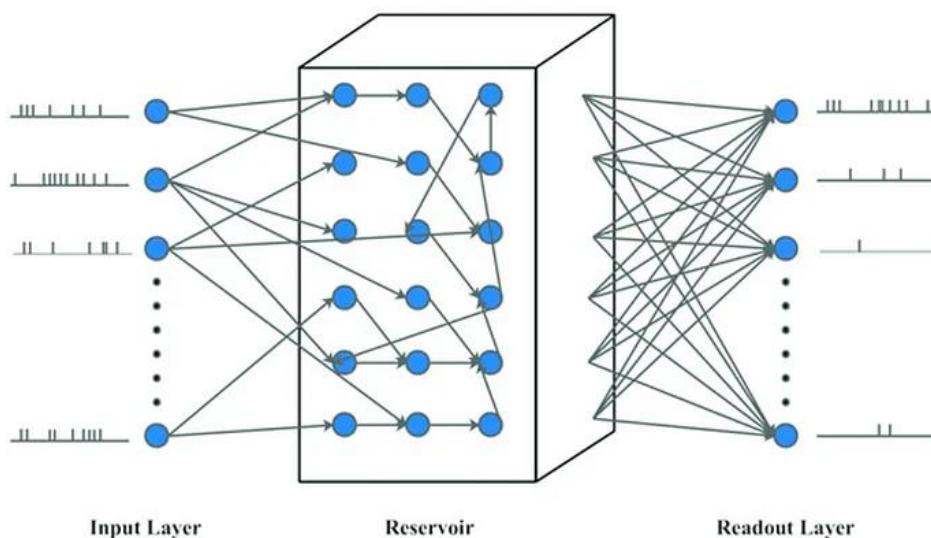


Figura 14 - Estrutura de uma *Liquid Neural Network*

2.5. Data Augmentation

Data Augmentation refere-se a um conjunto de técnicas utilizadas para aumentar artificialmente a quantidade e diversidade dos dados disponíveis para o treino de modelos de *machine learning*, sem a necessidade de recolher novos dados. Esta abordagem é especialmente útil quando se trabalha com conjuntos de dados limitados, contribuindo para reduzir o risco de *overfitting* e melhorar a generalização do modelo [30].

No contexto da previsão baseada em imagens, as técnicas de *Data Augmentation* envolvem transformações que preservem o conteúdo semântico da imagem, mas que introduzem variações visuais (Figura 15) [31]. Entre as transformações mais comuns incluem-se:

- Rotações aleatórias;
- Reflexões horizontais e verticais (*flipping*);
- Alterações de brilho, contraste ou saturação;
- Adição de ruído gaussiano;
- Cortes aleatórios (*random cropping*);
- Redimensionamento ou escalamento;
- Translações e distorções geométricas.

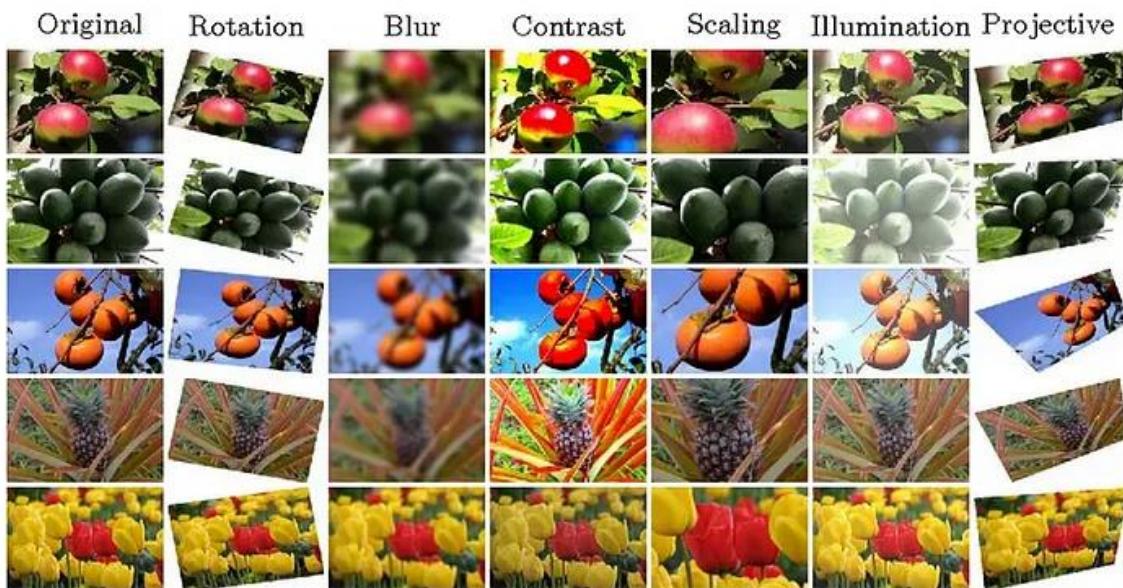


Figura 15 - Técnicas de *Image Augmentation*

Estas transformações simulam variações naturais que podem ocorrer nos dados do mundo real, permitindo que o modelo aprenda a ser invariante a essas mudanças. Em tarefas visuais, como classificação ou previsão com base em imagens, o *Data Augmentation* é uma técnica fundamental para melhorar a robustez e o desempenho do modelo, especialmente quando o número de amostras disponíveis é reduzido.

A seguir, diferencia-se o conceito de dados aumentados de dados sintéticos e explora-se um conjunto mais específico de técnicas de *Data Augmentation*.

2.5.1. Dados aumentados (*augmented*) vs. Dados sintéticos

Embora ambos contribuam para aumentar a quantidade de dados disponíveis para treino, os dados aumentados e os dados sintéticos têm origens e propósitos distintos.

Dados aumentados são obtidos a partir de dados reais existentes, através da aplicação de pequenas transformações que não alteram o significado semântico da amostra. No caso da

Image Augmentation, estas transformações incluem, por exemplo, rotações, reflexões, alterações de brilho ou contraste, entre outras. O objetivo é diversificar o conjunto de treino mantendo a integridade das classes, o que ajuda a melhorar a generalização do modelo.

Por outro lado, dados sintéticos são gerados artificialmente, sem partir de exemplos reais. Esta geração pode ser feita através de técnicas estatísticas, simulações ou redes neurais avançadas, como as *Generative Adversarial Networks* (GAN) [32]. Os dados sintéticos tentam imitar as características dos dados reais, sendo especialmente úteis quando os dados originais são escassos, difíceis de obter ou apresentam problemas de privacidade.

2.6. *Oversampling e Undersampling*

Conjuntos de dados desequilibrados são aqueles em que, pelo menos uma classe, possui significativamente menos amostras do que as outras. Este tipo de distribuição é comum em projetos de *data science* e *data mining*, e pode comprometer o desempenho de algoritmos de classificação, que tendem a favorecer a classe maioritária.

Para lidar com este problema, existem técnicas de rebalanceamento de dados, sendo as mais utilizadas *oversampling* e *undersampling* [33] [34].

O *oversampling* consiste em aumentar o número de amostras da classe minoritária, de forma a equilibrar a distribuição das classes. Esta técnica pode ser aplicada de duas formas principais:

- Duplicação de amostras: consiste na replicação direta de exemplos da classe minoritária já existentes no conjunto de dados;
- Geração de amostras sintéticas: envolve a criação de novas observações artificiais, geralmente através da introdução de pequenas variações nas amostras existentes.

O *undersampling*, por outro lado, reduz o número de amostras da classe maioritária, de modo a equilibrar o conjunto de dados. Isto pode ser feito selecionando aleatoriamente uma subamostra da classe com mais exemplos, descartando o excesso de dados.

A vantagem do *oversampling* em relação ao *undersampling* é, o facto do *oversampling* não diminuir o tamanho do conjunto de dados. Isto pode ser importante quando o nosso

modelo é complexo e precisamos de ter a certeza que o modelo tem dados suficientes para treinar.

A vantagem do *undersampling* em relação ao *oversampling* é, o facto de este não introduzir informação redundante no conjunto de dados. Isto é importante pois, quando se duplica observações existentes estamos a fazer com que os padrões das observações duplicadas estejam mais presentes no conjunto de dados, o que pode gerar *overfitting*.

2.7. Overfitting e Underfitting

Em *machine learning*, é fundamental encontrar o equilíbrio entre a capacidade de um modelo aprender a partir dos dados de treino e a sua capacidade de generalizar para dados novos. Dois dos problemas mais comuns nesse contexto são o *overfitting* e o *underfitting* [35].

O *overfitting* ocorre quando um modelo aprende excessivamente os dados de treino, captando não apenas os padrões relevantes, mas também ruído e as anomalias presentes nesses dados (Figura 16) [36]. Como resultado, o modelo apresenta um excelente desempenho nos dados de treino, mas falha em generalizar para dados nunca antes vistos.

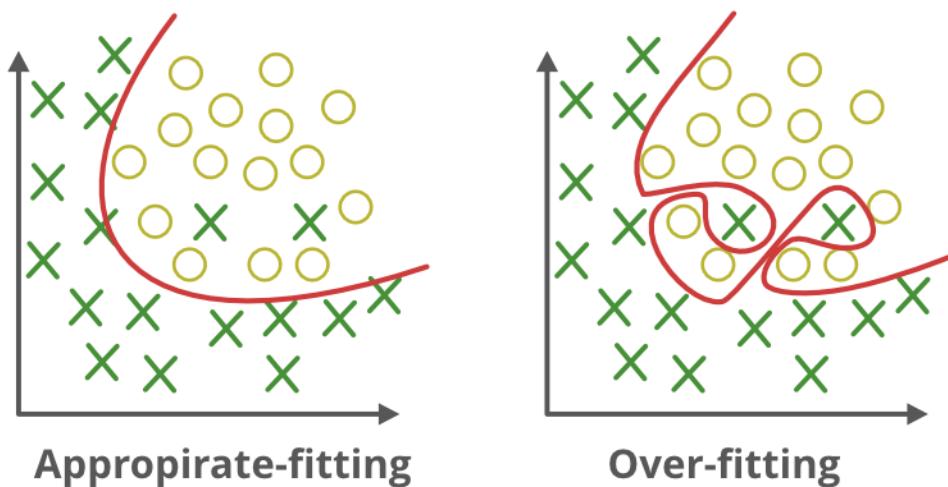


Figura 16 - Exemplo de *overfitting*

Este fenómeno é especialmente comum em cenários com:

- Conjuntos de dados pequenos ou desequilibrados;
- Modelos demasiado complexos para a quantidade de dados disponível;
- Treino por um número excessivo de épocas.

Em redes neurais profundas, o *overfitting* pode manifestar-se quando o modelo continua a melhorar no treino, mas piora nos dados de validação.

Estratégias para mitigar o *overfitting*:

- Regularização (por exemplo, *dropout* ou *weight decay*);
- *Early stopping* (interromper o treino quando o desempenho em validação começa a piorar);
- *Cross-validation* (para avaliar a capacidade de generalização do modelo);
- *Data augmentation* (aumentar artificialmente a variedade dos dados de treino).

Estas técnicas forçam o modelo a generalizar melhor, evitando o ajuste demasiado preciso aos dados de treino.

O *underfitting* é o oposto, e ocorre quando o modelo não consegue capturar a complexidade dos dados, apresentando baixo desempenho tanto nos dados de treino como nos de validação (Figura 17) [36]. Isto indica que o modelo tem capacidade expressiva insuficiente para aprender os padrões subjacentes ao problema.

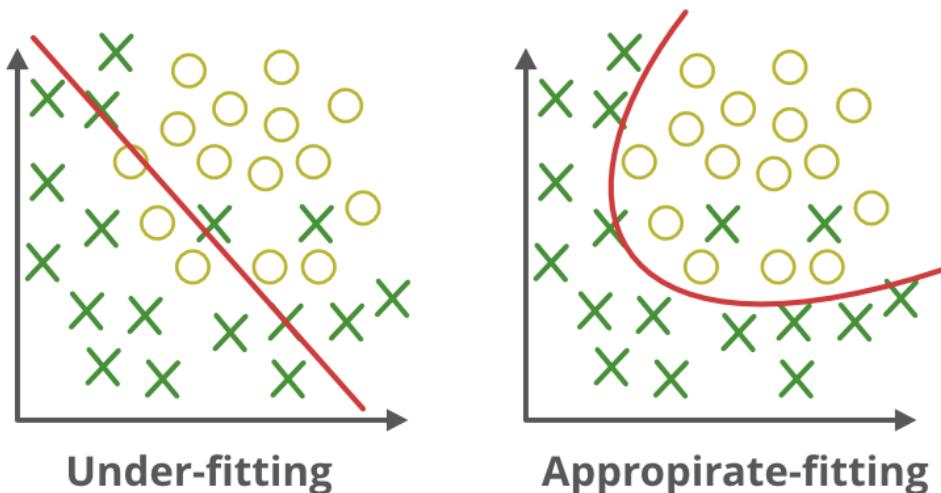


Figura 17 - Exemplo de *underfitting*

As causas mais comuns de *underfitting* incluem:

- Modelos demasiado simples;
- Redes neurais com poucos parâmetros;
- Treino com poucas épocas;
- Hiperparâmetros mal ajustados (por exemplo, uma taxa de aprendizagem elevada ou regularização excessiva).

Nestes casos, o modelo falha em aprender até mesmo os padrões mais óbvios.

Estratégias para mitigar o *underfitting*:

- Aumentar a complexidade do modelo (mais camadas, mais neurónios);
- Treinar durante mais épocas;
- Ajustar hiperparâmetros (por exemplo, diminuir a regularização ou ajustar a taxa de aprendizagem);
- Melhorar a qualidade e representatividade dos dados de entrada.

3. Descrição do processo

3.1. Ferramentas utilizadas

A elaboração de projetos de grande magnitude e de elevada complexidade técnica requer a utilização de um conjunto diversificado de ferramentas, que auxiliam em diferentes fases do desenvolvimento. Desde ambientes de programação interativa até plataformas de versionamento e *frameworks* de visualização, estas ferramentas permitem otimizar o fluxo de trabalho, facilitar a colaboração e garantir a reproduzibilidade dos resultados. No presente projeto, foram selecionadas ferramentas adequadas à análise de dados, ao treino de modelos de *machine learning* e ao desenvolvimento de uma interface de apresentação de resultados.

3.1.1. Jupyter Notebook

O *Jupyter Notebook* é uma aplicação *web* de código aberto que permite criar e compartilhar documentos interativos que combinam código executável, visualizações, equações matemáticas e texto descritivo [37]. É amplamente utilizado nas áreas de ciência de dados, *machine learning*, investigação científica e educação, pois oferece uma maneira conveniente de combinar código executável com elementos narrativos (Figura 18).

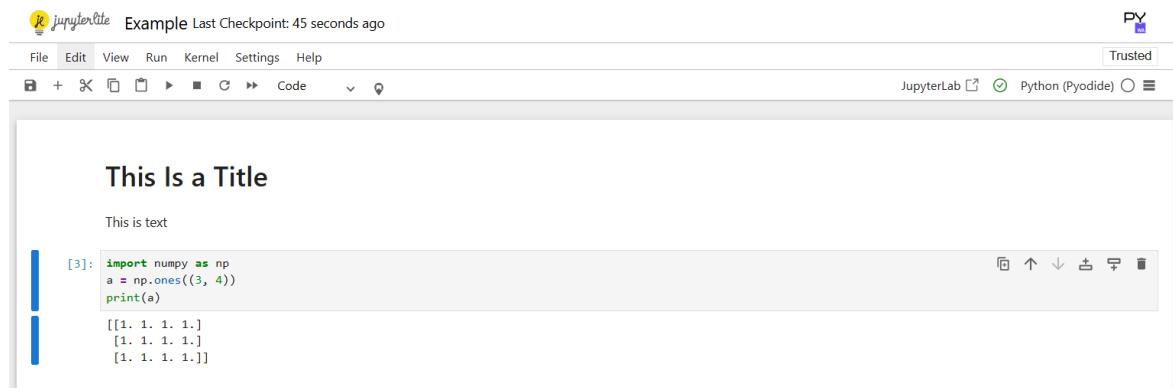


Figura 18 - Exemplo de um *Jupyter Notebook*

A sua principal característica é o suporte de células, que são unidades independentes onde é possível escrever e executar código de forma isolada, tornando o processo de desenvolvimento e teste de algoritmos mais dinâmico e modular. Esta estrutura facilita a deteção de erros, o ajuste de parâmetros e a avaliação incremental de resultados, promovendo uma maior produtividade e clareza no raciocínio analítico.

No contexto deste projeto, o *Jupyter Notebook* foi utilizado para a preparação e exploração dos dados meteorológicos, o treino e validação dos modelos de redes neurais,

bem como para a análise gráfica dos resultados. A sua interface intuitiva e modular facilitou a organização das experiências, permitindo uma iteração eficiente sobre diferentes configurações e abordagens.

3.1.2. *Flask*

O *Flask* é um *micro-framework web* desenvolvido em *Python*, reconhecido pela sua leveza, simplicidade e flexibilidade (Figura 19) [38]. Ao contrário de outros *frameworks* mais pesados, o *Flask* não impõe uma estrutura rígida ao projeto, permitindo aos programadores desenvolver aplicações *web* de forma modular e personalizada. Esta abordagem torna-o particularmente adequado para projetos de pequena e média escala, bem como para a criação de *Application Programming Interfaces* (API) que disponibilizam modelos de *machine learning* como serviços.

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

Figura 19 - Exemplo de uma *app* em *Flask*

Adicionalmente, o *Flask* conta com uma documentação extensa e acessível, acompanhada de exemplos práticos e tutoriais progressivos, o que facilita tanto a introdução como o aprofundamento técnico por parte dos programadores.

No âmbito deste projeto, o *Flask* foi utilizado no desenvolvimento do *backend* da aplicação *web* responsável pela disponibilização dos resultados das previsões de chuva. Através de uma API desenvolvida com esta ferramenta, foi possível disponibilizar as imagens de radar em tempo real, assim como as previsões produzidas pelo modelo de rede neuronal. A sua integração simples com bibliotecas *Python* de *machine learning*, garantiu um fluxo eficiente entre o processamento dos dados e a sua apresentação ao utilizador final.

3.1.3. *Vue.js*

O *Vue.js* é um *framework* progressivo de *JavaScript* utilizado para o desenvolvimento de interfaces *web* interativas e dinâmicas [39]. Baseia-se em tecnologias *web* padrão como, *Hypertext Markup Language* (HTML), *Cascading Style Sheets* (CSS) e *JavaScript*, e adota um modelo de programação declarativo e baseado em componentes. Esta arquitetura facilita

a organização do código, promovendo a sua reutilização e simplificando a manutenção da interface (Figura 20).

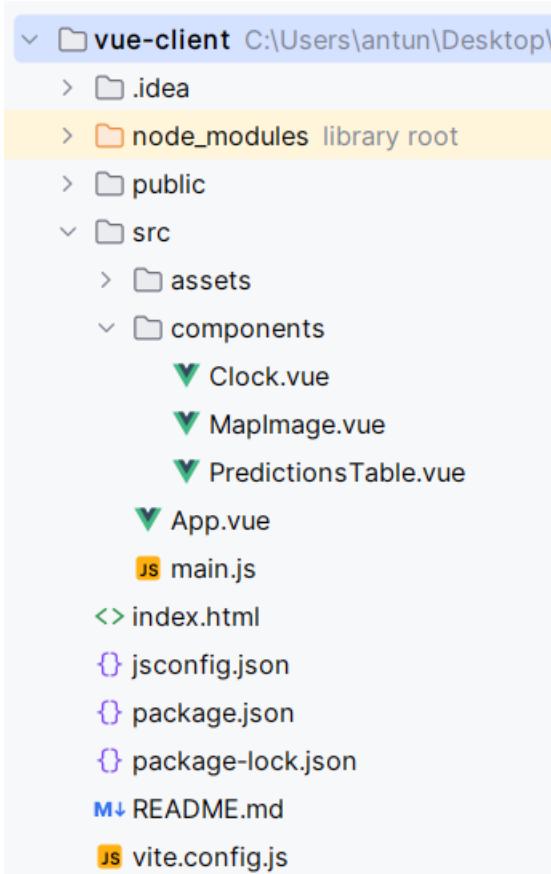


Figura 20 - Exemplo de uma *app* em *Vue.js*

Entre as suas principais vantagens destacam-se a curva de aprendizagem acessível e a documentação clara e abrangente. O *Vue.js* disponibiliza ainda um ecossistema robusto que inclui ferramentas para roteamento de páginas, gestão de estado e integração eficiente com API RESTful.

Neste projeto, o *Vue.js* foi utilizado no desenvolvimento da interface gráfica da aplicação *web*, permitindo ao utilizador consultar de forma intuitiva as previsões de precipitação e as imagens de radar associadas. A comunicação com o *backend*, implementado em *Flask*, foi realizada através de chamadas assíncronas a uma API, possibilitando a atualização dinâmica do conteúdo sem necessidade de recarregamento da página.

3.1.4. *TensorFlow*

Amplamente utilizado por cientistas de dados e programadores, o *TensorFlow* é uma biblioteca de código aberto desenvolvida pela equipa da *Google Brain* para fins de pesquisa em *machine learning* e redes neurais [40]. Concebido para suportar operações

matemáticas de alto desempenho, permite a construção e execução de redes neurais em ambientes heterogéneos, incluindo *Central Processing Units* (CPU), *Graphics Processing Units* (GPU) e *Tensor Processing Units* (TPU), com elevado grau de escalabilidade.

Uma das suas principais características é o sistema de grafos computacionais, que possibilita a representação e otimização de modelos complexos de forma eficiente. Além disso, a biblioteca fornece ainda o *Keras*, uma API de alto nível que torna a resolução de problemas de *machine learning* numa tarefa muito mais acessível e produtiva [41].

No projeto descrito, o *TensorFlow* foi a principal biblioteca utilizada no desenvolvimento, treino e avaliação dos modelos de redes neurais. Tanto as *Convolutional Neural Networks* (CNN) como as *Liquid Neural Networks* (LNN) foram implementadas com base nesta tecnologia, tirando partido da sua integração com outras ferramentas do ecossistema *Python*, como o *NumPy* e a biblioteca *Neural Circuit Policies* (NCP), dedicada à implementação de *liquid layers*. A sua capacidade de processar grandes volumes de dados e realizar treinos eficientes foi determinante para a experimentação com múltiplas arquiteturas e estratégias de otimização.

3.1.5. GitHub

O *GitHub* é uma plataforma de hospedagem de código-fonte baseada no sistema de controlo de versões *Git*, que permite a colaboração, partilha e gestão de projetos de *software* de forma eficiente e segura [42]. Amplamente adotado por comunidades académicas e profissionais, o *GitHub* facilita o trabalho em equipa, mesmo em projetos distribuídos, e promove boas práticas de desenvolvimento, como o versionamento de código, a revisão colaborativa e a manutenção de documentação contínua.

Entre as suas funcionalidades destacam-se os repositórios remotos, os *commits* e *branches*, os *issues* para rastreamento de tarefas, e os *pull requests*, que permitem integrar alterações de forma controlada. Através desta ferramenta, foi possível manter uma cópia de segurança do código-base, sincronizar alterações entre dispositivos, acompanhar a evolução do projeto e facilitar a análise e discussão de melhorias de forma colaborativa.

3.1.6. Excel

O *Microsoft Excel* é uma ferramenta amplamente utilizada para análise de dados, organização de informação e criação de gráficos, devido à sua interface intuitiva e à capacidade de lidar com grandes volumes de dados de forma tabular [43]. Para além das

funções básicas de cálculo, o *Excel* disponibiliza funcionalidades avançadas como tabelas dinâmicas, fórmulas complexas, filtros, segmentação de dados e integração com outras aplicações.

No âmbito deste projeto, o *Excel* foi uma ferramenta fundamental no apoio à análise dos resultados experimentais. Foi utilizado para organizar métricas de desempenho dos modelos de redes neurais, criar gráficos comparativos e identificar tendências nos dados. Esta visualização permitiu avaliar a eficácia das estratégias aplicadas e tomar decisões informadas sobre as configurações mais promissoras para os modelos.

3.2. Dataset utilizado

A construção de um *dataset* adequado foi uma etapa essencial neste projeto, dada a especificidade da informação necessária para treinar e avaliar os modelos de redes neurais aplicados à previsão de chuva em Portugal Continental. Ao contrário de outros contextos em que é possível recorrer a *datasets* públicos já consolidados, foi necessário criar um conjunto de dados próprio, adaptado às particularidades das fontes de informação meteorológica nacionais.

Inicialmente, utilizou-se como base o *dataset* previamente recolhido no âmbito do projeto do ano letivo anterior, tendo sido adicionados novos dados de forma incremental. No entanto, rapidamente se identificaram inconsistências e erros na estrutura e conteúdo dos dados anteriores, os quais, aliados à necessidade de redefinir as classes de precipitação utilizadas nos modelos, levaram à decisão de reconstruir o *dataset* de raiz.

Optou-se então por implementar uma estratégia mais abrangente de recolha de dados meteorológicos, que consistiu no armazenamento completo dos recursos disponibilizados pelo IPMA (Figura 21). Esta recolha incluiu não só imagens de radar, mas também imagens de satélite, valores de precipitação, temperatura, humidade, velocidade e direção do vento, entre outros. Esta abordagem não só permitiu a criação de um *dataset* mais robusto e consistente para o presente projeto, como também garantiu a possibilidade de reutilização e reinterpretação dos dados em projetos futuros com diferentes objetivos ou metodologias.

```
def get_data_and_images_from_ipma():
    # Get json data from IPMA
    json_data = get_data(IPMA_API_URL).json()

    # Initialize list of unique timestamps
    unique_timestamps = set()

    # Fetch important data of each station
    for feature in json_data['features']:
        if feature['properties']['idEstacao'] in list(STATIONS.keys()):
            station_data = feature['properties']
            save_station_json_file(station_data)
            # Store unique timestamp
            timestamp = datetime.fromisoformat(station_data['time'])
            unique_timestamps.add(timestamp)

    print('Json files successfully updated!')

    save_radar_and_satellite_images(unique_timestamps)

    print('Images saved successfully!')
```

Figura 21 - Código para obter dados do IPMA

Com base nesse repositório alargado de dados brutos, foi então construído o *dataset* final necessário ao treino dos modelos, contendo as imagens de radar e respetivos valores de precipitação registados.

As imagens de radar utilizadas foram obtidas através dos serviços públicos do IPMA, que disponibiliza imagens com uma frequência de cinco em cinco minutos, com acesso até um máximo de aproximadamente um mês anterior à data da consulta. Já os valores de precipitação de cada estação meteorológica, disponibilizados também pelo IPMA, apresentam uma limitação mais significativa: apenas são acessíveis as medições correspondentes às últimas três horas (ou seja, se for feito um pedido à API às 15:00h, são obtidos os dados das 12:00h, 13:00h e 14:00h).

Para contornar esta restrição e assegurar uma recolha contínua de dados ao longo do tempo, foi desenvolvido um *script* automatizado que realiza pedidos periódicos à API do IPMA a cada três horas (Figura 22). Este processo de captura recorrente garantiu a construção de um repositório de dados meteorológicos atualizado e em constante crescimento, permitindo uma maior cobertura temporal.

```

@echo off

call %0\..\venv\Scripts\activate

:loop

rem Run python script
python data_fetch.py

rem Wait for 10800 seconds (3 hours)
timeout /t 10800

rem Loop back to start
goto loop

```

Figura 22 - Script para automatizar a recolha de dados

3.2.1. Pré-processamento das imagens

Antes da utilização das imagens de radar nos modelos de previsão, foi necessário realizar um conjunto de operações de pré-processamento com o objetivo de eliminar ruído visual e extrair a informação relevante para cada região do país. Estas transformações foram fundamentais para garantir que os dados de entrada representassem de forma mais fiel e focada as condições meteorológicas a nível distrital.

Num primeiro passo, foi aplicada uma função desenvolvida pelos nossos colegas no projeto do ano letivo anterior, responsável pela remoção da borda preta presente nas imagens originais (Figura 23). Esta borda, que rodeia o conteúdo central da imagem, não contém informação útil e poderia introduzir ruído nos modelos de *machine learning*, sendo por isso descartada.

```

def remove_black_pixels(image):
    # Convert the image to RGBA mode (if it's not already in RGBA mode)
    image = image.convert('RGBA')
    # Get the pixel data as a list of tuples
    pixels = list(image.getdata())
    # Replace every black pixel with transparent
    new_pixels = []
    for pixel in pixels:
        if pixel[0] == 0 and pixel[1] == 0 and pixel[2] == 0:
            new_pixels.append((0, 0, 0, 0))
        else:
            new_pixels.append(pixel)
    # Create a new image with the same size and mode as the original image
    new_image = Image.new(image.mode, image.size)
    # Update the new image with the new pixel data
    new_image.putdata(new_pixels)
    # Return the new image
    return new_image

```

Figura 23 - Função para remover os píxeis pretos da imagem

De seguida, tendo em consideração que a imagem de radar representa a totalidade do território de Portugal Continental, procedeu-se ao recorte da imagem em secções correspondentes a cada distrito. Para tal, recorreu-se a uma caixa de coordenadas previamente definida pelos nossos colegas, que permitiu extrair subimagens com uma dimensão de 200x200 pixels centradas em cada distrito. Este recorte visou isolar a área de influência de cada estação meteorológica, de modo a estabelecer uma relação direta entre a imagem e os respetivos valores de precipitação.

Este processo de segmentação foi essencial para a construção do *dataset* final, permitindo alinhar espacialmente os dados visuais (imagens de radar) com os dados numéricos (valores de precipitação), o que constitui a base das tarefas de previsão exploradas neste trabalho.

3.2.2. Normalização dos dados

A normalização dos valores de precipitação foi também uma etapa essencial na preparação do *dataset*, uma vez que a forma como os dados são categorizados influencia diretamente o desempenho e a fiabilidade dos modelos de classificação.

No projeto do ano letivo anterior, a normalização foi realizada com base nos critérios de emissão de avisos meteorológicos do IPMA (Tabela 1) [44], os quais definem limiares de precipitação associados a alertas de cor (Verde, Amarelo, Laranja e Vermelho). Embora esta abordagem esteja alinhada com práticas oficiais de comunicação de risco, revelou-se pouco eficaz do ponto de vista da aprendizagem automática, devido ao desequilíbrio significativo entre classes. Na prática, são raras as situações de precipitação extrema que justificam avisos de nível Laranja ou Vermelho, o que resultou num número muito reduzido de amostras nessas categorias, comprometendo a generalização do modelo.

Tabela 1 - Critérios de emissão de avisos meteorológicos do IPMA

Aviso	Valor de precipitação (mm/h)
Verde	< 10
Amarelo	10 a 20
Laranja	21 a 40
Vermelho	> 40

Face a este desafio, decidiu-se reformular a estratégia de normalização, adotando os critérios de classificação de chuva definidos também pelo IPMA, mas com base na intensidade da precipitação (Tabela 2) [45]. Esta abordagem permite uma segmentação mais equilibrada e informativa, refletindo melhor a distribuição real dos dados recolhidos.

Tabela 2 - Critérios de classificação de precipitação do IPMA

Classificação	Valor de precipitação (mm/h)
Sem Chuva	< 0.1
Chuva Fraca	0.1 a 0.49
Chuva Moderada	0.5 a 4
Chuva Forte	> 4

Com esta nova classificação, foi possível obter um número de amostras significativamente mais equilibrado entre as diferentes categorias de precipitação (Tabela 3).

Tabela 3 - Tabela de amostras obtidas

Classe	Número de amostras
Sem Chuva	14771
Chuva Fraca	1327
Chuva Moderada	1428
Chuva Forte	149

3.2.1. *Oversampling e Undersampling*

Conforme apresentado na tabela anterior, a distribuição das amostras por classe revelou-se significativamente desequilibrada, com um número muito superior de exemplos na categoria de precipitação nula, em contraste com a escassez de amostras nos restantes casos, principalmente nas situações de precipitação forte. Este tipo de desproporção compromete a capacidade dos modelos em generalizar para eventos menos frequentes, favorecendo previsões tendenciosas para as classes mais representadas.

Para mitigar este problema, recorreu-se a técnicas de *undersampling*, com o objetivo de equilibrar o número de amostras por classe, reduzindo a predominância das mais frequentes. A estratégia seguida consistiu na criação de um *counter dictionary* que atuava como contador

de ocorrências por classe (Figura 24), descartando novas amostras assim que fosse atingido o limite máximo previamente definido pelo utilizador.

```
# Increment the counter for the current precipitation value
counter_dictionary[precipitation_value] += 1
```

Figura 24 - Código com o incremento de um *counter dictionary*

Adicionalmente, foi também explorada uma abordagem preliminar de *oversampling*, embora de forma controlada e ainda não testada exaustivamente. Esta técnica consistia na criação de amostras substitutas nos casos em que não existissem dados suficientes para determinada classe, recorrendo à cópia de imagens temporais adjacentes (funções *copy previous* e *copy next*) como forma de simular novos exemplos (Figura 25). Esta técnica poderá ser aprofundada em trabalhos futuros, com o objetivo de reforçar ainda mais as classes minoritárias sem recorrer exclusivamente à exclusão de dados.

```
def generate_replacement_image(folder_path, image_datetime, method):
    match method:
        case 'copy_previous':
            new_image_datetime = image_datetime - timedelta(hours=1)
            new_image_date_string = image_datetime.strftime("%Y-%m-%d")
            new_image_datetime_string = new_image_datetime.strftime("%Y-%m-%dT%H%M")
            new_image_path = folder_path + '/' + new_image_date_string + '/' + new_image_datetime_string + '.png'
            # If the image exist, open it
            if os.path.exists(new_image_path):
                return Image.open(new_image_path)
            # If the image doesn't exist, return None
            else:
                return None
        case 'copy_next':
            new_image_datetime = image_datetime + timedelta(hours=1)
            new_image_date_string = image_datetime.strftime("%Y-%m-%d")
            new_image_datetime_string = new_image_datetime.strftime("%Y-%m-%dT%H%M")
            new_image_path = folder_path + '/' + new_image_date_string + '/' + new_image_datetime_string + '.png'
            # If the image exist, open it
            if os.path.exists(new_image_path):
                return Image.open(new_image_path)
            # If the image doesn't exist, return None
            else:
                return None
        case _:
            return None
```

Figura 25 - Função para gerar imagem substituta

3.2.2. Data Augmentation

Para complementar as estratégias de balanceamento de classes descritas anteriormente, foi aplicada a técnica de *data augmentation*, com o objetivo de aumentar a diversidade das amostras de treino e melhorar a capacidade de generalização dos modelos.

Neste projeto, optou-se por realizar *data augmentation* em tempo real, recorrendo aos recursos de pré-processamento de imagem fornecidos pela biblioteca *TensorFlow*,

concretamente através da classe *ImageDataGenerator* (Figura 26). Esta ferramenta permite aplicar transformações aleatórias às imagens durante o processo de treino, como rotações, zoom, deslocamentos ou alterações de escala, sem necessidade de duplicar os dados fisicamente no disco, o que reduz significativamente o tempo de processamento.

```
# Data augmentation
datagen = keras.preprocessing.image.ImageDataGenerator(
    rotation_range=5,
    width_shift_range=0.02,
    height_shift_range=0.02,
    shear_range=0.02,
    zoom_range=0,
    fill_mode='nearest'
)
```

Figura 26 - Configuração do *Data Augmentation*

Neste caso foram definidas as seguintes transformações:

- Rotação das imagens num valor aleatório entre 0 e 5 graus, para a esquerda ou para a direita;
- Deslocação horizontal entre 0% e 2% da largura total;
- Deslocação vertical entre 0% e 2% da altura total;
- Cisalhamento entre 0% e 2%;
- Preenchimento automático dos píxeis vazios resultantes das transformações, com o valor do píxel mais próximo.

Durante o treino do modelo, é chamado o método *flow* da classe *ImageDataGenerator* (Figura 27), que recebe como parâmetros as imagens de treino (*train_images*), as respetivas classes em *one-hot encoding* (*train_labels_one_hot*) e o tamanho dos lotes a ser gerado (*batch_size*). O método devolve um gerador que, quando ativado, produz lotes de imagens com as transformações aleatórias definidas.

```
# Train model
history = cnn_model.fit(datagen.flow(train_images, train_labels_one_hot, batch_size=BATCH_SIZE), epochs=EPOCHS,
validation_data=(val_images, val_labels_one_hot), callbacks=[model_checkpoint, time_callback])
```

Figura 27 - Treino do modelo com *Data Augmentation* em tempo real

Cada lote produzido é único, assegurando que o modelo é exposto a variações realistas em cada época, o que reforça a sua capacidade de generalização. Assim, mesmo com um conjunto de dados limitado, é possível simular milhares de exemplos distintos durante o treino do modelo.

3.3. Construção das redes neurais

Após a finalização da preparação do *dataset* procedeu-se ao desenvolvimento e experimentação de várias arquiteturas de redes neurais com o objetivo de encontrar a que melhor se adaptasse ao objetivo em questão.

A construção dos modelos seguiu uma abordagem incremental: começou-se por uma *Convolutional Neural Network* tradicional (CNN), passando posteriormente para *Liquid Neural Networks* puras (LNN baseada em LTC e CfC) e, por fim, integraram-se estas unidades em redes híbridas (CNN + LNN), com e sem camadas *fully-connected*.

3.3.1. Arquitetura CNN original

A primeira arquitetura implementada neste projeto corresponde a uma CNN desenvolvida pelos colegas do projeto anterior. Esta rede serviu como ponto de partida para a avaliação do desempenho dos modelos subsequentes e como linha de base para a comparação de resultados.

A estrutura da rede é composta por um total de dezoito camadas, organizadas em duas fases principais: extração de *features* e classificação. A primeira fase inicia-se com a camada *Input*, responsável por definir a estrutura dos dados de entrada (imagens de 100x100 píxeis no formato Red-Green-Blue-Alpha).

De seguida temos uma camada *Conv2D* com 32 filtros de tamanho 3x3, função de ativação ReLU, *strides* de tamanho 1 e *padding* com valor *same* (ou seja, são mantidas as dimensões da imagem). Após esta camada, temos uma *BatchNormalization*, responsável por normalizar o *input*, ajustando o mesmo a uma certa escala. Esta combinação das camadas *Conv2D* e *BatchNormalization* é novamente repetida, procurando assim detetar os padrões mais significativos das imagens.

Seguidamente aplicou-se uma camada *MaxPooling2D*, com *pool size* de 2x2 e *strides* de tamanho 2, responsável por reduzir a dimensionalidade do *input*, sendo que, neste caso a camada analisa partes de 2x2 píxeis da imagem e apenas devolve como *output* o valor mais elevado dos quatro píxeis. Esta camada é bastante eficaz para extrair as características mais salientes, e o *pool size* pequeno é indicado para extrair o máximo dessas mesmas características.

Após esta camada, foi aplicado um *Dropout* de valor 0.35, responsável por desativar aleatoriamente 35% dos neurónios, esperando assim evitar o *overfitting*, e de seguida voltou-se a repetir o conjunto de camadas desde a primeira *Conv2D* até ao *Dropout*, tendo sido apenas alterado o número de filtros das *Conv2D* para 64 e o valor do *Dropout* para 0.0035, procurando assim identificar as verdadeiras características mais relevantes.

Terminando a fase de extração de *features* aplicou-se uma camada *Flatten*, que transforma os dados num *array* unidimensional, dando assim início à fase de classificação. Esta fase é composta por um conjunto de camadas *fully-connected*, sendo neste caso utilizadas duas camadas *Dense* com 64 neurónios cada, com função de ativação ReLU e com inicializador de pesos personalizável, sendo utilizado por definição o *Glorot Uniform*. Seguiu-se então uma nova camada de *Batch Normalization* antes de finalizar com uma última camada *Dense* com função de ativação *Softmax* e número de neurónios igual ao número de classes possíveis.

Ao longo do processo, a ReLU foi utilizada como função de ativação principal por promover não-linearidade e *sparsity*, características desejáveis para lidar com dados complexos. A função *Softmax*, utilizada na última camada, foi escolhida por ser a mais apropriada para problemas de classificação multi-classe.

De seguida, apresenta-se o código do modelo (Figura 28).

```
cnn_model = keras.Sequential([
    keras.layers.Input(shape=(IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS)),
    keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', strides=1, padding='same',
                       data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', strides=1, padding='same',
                       data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding='valid'),
    keras.layers.Dropout(0.35),
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu', strides=1, padding='same',
                       data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), strides=1, padding='same', activation='relu',
                       data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding='valid' ),
    keras.layers.Dropout(0.0035),
    keras.layers.Flatten(name='flatten')
])
cnn_model.add(keras.layers.Dense(HIDDEN_NEURONS, activation='relu', kernel_initializer=initializer1))
# Add hidden layers
for hidden_layer in range(HIDDEN_LAYERS):
    cnn_model.add(keras.layers.Dense(HIDDEN_NEURONS, activation='relu', kernel_initializer=initializer2))
cnn_model.add(keras.layers.BatchNormalization())
cnn_model.add(keras.layers.Dense(len(PRECIPITATION_CLASSESS), activation='softmax', kernel_initializer=initializer3))
```

Figura 28 - Código da arquitetura da CNN

3.3.2. Arquitetura LNN baseada em LTC

Dentro da categoria das LNN puras, a primeira a ser explorada foi uma arquitetura baseada na abordagem *Liquid Time-Constant* (LTC), uma classe de redes neuronais recorrentes contínuas no tempo, que permite capturar dinâmicas temporais complexas através de um mecanismo não-linear de atualização dos estados internos dos neurónios (ODE – *Ordinary Differential Equations*). Este tipo de rede é particularmente eficaz na modelação de sinais que apresentam dependências temporais, como é o caso dos fenómenos meteorológicos.

Foram desenvolvidas duas variantes da arquitetura LTC, dependendo da estrutura dos dados de entrada utilizada: uma versão para dados estáticos (imagens isoladas) e outra para séries de dados temporais (sequências de imagens).

Variante 1 – Dados estáticos

A primeira variante é composta por quatro camadas e assume como *input* uma única imagem de radar com quatro canais (100x100x4). Esta imagem é inicialmente reformatada com uma camada *Reshape* para um *array* tridimensional com a dimensão (1, 100x100x4), de forma a simular uma sequência temporal de um único passo. Este *array* é então processado pela camada LTC, utilizando a implementação *AutoNCP* (ligação entre neurónios automática), ao invés da tradicional *fully-connected* (ligação completa entre neurónios). Esta implementação permite extrair características de alto nível de forma dinâmica, diminuindo o esforço computacional. Por fim, os dados passam por uma camada *Dense* com ativação *Softmax*, que produz uma distribuição de probabilidades sobre as classes de precipitação.

Apresenta-se agora o código do modelo (Figura 29).

```
ltc_model = keras.Sequential([
    keras.layers.Input(shape=(IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS)),
    keras.layers.Reshape((1, IMAGE_HEIGHT * IMAGE_WIDTH * IMAGE_CHANNELS)),
    LTC(AutoNCP(HIDDEN_NEURONS, output_size=len(PRECIPITATION_CLASSES))),
    keras.layers.Dense(len(PRECIPITATION_CLASSES), activation='softmax')
])
```

Figura 29 - Código da arquitetura da LNN baseada em LTC com dados estáticos

Variante 2 – Dados temporais

Na segunda variante, a estrutura do modelo mantém-se exatamente a mesma, sendo a única alteração o formato do *input*, que passa a ser uma sequência de três imagens consecutivas com 1 hora de diferença entre cada. Estas imagens são igualmente reformatadas para uma sequência de *arrays*, com dimensão (3, 100x100x4), permitindo à camada LTC processar a informação de forma recorrente ao longo do tempo.

A principal vantagem desta abordagem reside na capacidade da camada LTC de adaptar dinamicamente a constante temporal de cada neurónio, consoante o conteúdo da sequência de entrada. Isto permite uma representação mais rica e eficiente de fenómenos temporais com variações não lineares, como é o caso das mudanças de precipitação ao longo do tempo.

De seguida encontra-se o código do modelo (Figura 30).

```
ltc_model = keras.Sequential([
    keras.layers.Input(shape=(SEQUENCE_LENGTH, IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS)),
    keras.layers.Reshape((SEQUENCE_LENGTH, IMAGE_HEIGHT * IMAGE_WIDTH * IMAGE_CHANNELS)),
    LTC(AutoNCP(HIDDEN_NEURONS, output_size=len(PRECIPITATION_CLASSES))),
    keras.layers.Dense(len(PRECIPITATION_CLASSES), activation='softmax')
])
```

Figura 30 - Código da arquitetura da LNN baseada em LTC com dados temporais

3.3.3. Arquitetura LNN baseada em CfC

A segunda das LNN puras desenvolvida foi uma baseada na abordagem *Closed-Form Continuous-Time* (CfC). Esta representa uma simplificação conceptual em relação ao modelo LTC, ao utilizar equações *closed-form* para modelar sistemas dinâmicos contínuos, em vez das ODE. O modelo CfC consegue manter a expressividade das redes recorrentes contínuas, ao mesmo tempo que reduz significativamente a complexidade computacional, tornando-se especialmente atrativo para aplicações em tempo real.

Tal como no modelo anterior, foram desenvolvidas duas variantes da arquitetura: uma para dados estáticos e outra para dados temporais. A única diferença entre as arquiteturas baseadas em LTC e CfC reside na substituição da camada LTC pela camada CfC, mantendo-se inalteradas todas as restantes componentes da rede, incluindo a utilização da implementação *AutoNCP*.

O código do modelo para dados estáticos encontra-se disponível na Figura 31, enquanto que para dados temporais está disponível na Figura 32.

```
cfc_model = keras.Sequential([
    keras.layers.Input(shape=(IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS)),
    keras.layers.Reshape((1, IMAGE_HEIGHT * IMAGE_WIDTH * IMAGE_CHANNELS)),
    CfC(AutoNCP(HIDDEN_NEURONS, output_size=len(PRECIPITATION_CLASSES))),
    keras.layers.Dense(len(PRECIPITATION_CLASSES), activation='softmax')
])
```

Figura 31 - Código da arquitetura da LNN baseada em CfC com dados estáticos

```
cfc_model = keras.Sequential([
    keras.layers.Input(shape=(SEQUENCE_LENGTH, IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS)),
    keras.layers.Reshape((SEQUENCE_LENGTH, IMAGE_HEIGHT * IMAGE_WIDTH * IMAGE_CHANNELS)),
    CfC(AutoNCP(HIDDEN_NEURONS, output_size=len(PRECIPITATION_CLASSES))),
    keras.layers.Dense(len(PRECIPITATION_CLASSES), activation='softmax')
])
```

Figura 32 - Código da arquitetura da LNN baseada em CfC com dados temporais

3.3.4. Arquitetura híbrida: CNN com LTC (sem camada *fully-connected*)

Com o objetivo de tirar partido das vantagens oferecidas tanto pelas CNN como pelas LNN, foi desenvolvida uma arquitetura híbrida que combina a fase de extração de *features* da CNN com uma camada recorrente LTC. Esta integração visa aliar a capacidade das redes convolucionais em extrair características espaciais relevantes à competência das redes líquidas em modelar dependências temporais complexas.

Foram novamente consideradas duas variantes distintas, consoante o tipo de dados fornecidos à rede: dados estáticos e dados temporais.

Variante 1 – Dados estáticos

Nesta variante, é utilizada a arquitetura convolucional descrita na secção 3.3.1, mas apenas até à camada *Flatten*. A partir desse ponto, a camada LTC é responsável por processar a representação vetorial extraída pela CNN, substituindo as camadas *fully-connected* tradicionais.

O código representativo deste modelo está exposto a seguir (Figura 33).

```

lnn_model = keras.Sequential([
    keras.layers.Input(shape=(IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS)),
    keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', strides=1, padding='same',
                       data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', strides=1, padding='same',
                       data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding='valid'),
    keras.layers.Dropout(0.35),
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu', strides=1, padding='same',
                       data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), strides=1, padding='same', activation='relu',
                       data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding='valid' ),
    keras.layers.Dropout(0.0035),
    keras.layers.Flatten(name='flatten'),
    keras.layers.Reshape((1, IMAGE_HEIGHT * IMAGE_WIDTH * IMAGE_CHANNELS), name='reshape')
])
lnn_model.add(LTC(AutoNCP(HIDDEN_NEURONS, output_size=len(PRECIPITATION_CLASSES))))
lnn_model.add(keras.layers.Dense(len(PRECIPITATION_CLASSES), activation='softmax'))

```

Figura 33 - Código da arquitetura híbrida CNN + LTC com dados estáticos (sem camada *fully-connected*)

Variante 2 – Dados temporais

Para esta variante, considerou-se novamente uma sequência de três imagens sucessivas como *input*. Todas as camadas convolucionais até *Flatten* foram encapsuladas com a classe *TimeDistributed*, permitindo aplicar os mesmos filtros convolucionais a cada passo temporal da sequência. A saída temporal da CNN foi então processada pela camada LTC, à semelhança da abordagem anterior.

Esta abordagem híbrida revelou-se promissora, ao combinar as vantagens das CNN na extração de padrões espaciais com a dinâmica adaptativa das LNN no domínio temporal.

A figura com o código da arquitetura encontra-se de seguida (Figura 34).

```

lnn_model = keras.Sequential([
    keras.layers.Input(shape=(SEQUENCE_LENGTH, IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS)),
    keras.layers.TimeDistributed(keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', strides=1,
                                                    padding='same', data_format='channels_last')),
    keras.layers.TimeDistributed(keras.layers.BatchNormalization()),
    keras.layers.TimeDistributed(keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', strides=1,
                                                    padding='same', data_format='channels_last')),
    keras.layers.TimeDistributed(keras.layers.BatchNormalization()),
    keras.layers.TimeDistributed(keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding='valid')),
    keras.layers.TimeDistributed(keras.layers.Dropout(0.35)),
    keras.layers.TimeDistributed(keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu', strides=1,
                                                    padding='same', data_format='channels_last')),
    keras.layers.TimeDistributed(keras.layers.BatchNormalization()),
    keras.layers.TimeDistributed(keras.layers.Conv2D(filters=64, kernel_size=(3, 3), strides=1, padding='same',
                                                    activation='relu', data_format='channels_last')),
    keras.layers.TimeDistributed(keras.layers.BatchNormalization()),
    keras.layers.TimeDistributed(keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding='valid' )),
    keras.layers.TimeDistributed(keras.layers.Dropout(0.0035)),
    keras.layers.TimeDistributed(keras.layers.Flatten(name='flatten')),
    keras.layers.Reshape((SEQUENCE_LENGTH, IMAGE_HEIGHT * IMAGE_WIDTH * IMAGE_CHANNELS), name='reshape')
])
lnn_model.add(LTC(AutoNCP(HIDDEN_NEURONS, output_size=len(PRECIPITATION_CLASSES))))
lnn_model.add(keras.layers.Dense(len(PRECIPITATION_CLASSES), activation='softmax'))

```

Figura 34 - Código da arquitetura híbrida CNN + LTC com dados temporais (sem camada *fully-connected*)

3.3.5. Arquitetura híbrida: CNN com CfC (sem camada *fully-connected*)

À semelhança do que já se tinha feito na secção 3.3.3 e seguindo a mesma linha de raciocínio da secção anterior, foi implementada uma variante da arquitetura híbrida, agora substituindo a camada LTC pela camada CfC. Esta substituição teve como objetivo avaliar o impacto da mudança de paradigma dentro das LNN no desempenho final do modelo híbrido.

As figuras com o código para dados estáticos (Figura 35) e para dados temporais (Figura 36) encontram-se a seguir.

```
lnn_model = keras.Sequential([
    keras.layers.Input(shape=(IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS)),
    keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', strides=1, padding='same',
                       data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', strides=1, padding='same',
                       data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding='valid'),
    keras.layers.Dropout(0.35),
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu', strides=1, padding='same',
                       data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), strides=1, padding='same', activation='relu',
                       data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding='valid' ),
    keras.layers.Dropout(0.0035),
    keras.layers.Flatten(name='flatten'),
    keras.layers.Reshape((1, IMAGE_HEIGHT * IMAGE_WIDTH * IMAGE_CHANNELS), name='reshape')
])
lnn_model.add(CfC(AutoNCP(HIDDEN_NEURONS, output_size=len(PRECIPITATION_CLASSES))))
lnn_model.add(keras.layers.Dense(len(PRECIPITATION_CLASSES), activation='softmax'))
```

Figura 36 - Código da arquitetura híbrida CNN + CfC com dados estáticos (sem camada *fully-connected*)

```
lnn_model = keras.Sequential([
    keras.layers.Input(shape=(SEQUENCE_LENGTH, IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS)),
    keras.layers.TimeDistributed(keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', strides=1,
                                                    padding='same', data_format='channels_last')),
    keras.layers.TimeDistributed(keras.layers.BatchNormalization()),
    keras.layers.TimeDistributed(keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', strides=1,
                                                    padding='same', data_format='channels_last')),
    keras.layers.TimeDistributed(keras.layers.BatchNormalization()),
    keras.layers.TimeDistributed(keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding='valid')),
    keras.layers.TimeDistributed(keras.layers.Dropout(0.35)),
    keras.layers.TimeDistributed(keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu', strides=1,
                                                    padding='same', data_format='channels_last')),
    keras.layers.TimeDistributed(keras.layers.BatchNormalization()),
    keras.layers.TimeDistributed(keras.layers.Conv2D(filters=64, kernel_size=(3, 3), strides=1, padding='same',
                                                    activation='relu', data_format='channels_last')),
    keras.layers.TimeDistributed(keras.layers.BatchNormalization()),
    keras.layers.TimeDistributed(keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding='valid' )),
    keras.layers.TimeDistributed(keras.layers.Dropout(0.0035)),
    keras.layers.TimeDistributed(keras.layers.Flatten(name='flatten')),
    keras.layers.Reshape((SEQUENCE_LENGTH, IMAGE_HEIGHT * IMAGE_WIDTH * IMAGE_CHANNELS), name='reshape')
])
lnn_model.add(CfC(AutoNCP(HIDDEN_NEURONS, output_size=len(PRECIPITATION_CLASSES))))
lnn_model.add(keras.layers.Dense(len(PRECIPITATION_CLASSES), activation='softmax'))
```

Figura 35 - Código da arquitetura híbrida CNN + CfC com dados temporais (sem camada *fully-connected*)

3.3.6. Arquitetura híbrida: CNN com LTC (com camada *fully-connected*)

Nesta secção, é apresentada uma extensão da arquitetura híbrida CNN + LTC, com a introdução de camadas *fully-connected* antes da camada LTC. O objetivo principal desta modificação foi permitir o teste de diferentes métodos de inicialização de pesos nessas camadas *Dense*, de forma a analisar o seu impacto no desempenho final do modelo.

Ao contrário das versões anteriores, nesta abordagem foi apenas desenvolvida uma variante para dados estáticos. A estrutura base do modelo é idêntica à da CNN original, sendo apenas introduzida a camada LTC antes da última camada *Dense* (responsável pela classificação final).

Abaixo apresenta-se o código da arquitetura (Figura 37).

```
lnn_model = keras.Sequential([
    keras.layers.Input(shape=(IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS)),
    keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', strides=1, padding='same',
    data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', strides=1, padding='same',
    data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding='valid'),
    keras.layers.Dropout(0.35),
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu', strides=1, padding='same',
    data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), strides=1, padding='same', activation='relu',
    data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding='valid' ),
    keras.layers.Dropout(0.0035),
    keras.layers.Flatten(name='flatten')
])
lnn_model.add(keras.layers.Dense(HIDDEN_NEURONS, activation='relu', kernel_initializer=initializer1))
# Add hidden layers
for hidden_layer in range(HIDDEN_LAYERS):
    lnn_model.add(keras.layers.Dense(HIDDEN_NEURONS, activation='relu', kernel_initializer=initializer2))
lnn_model.add(keras.layers.BatchNormalization())
lnn_model.add(keras.layers.Reshape((1, HIDDEN_NEURONS), name='reshape'))
lnn_model.add(LTC(AutoNCP(HIDDEN_NEURONS, output_size=len(PRECIPITATION_CLASSES))))
lnn_model.add(keras.layers.Dense(len(PRECIPITATION_CLASSES), activation='softmax', kernel_initializer=initializer3))
```

Figura 37 - Código da arquitetura híbrida CNN + LTC com dados estáticos (com camada *fully-connected*)

3.3.7. Arquitetura híbrida: CNN com CfC (com camada *fully-connected*)

Esta secção apresenta a última arquitetura testada neste projeto, baseada na integração de uma CNN completa com uma camada CfC, seguindo a mesma lógica da versão anterior com a camada LTC.

A única modificação efetuada em relação ao modelo anterior consistiu, mais uma vez, na substituição da camada LTC pela camada CfC. A estrutura do modelo mantém-se rigorosamente igual: uma CNN com camadas *fully-connected* adicionadas antes da LNN, com o objetivo de testar diferentes inicializações de pesos e avaliar o impacto da camada recorrente contínua na fase final de classificação.

A seguir apresenta-se o código do modelo (Figura 38).

```

lnn_model = keras.Sequential([
    keras.layers.Input(shape=(IMAGE_HEIGHT, IMAGE_WIDTH, IMAGE_CHANNELS)),
    keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', strides=1, padding='same',
    data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', strides=1, padding='same',
    data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding='valid'),
    keras.layers.Dropout(0.35),
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu', strides=1, padding='same',
    data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), strides=1, padding='same', activation='relu',
    data_format='channels_last'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding='valid' ),
    keras.layers.Dropout(0.0035),
    keras.layers.Flatten(name='flatten')
])
lnn_model.add(keras.layers.Dense(HIDDEN_NEURONS, activation='relu', kernel_initializer=initializer1))
# Add hidden layers
for hidden_layer in range(HIDDEN_LAYERS):
    lnna_model.add(keras.layers.Dense(HIDDEN_NEURONS, activation='relu', kernel_initializer=initializer2))
lnn_model.add(keras.layers.BatchNormalization())
lnn_model.add(keras.layers.Reshape((1, HIDDEN_NEURONS), name='reshape'))
lnn_model.add(CfC(AutoNCP(HIDDEN_NEURONS, output_size=len(PRECIPITATION_CLASSES))))
lnn_model.add(keras.layers.Dense(len(PRECIPITATION_CLASSES), activation='softmax', kernel_initializer=initializer3))

```

Figura 38 - Código da arquitetura híbrida CNN + CfC com dados estáticos (com camada *fully-connected*)

3.4. Treino e avaliação dos modelos

Este capítulo descreve o processo de treino e avaliação dos diferentes modelos desenvolvidos ao longo do projeto, com o objetivo de analisar o seu desempenho na tarefa de previsão de precipitação. Foram implementadas várias abordagens, cada uma com pequenas alterações estruturais ou metodológicas, que permitiram explorar diferentes combinações de redes neurais.

Apesar das diferenças entre abordagens, houve um conjunto de parâmetros e metodologias que se mantiveram constantes ao longo de todas as experiências, garantindo a consistência e comparabilidade dos resultados. Destacam-se os seguintes pontos comuns:

Reprodutibilidade: foi definida uma *seed* fixa para assegurar que os resultados obtidos fossem replicáveis em diferentes execuções.

Horizonte de previsão: para todas as abordagens, considerou-se uma diferença temporal de 1 hora entre a imagem de radar e o valor de precipitação associado. Apenas na última abordagem foi também testada a previsão a 2 e 3 horas.

Número de amostras: foram utilizadas 140 amostras, de cada classe, para dados estáticos (imagens isoladas) e 125 amostras para séries temporais (sequências de imagens).

Data Augmentation: sempre que foi aplicada a técnica de *data augmentation*, esta tinha um *batch size* de valor 8.

Cross-Validation: recorreu-se à estratégia de *Stratified K-Fold Cross Validation* com 5 *folds*, assegurando uma distribuição equilibrada das classes em cada partição, correspondendo a 80% dos dados para treino e 20% para validação.

Otimização: o optimizador utilizado em todas as abordagens foi o *Adamax*, com uma taxa de aprendizagem de 0.001, por apresentar bons resultados em dados esparsos.

Loss function: a função *loss* utilizada foi a *categorical crossentropy*, apropriada para problemas de classificação com múltiplas classes.

Métricas de avaliação: os modelos foram avaliados segundo as métricas de *accuracy*, *precision*, *recall* e *f1-score*, sendo dada especial atenção à *accuracy*, como métrica principal de comparação entre abordagens.

Funções *callback*: foram desenvolvidas duas funções *callback* a serem utilizadas durante o treino do modelo. A primeira foi *time_callback*, responsável por registar o tempo médio por época durante o treino. A segunda foi *model_checkpoint*, responsável por guardar os melhores pesos do modelo com base na performance de validação, permitindo a posterior reutilização do modelo otimizado.

Nos tópicos seguintes serão detalhadas as diferentes abordagens testadas, com a apresentação dos respetivos resultados.

3.4.1. Primeira abordagem

A primeira abordagem teve como objetivo avaliar o desempenho do modelo base, a CNN original aplicada a dados estáticos, de forma a estabelecer uma linha de base para a comparação com os modelos mais avançados testados posteriormente.

Foram realizados quatro testes distintos, com variação no número de épocas de treino e na utilização ou não de *data augmentation*:

CNN com 50 epochs e *data augmentation*: Nesta primeira configuração, o modelo foi treinado durante 50 épocas utilizando a técnica de *data augmentation* em tempo real. O objetivo era verificar se a rede conseguia começar a aprender padrões relevantes com um número reduzido de épocas, mantendo a diversidade dos dados.

Os resultados obtidos neste teste podem ser consultados na Tabela 4.

Tabela 4 - Resultados do teste: CNN com 50 epochs e *data augmentation*

Métrica	Valor
<i>Accuracy</i>	0.992
<i>Validation Accuracy</i>	0.6964
<i>Loss</i>	0.0457
<i>Validation Loss</i>	1.1042
<i>Precision</i>	0.9942
<i>Validation Precision</i>	0.7147
<i>Recall</i>	0.9902
<i>Validation Recall</i>	0.6821
<i>Average Epoch Runtime</i>	14.7486

CNN com 100 epochs e data augmentation: Com base nos resultados promissores da configuração anterior, foi duplicado o número de épocas de treino para 100. Esta alteração permitiu analisar o impacto do treino mais prolongado sobre a capacidade de generalização do modelo, mantendo-se o *data augmentation* como ferramenta de suporte ao processo de aprendizagem.

Os resultados obtidos neste teste podem ser consultados na Tabela 5.

Tabela 5 - Resultados do teste: CNN com 100 epochs e data augmentation

Métrica	Valor
<i>Accuracy</i>	0.9991
<i>Validation Accuracy</i>	0.7321
<i>Loss</i>	0.0144
<i>Validation Loss</i>	0.9646
<i>Precision</i>	0.9991
<i>Validation Precision</i>	0.7539
<i>Recall</i>	0.9987
<i>Validation Recall</i>	0.7268
<i>Average Epoch Runtime</i>	14.5111

CNN com 200 epochs e data augmentation: Para avaliar se o aumento do tempo de treino levaria a uma melhoria contínua do desempenho, foi realizada uma terceira experiência com 200 épocas. O objetivo era perceber se o modelo ainda beneficiava de mais iterações ou se entrava numa fase de *overfitting*, mesmo com os mecanismos de regularização presentes na arquitetura (*dropout* e *batch normalization*).

Os resultados obtidos neste teste podem ser consultados na Tabela 6.

Tabela 6 - Resultados do teste: CNN com 200 epochs e dataaugmentation

Métrica	Valor
<i>Accuracy</i>	1.0
<i>Validation Accuracy</i>	0.7286
<i>Loss</i>	0.0058
<i>Validation Loss</i>	1.0272
<i>Precision</i>	1.0
<i>Validation Precision</i>	0.7635
<i>Recall</i>	1.0
<i>Validation Recall</i>	0.7232
<i>Average Epoch Runtime</i>	13.8198

CNN com 100 epochs sem data augmentation: Por fim, foi testada uma quarta configuração com 100 épocas, mas sem aplicar *data augmentation*. Esta experiência permitiu comprovar o benefício da diversificação de dados na fase de treino do modelo, tendo sido a partir daí aplicada a todos os modelos seguintes.

Os resultados obtidos neste teste podem ser consultados na Tabela 7.

Tabela 7 - Resultados do teste: CNN com 100 epochs sem data augmentation

Métrica	Valor
<i>Accuracy</i>	1.0
<i>Validation Accuracy</i>	0.6429
<i>Loss</i>	0.0005
<i>Validation Loss</i>	1.4967
<i>Precision</i>	1.0
<i>Validation Precision</i>	0.6595
<i>Recall</i>	1.0
<i>Validation Recall</i>	0.6321
<i>Average Epoch Runtime</i>	13.3992

3.4.2. Segunda abordagem

A segunda abordagem teve como objetivo avaliar o desempenho das LNN puras, tanto a arquitetura LTC como a CfC, quando aplicadas a dados estáticos. Nesta fase, ambos os modelos foram treinados durante 100 épocas, utilizando as mesmas configurações globais.

Apesar destas redes estarem desenhadas para lidar com dados temporais, foi importante compreender o seu comportamento em cenários mais simples, onde apenas uma imagem (representando um instante no tempo) era utilizada como *input*. Para simular uma sequência temporal de um único passo, cada imagem foi reformatada para um vetor tridimensional, o que permitiu compatibilizar o formato dos dados com os requisitos das camadas LNN.

Os resultados obtidos em ambos os testes podem ser consultados na Tabela 8 e 9.

Tabela 8 - Resultados do teste: LTC com dados estáticos e 100 *epochs*

Métrica	Valor
<i>Accuracy</i>	0.8125
<i>Validation Accuracy</i>	0.6679
<i>Loss</i>	0.5051
<i>Validation Loss</i>	0.8336
<i>Precision</i>	0.89
<i>Validation Precision</i>	0.8667
<i>Recall</i>	0.7741
<i>Validation Recall</i>	0.6107
<i>Average Epoch Runtime</i>	28.0608

Tabela 9 - Resultados do teste: CfC com dados estáticos e 100 epochs

Métrica	Valor
<i>Accuracy</i>	1.0
<i>Validation Accuracy</i>	0.7143
<i>Loss</i>	0.0064
<i>Validation Loss</i>	0.878
<i>Precision</i>	1.0
<i>Validation Precision</i>	0.731
<i>Recall</i>	1.0
<i>Validation Recall</i>	0.6982
<i>Average Epoch Runtime</i>	5.0685

3.4.3. Terceira abordagem

Na terceira abordagem, foram avaliadas as arquiteturas híbridas (sem camada *fully-connected*), nomeadamente a CNN + LTC e a CNN + CfC. Ambos os modelos foram aplicados a dados estáticos e treinados durante 100 épocas, mantendo-se os restantes parâmetros constantes conforme indicado na introdução deste capítulo.

Nesta versão híbrida, o bloco convolucional da CNN foi mantido até à camada *Flatten*, responsável por transformar o mapa de características tridimensional num vetor unidimensional. A partir daí, em vez de seguir com camadas *fully-connected* como na arquitetura original, foi integrada diretamente uma camada LTC ou CfC, responsável por interpretar o vetor de características como uma sequência temporal de um único passo (sem dinâmica temporal real).

O objetivo desta abordagem foi perceber se a substituição da fase de classificação tradicional por uma camada líquida, mesmo sem dados sequenciais, poderia introduzir melhorias no desempenho, nomeadamente pela sua natureza não-linear e adaptativa.

A comparação entre estas arquiteturas híbridas e os modelos puramente convolucionais ou puramente líquidos permitiu uma análise mais detalhada da utilidade de cada componente, isolando o impacto das camadas líquidas na generalização do modelo.

De seguida são apresentados os resultados relativos aos dois testes (Tabela 10) e (Tabela 11).

Tabela 10 - Resultados do teste: CNN + LTC (sem camada *fully-connected*) com dados estáticos e 100 epochs

Métrica	Valor
<i>Accuracy</i>	0.9719
<i>Validation Accuracy</i>	0.7464
<i>Loss</i>	0.1303
<i>Validation Loss</i>	0.6459
<i>Precision</i>	0.9766
<i>Validation Precision</i>	0.8844
<i>Recall</i>	0.9679
<i>Validation Recall</i>	0.7286
<i>Average Epoch Runtime</i>	38.8977

Tabela 11 - Resultados do teste: CNN + CfC (sem camada *fully-connected*) com dados estáticos e 100 epochs

Métrica	Valor
<i>Accuracy</i>	1.0
<i>Validation Accuracy</i>	0.7161
<i>Loss</i>	0.0015
<i>Validation Loss</i>	1.2754
<i>Precision</i>	1.0
<i>Validation Precision</i>	0.7186
<i>Recall</i>	1.0
<i>Validation Recall</i>	0.7161
<i>Average Epoch Runtime</i>	16.4166

3.4.4. Quarta abordagem

A quarta abordagem teve como principal objetivo estudar o impacto dos métodos de inicialização de pesos nas camadas *fully-connected* das arquiteturas híbridas, mais especificamente com os métodos *Glorot Uniform* (técnica por omissão já utilizada na primeira abordagem) e PSJ (um método alternativo desenvolvido com o objetivo de explorar uma inicialização mais personalizada), este último já validado no projeto do ano letivo anterior como o mais eficaz para a arquitetura CNN.

Embora o projeto anterior já tenha explorado extensivamente a inicialização de pesos para a CNN, optou-se por realizar novamente testes com o método PSJ na CNN como modelo de controlo, uma vez que, neste projeto, foram introduzidas alterações relevantes ao *dataset*, incluindo ajustes nas classes de precipitação e no seu equilíbrio.

Foram, assim, testadas as seguintes arquiteturas com dados estáticos e 100 épocas de treino:

- A CNN original com o método de inicialização PSJ (como controlo) (Tabela 12);

Tabela 12 - Resultados do teste: CNN com dados estáticos, 100 *epochs* e método PSJ

Métrica	Valor
<i>Accuracy</i>	1.0
<i>Validation Accuracy</i>	0.7125
<i>Loss</i>	0.0091
<i>Validation Loss</i>	0.9445
<i>Precision</i>	1.0
<i>Validation Precision</i>	0.7307
<i>Recall</i>	1.0
<i>Validation Recall</i>	0.7
<i>Average Epoch Runtime</i>	14.1143

- A arquitetura híbrida CNN + LTC com camada *fully-connected*, testada com os métodos *Glorot Uniform* (Tabela 13) e PSJ (Tabela 14);

Tabela 13 - Resultados do teste: CNN + LTC (com camada *fully-connected*) com dados estáticos, 100 *epochs* e *Glorot Uniform*

Métrica	Valor
<i>Accuracy</i>	0.9973
<i>Validation Accuracy</i>	0.7393
<i>Loss</i>	0.0225
<i>Validation Loss</i>	0.7331
<i>Precision</i>	0.9978
<i>Validation Precision</i>	0.9742
<i>Recall</i>	0.9969

<i>Validation Recall</i>	0.7321
<i>Average Epoch Runtime</i>	14.612

Tabela 14 - Resultados do teste: CNN + LTC (com camada *fully-connected*) com dados estáticos, 100 *epochs* e PSJ

Métrica	Valor
<i>Accuracy</i>	0.9875
<i>Validation Accuracy</i>	0.7089
<i>Loss</i>	0.0776
<i>Validation Loss</i>	0.8099
<i>Precision</i>	0.9922
<i>Validation Precision</i>	0.9373
<i>Recall</i>	0.9871
<i>Validation Recall</i>	0.7054
<i>Average Epoch Runtime</i>	14.5462

- A arquitetura híbrida CNN + CfC com camada *fully-connected*, testada com os mesmos dois métodos de inicialização (Tabela 15) e (Tabela 16).

Tabela 15 - Resultados do teste: CNN + CfC (com camada *fully-connected*) com dados estáticos, 100 *epochs* e *Glorot Uniform*

Métrica	Valor
<i>Accuracy</i>	0.9996
<i>Validation Accuracy</i>	0.7357
<i>Loss</i>	0.0076
<i>Validation Loss</i>	0.9321
<i>Precision</i>	0.9996
<i>Validation Precision</i>	0.7527
<i>Recall</i>	0.9996
<i>Validation Recall</i>	0.7286
<i>Average Epoch Runtime</i>	14.2309

Tabela 16 - Resultados do teste: CNN + CfC (com camada *fully-connected*) com dados estáticos, 100 *epochs* e PSJ

Métrica	Valor
<i>Accuracy</i>	0.9991
<i>Validation Accuracy</i>	0.7286
<i>Loss</i>	0.0125
<i>Validation Loss</i>	0.8364
<i>Precision</i>	0.9996
<i>Validation Precision</i>	0.8529
<i>Recall</i>	0.9991
<i>Validation Recall</i>	0.7232
<i>Average Epoch Runtime</i>	14.5454

A motivação por detrás desta abordagem prende-se com o facto de que a qualidade da inicialização dos pesos pode afetar significativamente a velocidade de convergência e a capacidade de generalização dos modelos neuronais, no entanto pensamos que a pequena dimensão do *dataset* possa ter sido um fator limitador nesta avaliação.

3.4.5. Quinta abordagem

Na quinta abordagem, o foco centrou-se na avaliação das arquiteturas LTC e CfC, mas agora aplicadas a dados temporais. A principal motivação desta abordagem foi explorar a capacidade destas redes em modelar dependências temporais, uma vez que fenómenos meteorológicos, como a precipitação, evoluem ao longo do tempo e apresentam padrões complexos que podem beneficiar de uma análise sequencial.

Para isso, cada amostra foi composta por uma sequência de três imagens consecutivas de radar, com um intervalo de uma hora entre elas. Este formato foi devidamente adaptado na estrutura das redes, permitindo à camada líquida processar a sequência de forma recorrente e contínua no tempo.

Foram então obtidos os seguintes resultados com 100 épocas de treino:

- LTC com dados temporais (Tabela 17);

Tabela 17 - Resultados do teste: LTC com dados temporais e 100 epochs

Métrica	Valor
<i>Accuracy</i>	0.775
<i>Validation Accuracy</i>	0.69
<i>Loss</i>	0.552
<i>Validation Loss</i>	0.7879
<i>Precision</i>	0.986
<i>Validation Precision</i>	0.9688
<i>Recall</i>	0.6935
<i>Validation Recall</i>	0.586
<i>Average Epoch Runtime</i>	66.967

- CfC com dados temporais (Tabela 18).

Tabela 18 - Resultados do teste: CfC com dados temporais e 100 epochs

Métrica	Valor
<i>Accuracy</i>	1.0
<i>Validation Accuracy</i>	0.642
<i>Loss</i>	0.0019
<i>Validation Loss</i>	1.063
<i>Precision</i>	1.0
<i>Validation Precision</i>	0.6625
<i>Recall</i>	1.0
<i>Validation Recall</i>	0.632
<i>Average Epoch Runtime</i>	8.6883

3.4.6. Sexta abordagem

Nesta abordagem foi novamente avaliada a aplicação de dados temporais, mas agora às arquiteturas híbridas. Tal como nas variantes puras da abordagem anterior, os dados de entrada consistiram em sequências de três imagens de radar, captadas com um intervalo de uma hora entre cada uma.

As estruturas testadas mantiveram o conceito das anteriores versões híbridas, onde a CNN é responsável pela extração de características espaciais, sendo cada uma das suas camadas envolvida num bloco *TimeDistributed*, permitindo processar as sequências de forma independente por imagem. Após a camada *Flatten*, os vetores resultantes são encaminhados para a LNN, responsável pela exploração das dinâmicas temporais.

Foram testadas as seguintes arquiteturas:

- CNN + LTC com 100 épocas (Tabela 19) e com 200 épocas (Tabela 20);

Tabela 19 - – Resultados do teste: CNN + LTC (sem camada *fully-connected*) com dados temporais e 100 epochs

Métrica	Valor
<i>Accuracy</i>	0.9585
<i>Validation Accuracy</i>	0.81
<i>Loss</i>	0.173
<i>Validation Loss</i>	0.5447
<i>Precision</i>	0.9864
<i>Validation Precision</i>	0.9384
<i>Recall</i>	0.9495
<i>Validation Recall</i>	0.796
<i>Average Epoch Runtime</i>	119.2709

Tabela 20 - Resultados do teste: CNN + LTC (sem camada *fully-connected*) com dados temporais e 200 epochs

Métrica	Valor
<i>Accuracy</i>	1.0
<i>Validation Accuracy</i>	0.796
<i>Loss</i>	0.0092
<i>Validation Loss</i>	0.607
<i>Precision</i>	1.0
<i>Validation Precision</i>	0.95
<i>Recall</i>	1.0
<i>Validation Recall</i>	0.788
<i>Average Epoch Runtime</i>	117.7742

- CNN + CfC com 100 épocas (Tabela 21) e com 200 épocas (Tabela 22).

Tabela 21 - Resultados do teste: CNN + CfC (sem camada *fully-connected*) com dados temporais e 100 epochs

Métrica	Valor
<i>Accuracy</i>	1.0
<i>Validation Accuracy</i>	0.672
<i>Loss</i>	0.0017
<i>Validation Loss</i>	1.5988
<i>Precision</i>	1.0
<i>Validation Precision</i>	0.6763
<i>Recall</i>	1.0
<i>Validation Recall</i>	0.672
<i>Average Epoch Runtime</i>	63.1517

Tabela 22 - Resultados do teste: CNN + CfC (sem camada *fully-connected*) com dados temporais e 200 epochs

Métrica	Valor
<i>Accuracy</i>	1.0
<i>Validation Accuracy</i>	0.714
<i>Loss</i>	0.0004
<i>Validation Loss</i>	1.4399
<i>Precision</i>	1.0
<i>Validation Precision</i>	0.7172
<i>Recall</i>	1.0
<i>Validation Recall</i>	0.712
<i>Average Epoch Runtime</i>	62.8383

O teste com 100 épocas teve como objetivo manter a comparabilidade com as abordagens anteriores, enquanto o treino com 200 épocas procurou avaliar o impacto de uma maior profundidade de treino.

3.4.7. Abordagem final

Após a conclusão das abordagens anteriores, foi possível identificar a arquitetura com melhor desempenho, tendo como principal critério a *validation accuracy*. A rede híbrida CNN + LTC aplicada a dados temporais destacou-se das restantes, revelando-se mais eficaz na identificação de padrões espaciais e temporais relevantes à previsão da precipitação.

Com base neste resultado, foi realizada uma nova fase de treino com o objetivo de ajustar o modelo para uma aplicação prática, nomeadamente a previsão da precipitação a curto prazo. Para isso, foram realizados três novos treinos de 100 épocas cada, todos utilizando o carregamento dos melhores pesos previamente guardados (através da função *model_checkpoint*), garantindo assim partida otimizada para o novo ajuste fino.

Em cada treino, o modelo foi treinado para prever a precipitação com um intervalo de 1 hora (Tabela 23), 2 horas (Tabela 24) e 3 horas (Tabela 25).

Tabela 23 - Resultados do teste: CNN + LTC (sem camada *fully-connected*) com dados temporais e 100 epochs para previsões de 1h (*load best weights*)

Métrica	Valor
<i>Accuracy</i>	1.0
<i>Validation Accuracy</i>	0.974
<i>Loss</i>	0.0106
<i>Validation Loss</i>	0.1135
<i>Precision</i>	1.0
<i>Validation Precision</i>	0.9772
<i>Recall</i>	1.0
<i>Validation Recall</i>	0.97
<i>Average Epoch Runtime</i>	118.5918

Tabela 24 - Resultados do teste: CNN + LTC (sem camada *fully-connected*) com dados temporais e 100 epochs para previsões de 2h (*load best weights*)

Métrica	Valor
<i>Accuracy</i>	1.0
<i>Validation Accuracy</i>	0.9678
<i>Loss</i>	0.0109
<i>Validation Loss</i>	0.1357
<i>Precision</i>	1.0
<i>Validation Precision</i>	0.9696
<i>Recall</i>	1.0
<i>Validation Recall</i>	0.9658
<i>Average Epoch Runtime</i>	119.1092

Tabela 25 - Resultados do teste: CNN + LTC (sem camada *fully-connected*) com dados temporais e 100 epochs para previsões de 3h (*load best weights*)

Métrica	Valor
<i>Accuracy</i>	0.999
<i>Validation Accuracy</i>	0.8875
<i>Loss</i>	0.0261
<i>Validation Loss</i>	0.3466
<i>Precision</i>	0.999
<i>Validation Precision</i>	0.9031
<i>Recall</i>	0.999
<i>Validation Recall</i>	0.8735
<i>Average Epoch Runtime</i>	118.6177

Esta configuração visa simular cenários reais de previsão e permitir a sua integração em aplicações práticas, como o website desenvolvido no âmbito deste projeto, onde o modelo poderá ser utilizado para prever a precipitação nas próximas horas com base em imagens de radar em tempo real.

4. Aplicação real do modelo

Após a criação, treino e validação do modelo de previsão de precipitação com recurso a LNN, foi desenvolvida uma aplicação prática para tornar o modelo acessível de forma simples e intuitiva. Esta aplicação é composta por duas partes principais: uma API desenvolvida com *Flask*, que disponibiliza o modelo de previsão, e uma interface *web*, chamada Web Meteo, que permite ao utilizador visualizar as imagens de radar e obter previsões de precipitação em tempo real.

A aplicação foi pensada com o objetivo de demonstrar, de forma concreta, como este tipo de tecnologia pode ser integrado em soluções reais, contribuindo para a sua adoção em contextos práticos, como sistemas de apoio à decisão em meteorologia local ou aplicações educativas.

As próximas secções descrevem com maior detalhe o funcionamento da API, a interface da aplicação *web* e os passos necessários para a sua instalação e utilização.

4.1. API em *Flask*

A componente de *backend* da aplicação foi desenvolvida utilizando o *micro-framework* *Flask*, que permite criar APIs REST de forma simples e eficiente. Esta API tem como função principal servir o modelo de previsão de precipitação, disponibilizando *endpoints* que automatizam tanto a obtenção de imagens de radar como a geração de previsões com base nas mesmas.

A API utiliza o modelo treinado na abordagem final e carrega os pesos previamente guardados em ficheiros com extensão .h5, correspondentes às previsões de 1h, 2h e 3h no futuro. Esta modularidade permite aplicar o mesmo processo de inferência para diferentes janelas temporais.

Nas secções seguintes são descritos os dois *endpoints* disponibilizados pela API.

4.1.1. Endpoint /radar-images

Este *endpoint* é responsável por obter uma sequência de 3 imagens de radar disponibilizadas pelo IPMA (Figura 39). O processo inicia-se pela identificação do múltiplo de 5 minutos mais recente (com base na hora atual), ao qual se subtraem 10 minutos, de forma a maximizar a probabilidade de a imagem mais recente já estar disponível nos servidores do IPMA. As duas imagens anteriores correspondem aos registo de 1 hora e 2 horas atrás, respetivamente, garantindo um intervalo temporal adequado à entrada do modelo.

```
@app.route(rule: '/radar-images', methods=['GET'])
def radar_images():
    try:
        images = get_radar_images_sequence(SEQUENCE_LENGTH)
        images_strs = []
        for img in images:
            if img is not None:
                buffered = io.BytesIO()
                img.save(buffered, format="PNG")
                img_str = base64.b64encode(buffered.getvalue()).decode('utf-8')
                images_strs.append(img_str)
            else:
                images_strs.append(None)

        return jsonify({
            'image1': images_strs[0],
            'image2': images_strs[1],
            'image3': images_strs[2]
        })
    except Exception as e:
        print(e)
        return jsonify({'error': 'Ocorreu um erro'}), 500
```

Figura 39 - Código do endpoint /radar-images

4.1.2. *Endpoint /predict-rain*

Este *endpoint* utiliza a sequência de imagens obtida pelo *endpoint* anterior e realiza o seguinte processo (Figura 40):

1. Segmentação por distrito – as imagens de radar são cortadas de acordo com os limites geográficos de cada distrito de Portugal Continental.
2. Pré-processamento – os dados são preparados conforme o formato esperado pelo modelo (dimensões, normalização, etc.).
3. Inferência – para cada distrito, são feitas previsões de precipitação para 1h, 2h e 3h no futuro, recorrendo aos ficheiros .h5 correspondentes.
4. Resposta *JSON* – os resultados são devolvidos no formato *JSON*, indicando o valor previsto de precipitação para cada distrito e horizonte temporal.

```
@app.route(rule: '/predict-rain', methods=['GET'])
def predict_rain():
    try:
        images = get_radar_images_sequence(SEQUENCE_LENGTH)
        full_hour_prediction_dictionary = {}
        model_weights_list = os.listdir("model_weights/")
        for model_weights in model_weights_list:
            model.load_weights("model_weights/" + str(model_weights))
            current_hour_prediction_dictionary = {}
            for station_id in STATIONS.keys():
                station_images = []
                for img in images:
                    image = process_radar_image(img, station_id)
                    station_images.append(image)
                predictions = model.predict(np.expand_dims(station_images, axis=0))
                current_hour_prediction_dictionary[STATIONS[station_id]['district']] = int(np.argmax(predictions[0]))
            full_hour_prediction_dictionary[extract_number_from_filename(model_weights)] = current_hour_prediction_dictionary
        return json.dumps(full_hour_prediction_dictionary)
    except Exception as e:
        print(e)
        return jsonify({'error': 'Ocorreu um erro'}), 500
```

Figura 40 - Código do *endpoint* /predict-rain

4.2. Web Meteo

A interface *web* desenvolvida para esta aplicação tem como objetivo fornecer ao utilizador uma forma intuitiva e interativa de visualizar as previsões de precipitação geradas pelo modelo. A aplicação foi construída com recurso ao *framework Vue.js*, que permite uma separação clara de componentes e uma renderização eficiente do conteúdo dinâmico.

Na figura seguinte (Figura 41), estão representados os elementos fundamentais desta interface *web*. Do lado esquerdo, encontra-se a tabela com as previsões de precipitação para cada distrito e do lado direito, é apresentado o mapa de Portugal Continental, sobre o qual são integradas as imagens de radar fornecidas pelo IPMA. A intensidade de precipitação (em mm/h) é representada visualmente através de manchas em tons de azul.

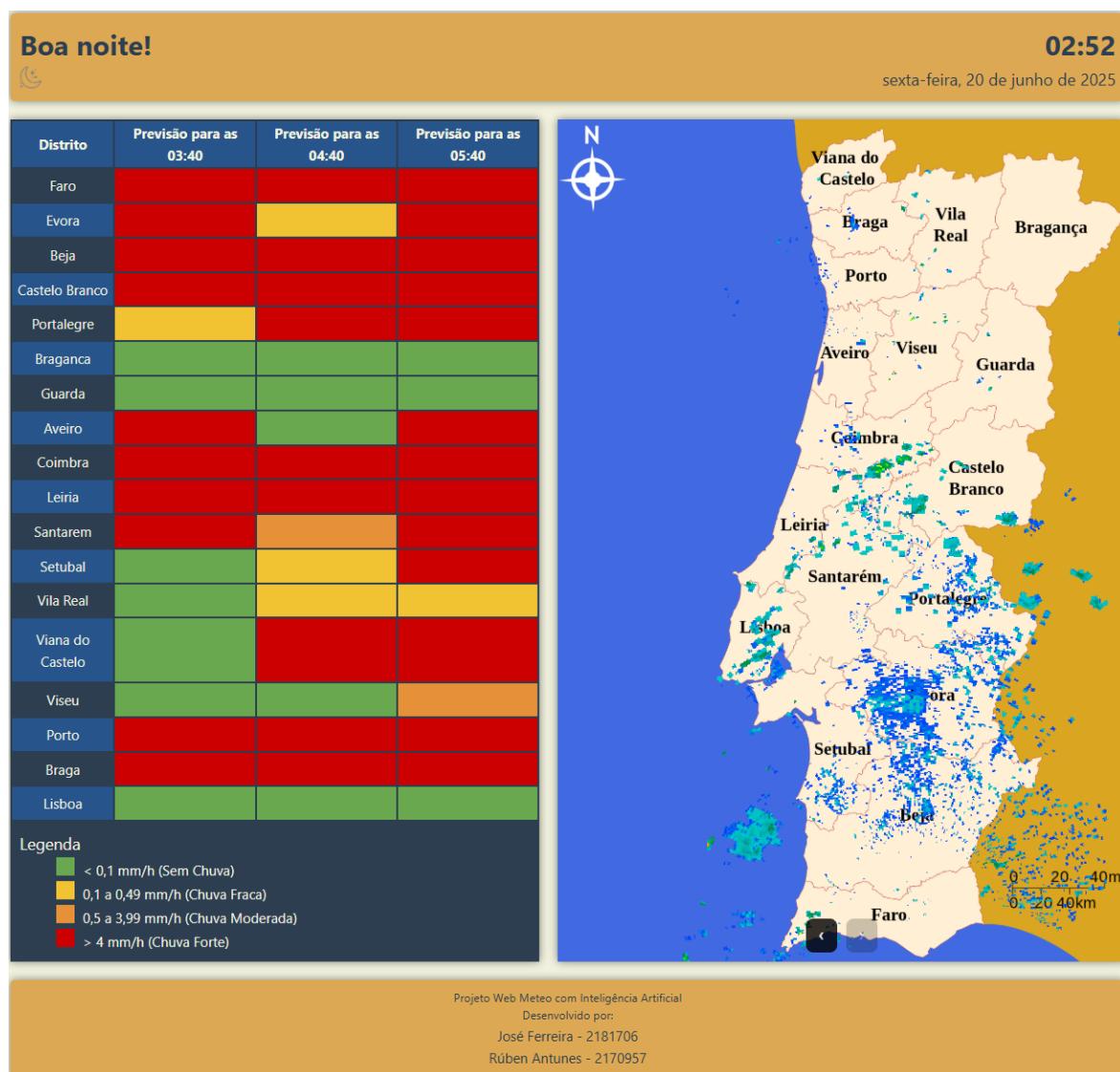


Figura 41 - Website Web Meteo

O *frontend* comunica diretamente com a API desenvolvida, tanto para a obtenção das imagens de radar como para os resultados das previsões. A interface é composta por vários componentes, cada um com uma função específica na experiência do utilizador, contribuindo para uma visualização clara, estruturada e acessível da informação meteorológica. A seguir, cada um dos componentes será descrito em detalhe.

4.2.1. Clock.vue

O componente Clock.vue é responsável por apresentar ao utilizador a data e hora atual, atualizada em tempo real. Para além disso, o componente também exibe uma saudação personalizada, que varia consoante o período do dia em que o utilizador acede à aplicação.

Esta funcionalidade adiciona um toque de personalização e interatividade, o que torna a experiência do utilizador mais amigável e acolhedora.

De seguida é apresentado o código correspondente ao *template* (Figura 42) e ao *script* (Figura 43) do componente.

```
<template>
  <div class="clock-container">
    <div class="greeting-container">
      <span class="greeting">{{ greeting }}</span>
      
    </div>
    <div class="date-container">
      <div class="clock-time">{{ formattedTime }}</div>
      <div class="clock-date">{{ formattedDate }}</div>
    </div>
  </div>
</template>
```

Figura 42 - Código do *template* do Clock.vue

```

<script> Show component usages
export default { Show usages
Windsurf: Refactor | Explain | Docstring | ×
data() {
  return {
    formattedTime: '',
    formattedDate: '',
    greeting: '',
    iconPath: '',
    iconAlt: ''
  };
},
Windsurf: Refactor | Explain | Docstring | ×
mounted() {
  this.updateTime();
  setInterval(this.updateTime, timeout: 60000); // 1 minute
},
methods: {
  Windsurf: Refactor | Explain | Docstring | ×
  updateTime() {
    const now = new Date();
    const timeOptions = {
      hour: 'numeric',
      minute: 'numeric'
    };
    this.formattedTime = now.toLocaleString( locales: 'pt-PT', timeOptions);
    const dateOptions = {
      weekday: 'long',
      day: 'numeric',
      month: 'long',
      year: 'numeric'
    };
    this.formattedDate = now.toLocaleString( locales: 'pt-PT', dateOptions);
    const hour = now.getHours();
    if (hour >= 5 && hour < 12) {
      this.greeting = 'Bom dia!';
      this.iconPath = '../src/assets/sun.png';
      this.iconAlt = 'Sol';
    } else if (hour >= 12 && hour < 18) {
      this.greeting = 'Boa tarde!';
      this.iconPath = '../src/assets/sun.png';
      this.iconAlt = 'Sol';
    } else {
      this.greeting = 'Boa noite!';
      this.iconPath = '../src/assets/moon.png';
      this.iconAlt = 'Lua';
    }
  }
};
</script>

```

Figura 43 - Código do script do Clock.vue

4.2.2. MapImage.vue

O componente MapImage.vue é responsável por realizar o pedido de obtenção das imagens de radar à API, através do *endpoint* definido para o efeito.

Após a receção das imagens, o componente trata de integrá-las visualmente sobre um mapa estático do território nacional, permitindo assim ao utilizador observar as alterações da precipitação ao longo do tempo. As imagens são posicionadas com precisão sobre o mapa, respeitando a escala e os limites geográficos dos distritos, proporcionando uma visualização clara e informativa.

Além da sobreposição das imagens, o componente oferece controlos de navegação temporal, que permitem ao utilizador alternar entre as diferentes imagens (com intervalos de 1 hora), funcionando como uma pequena “animação” ou linha temporal da evolução da precipitação. Estes controlos ajudam a reforçar a compreensão do utilizador sobre o movimento e intensidade da precipitação recente.

A integração entre os dados proveniente da API e a sua representação visual no *frontend* é um dos pontos centrais da aplicação, sendo este componente essencial para garantir uma experiência interativa e informativa.

De seguida estão representados o código correspondente ao *template* (Figura 44) e ao *script* (Figura 45) do componente.

```
<template>
  <div class="map-container">
    
    
    <div class="slider-controls">
      <button @click="prevImage" :disabled="currentImgIndex === 0"></button>
      <button @click="nextImage" :disabled="currentImgIndex === radarImages.length - 1"></button>
    </div>
  </div>
</template>
```

Figura 44 - Código do *template* do MapImage.vue

```

<script> Show component usages
import axios from "axios";

export default { Show usages
  name: "MapImage",
  Windsurf: Refactor | Explain | Docstring | X
  data() {
    return {
      radarImages: [],
      currentImgIndex: 0,
    };
  },
  Windsurf: Refactor | Explain | Docstring | X
  async mounted() {
    this.getRadarImages();
  },
  methods: {
    Windsurf: Refactor | Explain | Docstring | X
    getRadarImages() {
      console.log("Getting radar images...");
      axios
        .get( url: "http://localhost:5000/radar-images")
        .then((response) => {
          const data = response.data;
          this.radarImages = [
            `data:image/png;base64,${data.image1}`,
            `data:image/png;base64,${data.image2}`,
            `data:image/png;base64,${data.image3}`,
          ];
          this.currentImgIndex = 2;
          console.log("Radar images received!");
        })
        .catch((error) => {
          console.log(error);
        });
    },
    Windsurf: Refactor | Explain | Docstring | X
    nextImage() {
      if (this.currentImgIndex < this.radarImages.length - 1) {
        this.currentImgIndex++;
      }
    },
    Windsurf: Refactor | Explain | Docstring | X
    prevImage() {
      if (this.currentImgIndex > 0) {
        this.currentImgIndex--;
      }
    },
  },
};

</script>

```

Figura 45 - Código do *script* do MapImage.vue

4.2.3. PredictionsTable.vue

O componente PredictionsTable.vue é responsável por realizar o pedido ao *endpoint* /predict-rain da API, que fornece as previsões de precipitação com base na sequência de imagens de radar previamente obtida.

Após receber a resposta da API, o componente processa e apresenta os resultados numa tabela organizada, onde cada linha corresponde a um dos distritos de Portugal Continental, e as colunas representam a previsão de chuva para as próximas 1, 2 e 3 horas.

As previsões fornecidas são apresentadas utilizando um código de cores, onde cada cor representa a intensidade de precipitação prevista. Neste caso, verde indica ausência de chuva, amarelo corresponde a chuva fraca, laranja a chuva moderada e vermelho a chuva forte. Esta abordagem facilita a leitura imediata da previsão de cada distrito.

As figuras seguintes incluem o código do *template* (Figura 46) e do *script* (Figura 47) do componente.

```
<template>
  <div v-if="isLoading" class="text-placeholder">
    <p>A calcular previsões...</p>
  </div>
  <div v-else class="table-container">
    <table class="table-data">
      <thead>
        <tr>
          <th>Distrito</th>
          <th>Previsão para as {{ time_delay1h }}</th>
          <th>Previsão para as {{ time_delay2h }}</th>
          <th>Previsão para as {{ time_delay3h }}</th>
        </tr>
      </thead>
      <tbody>
        <tr v-for="(item, key) in formattedData" :key="key">
          <td>{{ key }}</td>
          <td :style="getStyle(item[0])"></td>
          <td :style="getStyle(item[1])"></td>
          <td :style="getStyle(item[2])"></td>
        </tr>
      </tbody>
    </table>
    <div class="table-legends">
      <h3>Legenda</h3>
      <ul>
        <li><span class="color-box" style="...></span> < 0,1 mm/h (Sem Chuva)</li>
        <li><span class="color-box" style="...></span> 0,1 a 0,49 mm/h (Chuva Fraca)</li>
        <li><span class="color-box" style="...></span> 0,5 a 3,99 mm/h (Chuva Moderada)</li>
        <li><span class="color-box" style="...></span> > 4 mm/h (Chuva Forte)</li>
      </ul>
    </div>
  </div>
</template>
```

Figura 46 - Código do *template* do PredictionsTable.vue

```

<script> Show component usages
import axios from "axios";

export default { Show usages
  name: "PredictionsTable",
  data() {
    return {
      formattedData: {},
      isLoading: true,
      time_delay1h: "",
      time_delay2h: "",
      time_delay3h: ""
    };
  },
  mounted() {
    this.fetchData();
  },
  methods: {
    fetchData() {
      this.isLoading = true;
      this.formattedData = {};
      this.time_delay1h = "";
      this.time_delay2h = "";
      this.time_delay3h = "";
      console.log("Fetching data...");
      axios
        .get(url: "http://localhost:5000/predict-rain")
        .then((response) => {
          const data = response.data;
          for (const [key, value] of Object.entries(data)) {
            for (const [key2, value2] of Object.entries(value)) {
              if (!this.formattedData[key2]) {
                this.formattedData[key2] = [];
              }
              this.formattedData[key2].push(value2);
            }
          }
          this.isLoading = false;
          console.log(this.formattedData);
          // Obter a hora atual em UTC
          var current_datetime_utc = new Date();
          // Arredondar a hora atual para o múltiplo de 5 mais próximo
          var rounded_datetime = new Date(current_datetime_utc);
          rounded_datetime.setMinutes(
            min: Math.floor(x: rounded_datetime.getMinutes() / 5) * 5
          );
          // Atrasar a hora em 10 minutos
          var datetime_delay = new Date(
            value: rounded_datetime.getTime() - 10 * 60000 + 3600000
          );
    }
  }
}

```

```

    // Converter a hora para o formato hh:mm
    this.time_delay1h = datetime_delay.toLocaleTimeString( locales: [], options: {
      hour: "2-digit",
      minute: "2-digit",
    });
    this.time_delay2h = new Date(
      value: datetime_delay.getTime() + 3600000
    ).toLocaleTimeString( locales: [], options: {
      hour: "2-digit",
      minute: "2-digit",
    });
    this.time_delay3h = new Date(
      value: datetime_delay.getTime() + 2 * 3600000
    ).toLocaleTimeString( locales: [], options: {
      hour: "2-digit",
      minute: "2-digit",
    });
  )
  .catch(error) => {
  | console.error(error);
  );
},
getStyle(value) {
  switch (value) {
    case 0:
    | return { backgroundColor: 'var(--vt-c-warning-green)' }; // Sem Chuva
    case 1:
    | return { backgroundColor: 'var(--vt-c-warning-yellow)' }; // Chuva Fraca
    case 2:
    | return { backgroundColor: 'var(--vt-c-warning-orange)' }; // Chuva Moderada
    case 3:
    | return { backgroundColor: 'var(--vt-c-warning-red)' }; // Chuva Forte
    default:
    | return {};
  }
},
},
};

</script>

```

Figura 47 - Código do script do PredictionsTable.vue

4.3. Instalação e uso

Para permitir o funcionamento da aplicação Web Meteo, é necessário configurar corretamente tanto a API (*backend*) como a interface *web* (*frontend*). Esta secção descreve os passos necessários para instalar e executar a aplicação localmente, incluindo os requisitos técnicos e as instruções de inicialização.

4.3.1. Requisitos

Para correr o projeto, são necessários os seguintes pré-requisitos:

- Python: 3.12.10
- Node.js: 24.2.0
- Npm: 11.3.0
- Bibliotecas Python:
 - Flask-cors: 6.0.0
 - Requests: 2.32.3
 - Pillow: 11.2.1
 - Numpy: 2.1.3
 - Tensorflow: 2.19.0
 - Ncps: 1.0.1
- Framework frontend:
 - Vue: 3.5.15
 - Axios: 1.9.0

As bibliotecas *Python* podem ser instaladas com o seguinte comando:

- `Pip install -r requirements.txt`

E para o *frontend*:

- `Npm install`

4.3.2. Instruções

Para correr a aplicação localmente, siga os seguintes passos:

1. Iniciar a API *Flask*:

No diretório onde está localizada a API execute: `py app.py run`

2. Iniciar o cliente *Vue*:

No diretório do *frontend* execute: `npm run dev`

3. Aceder à aplicação:

No *browser* aceder ao endereço `http://localhost:5173`

5. Análise de resultados

Após a implementação, treino e validação dos diferentes modelos ao longo das várias abordagens descritas neste relatório, procede-se agora à análise crítica e comparativa dos resultados obtidos. Esta análise visa identificar padrões de desempenho, destacar os modelos com melhores resultados e justificar as diferenças observadas com base na estrutura das redes, nos dados utilizados e nas estratégias de treino aplicadas.

A análise será feita de forma estruturada, começando pelos resultados das abordagens com dados estáticos e evoluindo para os modelos que utilizam dados temporais. Cada secção irá incluir:

- Tabelas resumo com os principais indicadores de avaliação: *accuracy*, *loss* e *avg_epoch_runtime* (tempo médio por época);
- Gráficos ilustrativos da evolução do *loss* e da *accuracy* durante o treino;
- Interpretação dos resultados e comparação entre modelos semelhantes.

Por fim, será realizada uma análise específica sobre o modelo final escolhido (CNN + LTC com dados temporais), incluindo os seus desempenhos para previsões de 1h, 2h e 3h, destacando-se a sua aplicabilidade prática.

Para avaliar a eficácia e performance dos modelos desenvolvidos, foi dada especial atenção às seguintes métricas:

- *Validation_accuracy*, que indica a proporção de previsões corretas feitas pelo modelo para conjuntos de dados nunca antes vistos;
- *Accuracy*, que mede a percentagem de previsões corretas para o conjunto de dados de treino;
- *Validation_loss*, que representa a diferença entre as previsões do modelo e os valores reais, no conjunto de dados de validação;
- *Loss*, que indica o mesmo que a métrica anterior, mas para o conjunto de dados de treino.

5.1. Resultados com dados estáticos

Nesta secção, analisam-se os resultados obtidos nas quatro primeiras abordagens do projeto, nas quais foram utilizados dados estáticos, ou seja, imagens de radar isoladas. O principal objetivo foi identificar quais modelos apresentavam melhor capacidade de generalização, com base na *cross-validation*. Para além disso, procurou-se também validar a utilidade da técnica de *data augmentation* e explorar diferentes estratégias de inicialização de pesos.

A Tabela 26 apresenta um resumo dos resultados obtidos.

Tabela 26 - – Resumo dos resultados obtidos nos testes com dados estáticos

Modelo	Epochs	Acc	Val_Acc	Loss	Val_Loss	Avg_Time
CNN	50	0.992	0.6964	0.0457	1.1042	14.75
CNN	100	0.9991	0.7321	0.0144	0.9646	14.51
CNN	200	1.0	0.7286	0.0058	1.0272	13.82
CNN (No DA)	100	1.0	0.6429	0.0005	1.4967	13.40
CNN (PSJ)	100	1.0	0.7125	0.0091	0.9445	14.11
LTC	100	0.8125	0.6679	0.5051	0.8336	28.06
CfC	100	1.0	0.7143	0.0064	0.878	5.07
CNN+LTC	100	0.9719	0.7464	0.1303	0.6459	38.90
CNN+LTC (GU)	100	0.9973	0.7393	0.0225	0.7331	14.61
CNN+LTC (PSJ)	100	0.9875	0.7089	0.0776	0.8099	14.55
CNN+CfC	100	1.0	0.7161	0.0015	1.2754	16.42
CNN+CfC (GU)	100	0.9996	0.7357	0.0076	0.9321	14.23
CNN+CfC (PSJ)	100	0.9991	0.7286	0.0125	0.8364	14.55

Nos gráficos seguintes podemos observar a evolução da *accuracy* (Figura 48), da *validation_accuracy* (Figura 49), do *loss* (Figura 50) e do *validation_loss* (Figura 51) ao longo das épocas de treino.

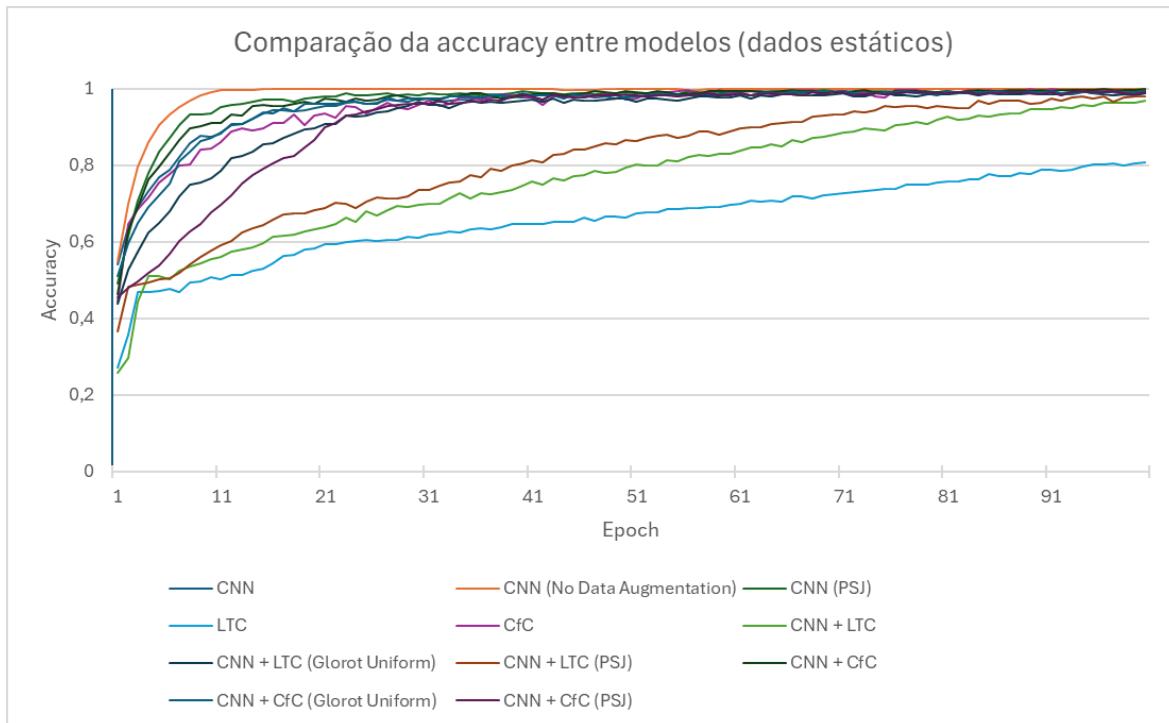


Figura 48 - Gráfico comparativo da *accuracy* em dados estáticos

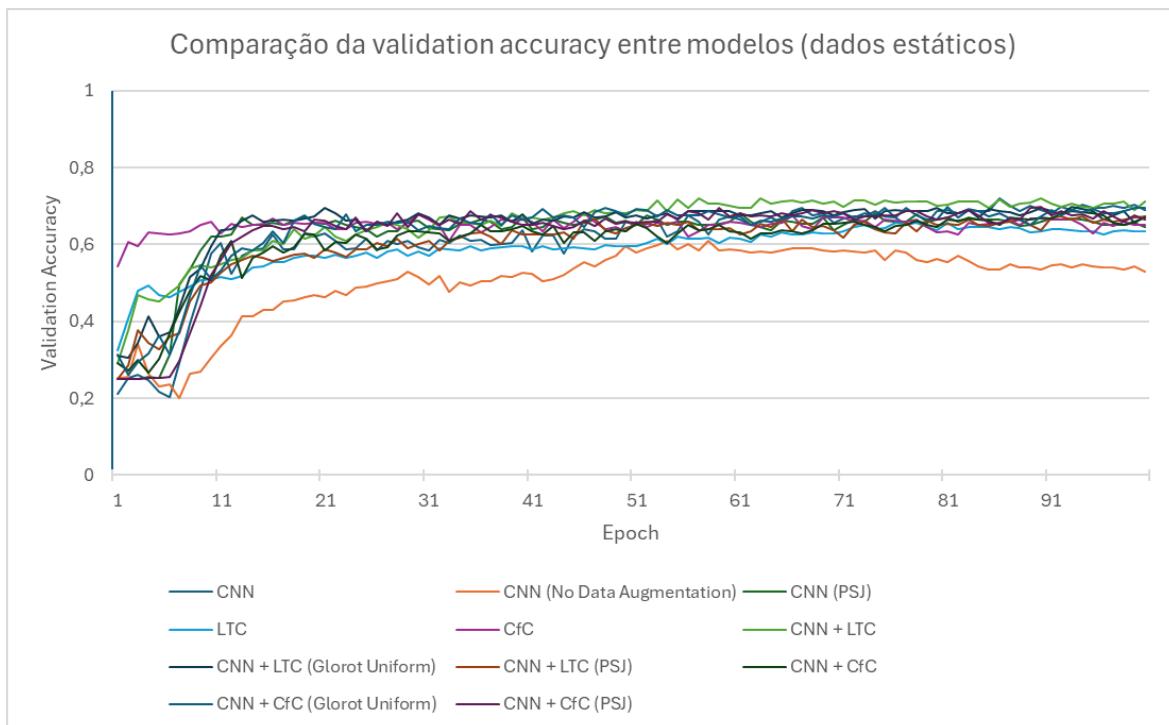
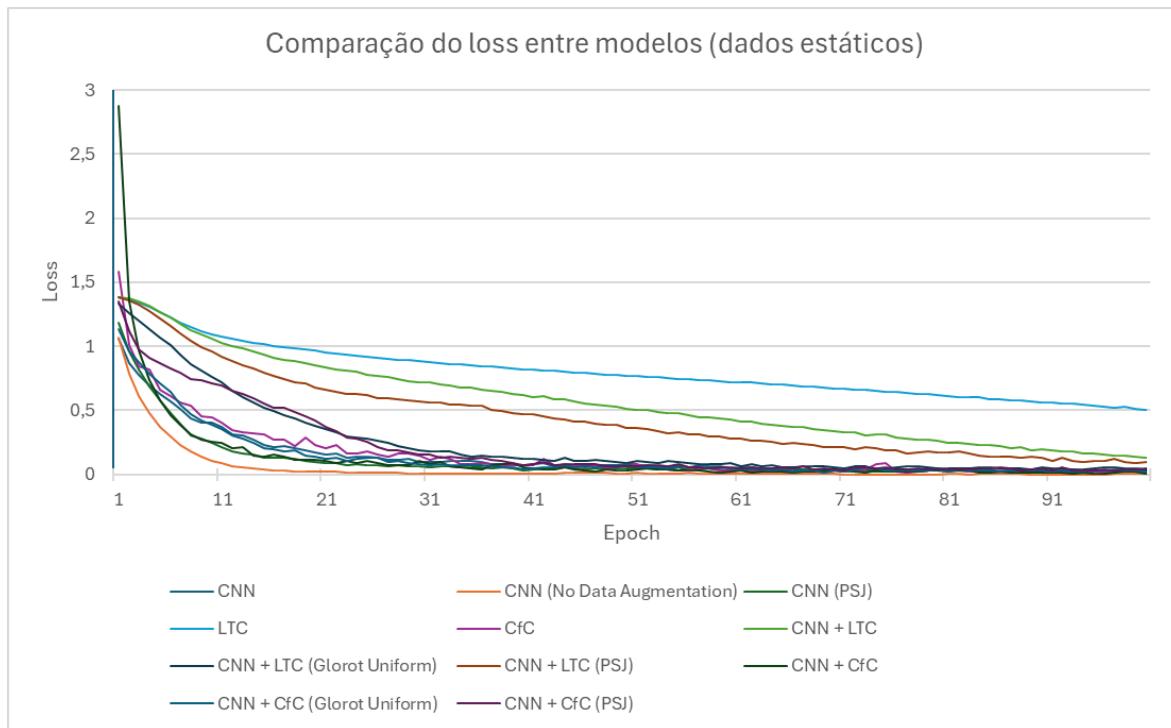
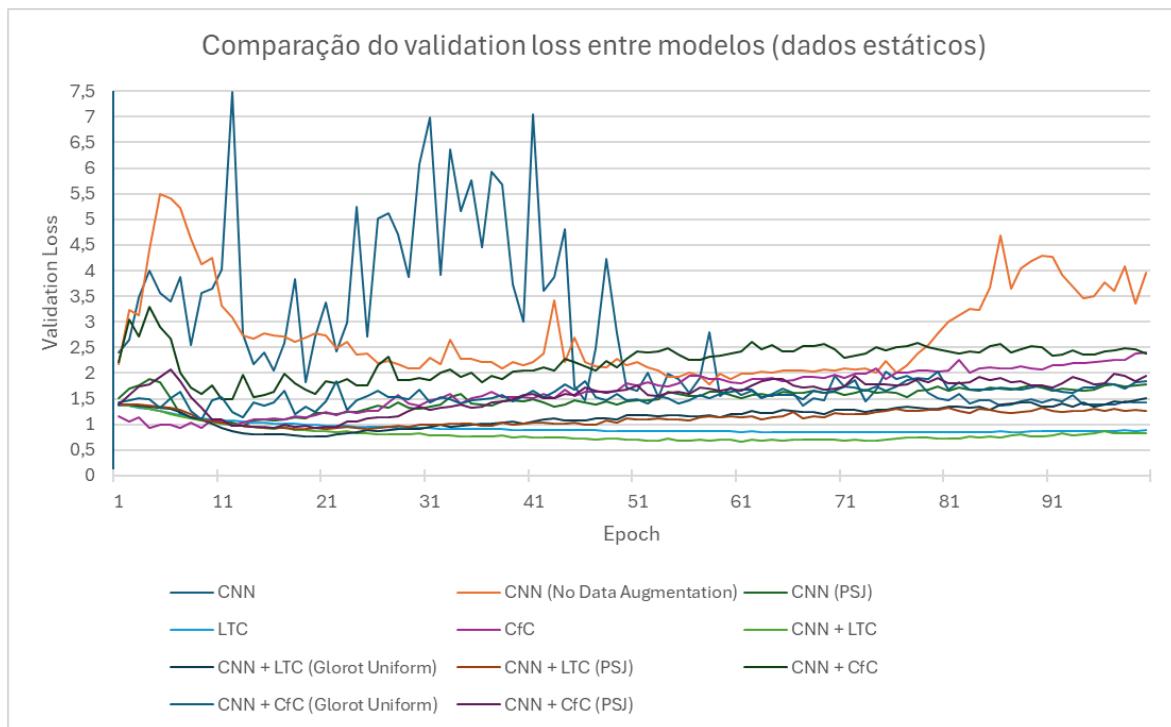


Figura 49 - Gráfico comparativo da *validation accuracy* em dados estáticos

**Figura 51** - Gráfico comparativo da *loss* em dados estáticos**Figura 50** - Gráfico comparativo da *validation loss* em dados estáticos

Analisando os resultados conseguimos observar que a CNN original demonstrou um bom desempenho, especialmente com 100 épocas e *data augmentation*, atingindo uma *validation_accuracy* de 73.21%. Com 200 épocas, não houve melhoria significativa, sugerindo possível *overfitting*. A versão sem *data augmentation* obteve um desempenho consideravelmente inferior, com *validation_accuracy* de 64.29%, confirmando a importância dessa técnica para generalização.

O método PSJ foi novamente testado como controlo, dado que no projeto anterior apresentou bons resultados. Embora tenha mantido um desempenho competitivo (*validation_accuracy* de 71.25%), não superou as melhores configurações com *Glorot Uniform*).

A utilização das camadas LTC e CfC, nas arquiteturas puramente recorrentes, teve resultados mistos. A CfC destacou-se com uma *validation_accuracy* de 71.43% e um tempo de treino muito inferior. A LTC, por outro lado, apresentou o valor mais baixo (66.79%), apesar de uma boa performance nas métricas de treino.

Importa destacar um comportamento característico das arquiteturas com LTC: ao contrário da maioria dos modelos testados, que rapidamente atingem a sua performance máxima e estabilizam, os modelos com LTC apresentam uma evolução mais gradual da *accuracy* ao longo das épocas. Este padrão sugere que, se treinados durante mais tempo, poderiam alcançar desempenhos comparáveis (ou até superiores) aos restantes modelos. Para avaliar mais rigorosamente o seu potencial, seria recomendável, em trabalhos futuros, realizar experiências com um tempo de treino mais prolongado.

Apesar de tudo, as arquiteturas híbridas demonstraram globalmente os melhores desempenhos, nomeadamente, a CNN + LTC (sem camada *fully-connected*) com uma *validation_accuracy* de 74.64%, a CNN + LTC (com camada *fully-connected* e método de inicialização *Glorot Uniform*) com um resultado de 73.93% e a CNN + CfC (com camada *fully-connected* e método de inicialização *Glorot Uniform*) com um valor de 73.57%.

5.2. Resultados com dados temporais

Nesta secção são analisados os resultados obtidos com dados temporais, correspondentes a sequências de imagens de radar. As redes foram treinadas para processar estas sequências com arquiteturas recorrentes puras (LTC e CfC) e híbridas (CNN + LTC/CfC), num primeiro momento com 100 épocas, e posteriormente com 200 épocas para os modelos híbridos.

A Tabela 27 apresenta um resumo dos principais resultados obtidos.

Tabela 27 - Resumo dos resultados obtidos nos testes com dados temporais

Modelo	Epochs	Acc	Val_Acc	Loss	Val_Loss	Avg_Time
LTC	100	0.775	0.69	0.552	0.7879	66.97
CfC	100	1.0	0.642	0.0019	1.063	8.69
CNN+LTC	100	0.9585	0.81	0.173	0.5447	119.27
CNN+LTC	200	1.0	0.796	0.0092	0.607	117.77
CNN+CfC	100	1.0	0.672	0.0017	1.5988	63.15
CNN+CfC	200	1.0	0.714	0.0004	1.4399	62.84

Os gráficos seguintes demonstram a evolução da *accuracy* (Figura 52), da *validation_accuracy* (Figura 53), do *loss* (Figura 54) e do *validation_loss* (Figura 55) ao longo das épocas de treino.

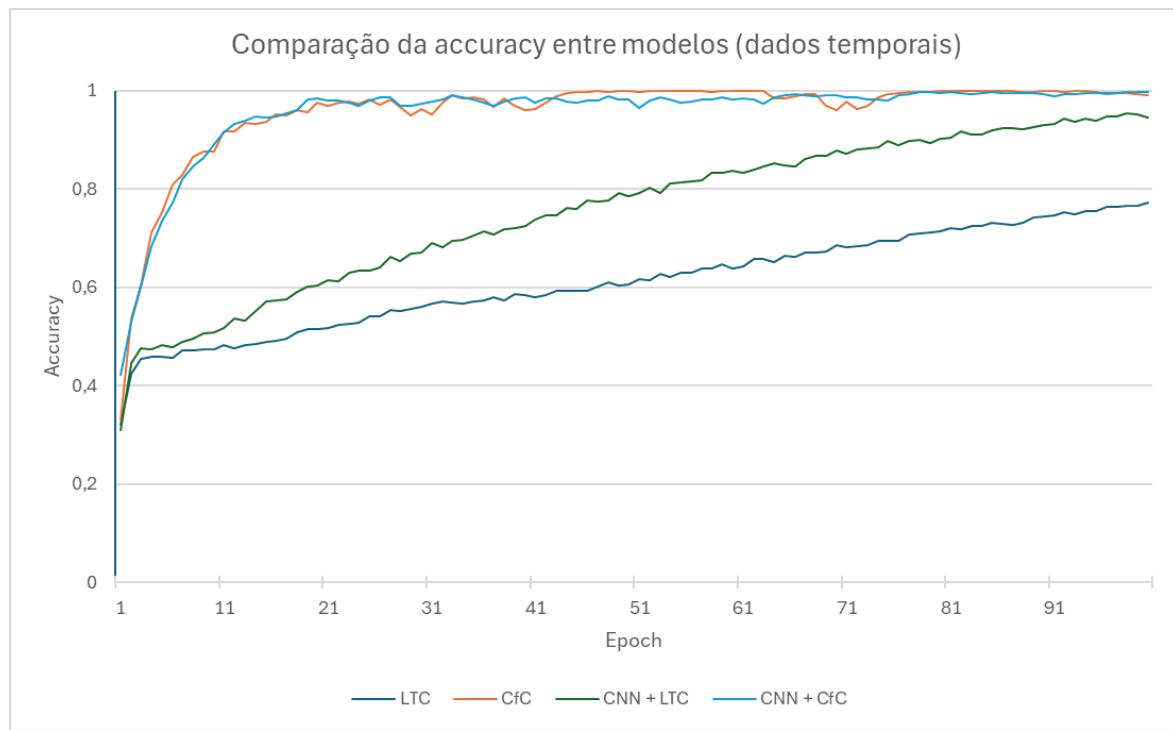


Figura 52 - Gráfico comparativo da *accuracy* em dados temporais

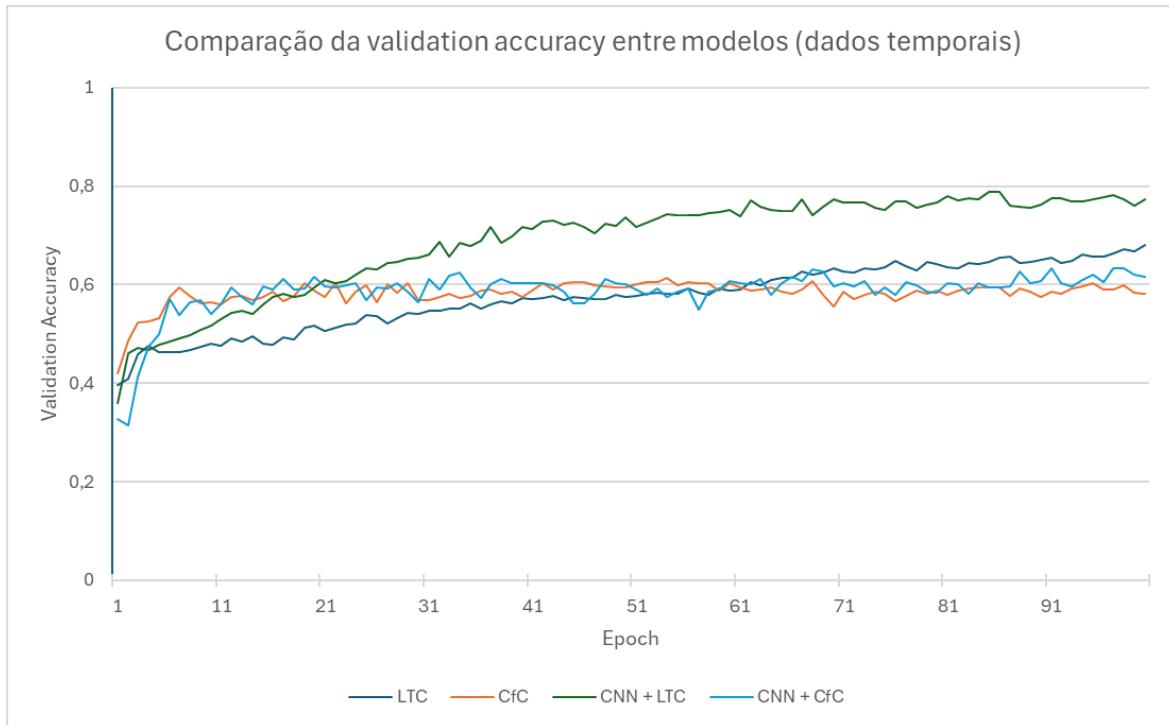


Figura 54 - Gráfico comparativo da *validation accuracy* em dados temporais

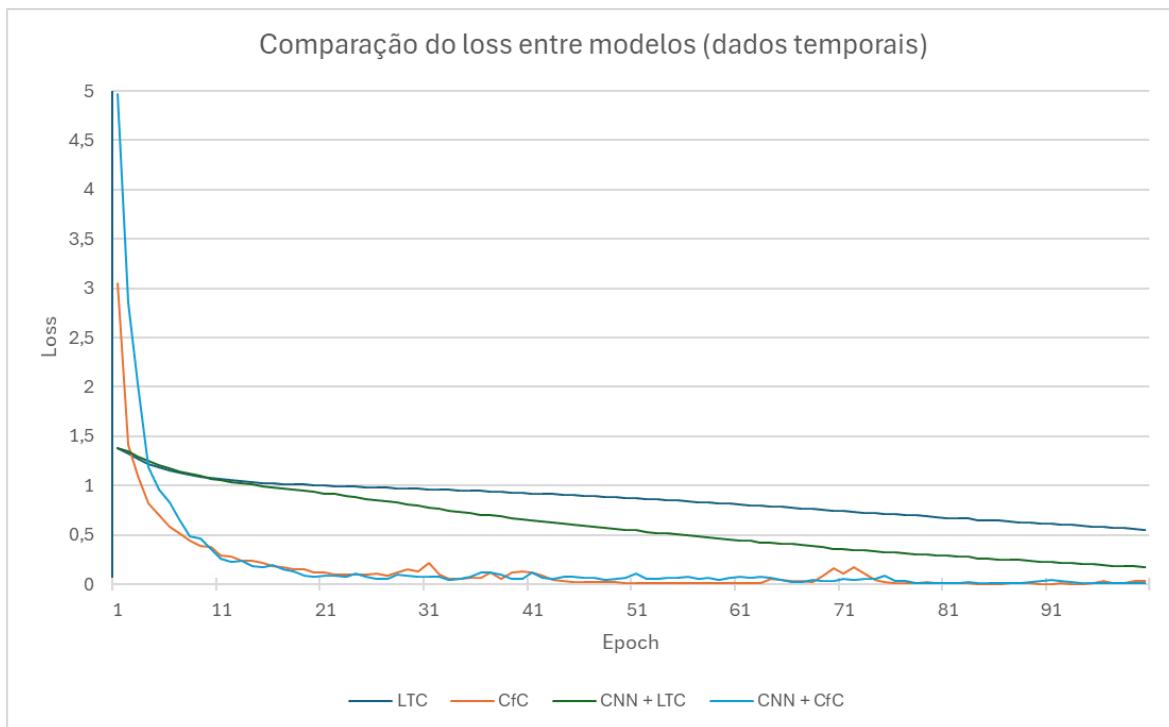


Figura 53 - Gráfico comparativo da *loss* em dados temporais

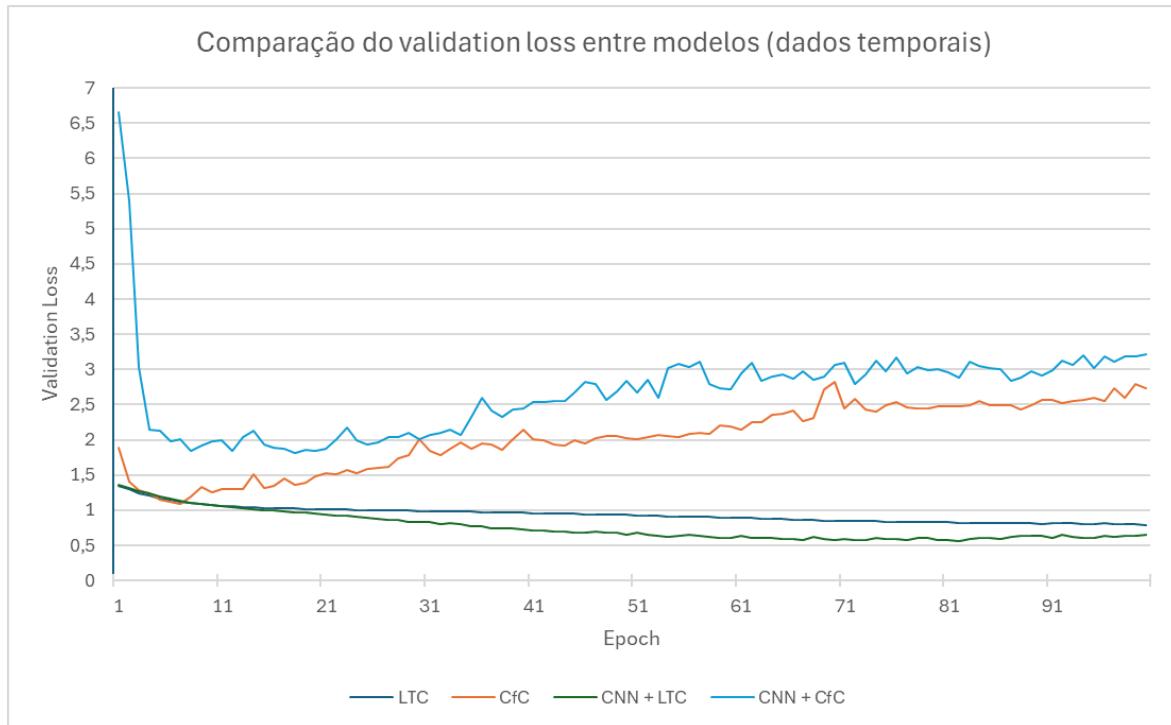


Figura 55 - Gráfico comparativo da *validation loss* em dados temporais

De forma geral, a utilização de dados temporais permitiu melhorar a capacidade de generalização dos modelos, sendo este o grupo com os melhores resultados de validação em todo o projeto.

A arquitetura híbrida CNN + LTC destacou-se como a melhor solução, obtendo uma *validation accuracy* de 81% com 100 épocas, e mantendo desempenho elevado mesmo com 200 épocas (79.60%), com métricas de *loss* e *validation loss* consistentes e equilibradas. Este resultado justifica a sua seleção para a abordagem final.

Como já observado nos dados estáticos, os modelos com LTC demonstraram um comportamento particular: a *accuracy* sobe de forma gradual e constante, sugerindo que o modelo ainda não atingiu a sua capacidade máxima após 100 épocas. Desta vez, o padrão também se refletiu na *validation accuracy*, possivelmente devido ao aumento de complexidade dos dados. Mantém-se assim a necessidade de testar com mais profundidade o potencial desta arquitetura.

Por outro lado, os modelos com Cfc obtiveram tempo de execução muito inferior e bom desempenho na *accuracy* de treino, mas uma *validation accuracy* mais fraca (64.2%), o que compromete a sua aplicabilidade prática nestas condições.

Em suma, as redes híbridas com LTC foram as que mais beneficiaram da introdução de dados temporais, e o seu comportamento ao longo do treino sugere que a escolha do número de épocas pode ter um impacto significativo nos resultados. A identificação da melhor arquitetura (CNN + LTC com dados temporais) fundamentou a sua seleção para a última fase do projeto.

5.3. Resultados na abordagem final

Com base nos resultados anteriores, foi identificada como a arquitetura com melhor desempenho a rede híbrida CNN + LTC com dados temporais. Assim, para avaliar a sua aplicabilidade prática, foram realizados novos treinos com esta arquitetura, desta vez carregando os melhores pesos obtidos durante o treino, e testando intervalos entre a imagem e o valor de precipitação de 1, 2 e 3 horas.

O objetivo desta fase foi avaliar e potenciar ao máximo a capacidade de o modelo prever a precipitação com diferentes horizontes temporais, o que é fundamental para casos de uso reais, como o website desenvolvido no contexto deste projeto.

O resumo dos resultados obtidos pode ser consultado na seguinte Tabela 28.

Tabela 28 - Resumo dos resultados obtidos nos testes com o modelo final

Modelo	Epochs	Acc	Val_Acc	Loss	Val_Loss	Avg_Time
CNN+LTC (1h)	100	1.0	0.974	0.0106	0.1135	118.59
CNN+LTC (2h)	100	1.0	0.9678	0.0109	0.1357	119.11
CNN+LTC (3h)	100	0.999	0.8875	0.0261	0.3466	118.62

Os gráficos relativos à *accuracy*, *validation_accuracy*, *loss* e *validation_loss* podem ser consultados nas Figuras 56, 57, 58 e 59 respectivamente.

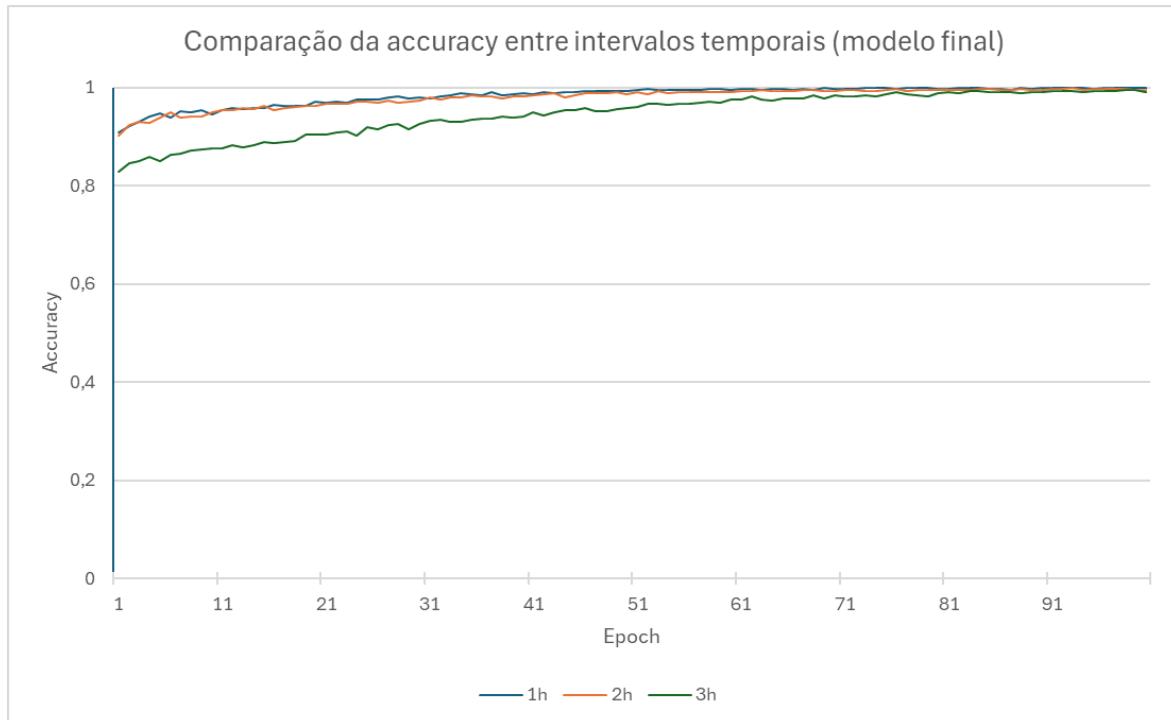


Figura 56 - Gráfico comparativo da *accuracy* no modelo final

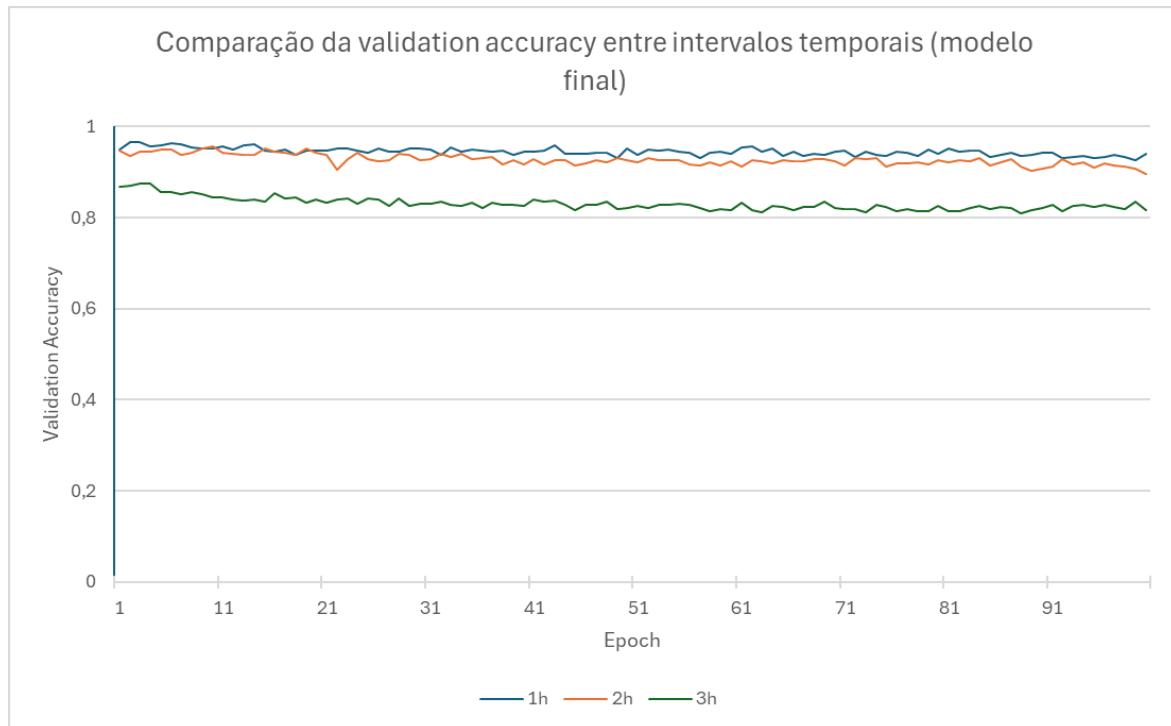


Figura 57 - Gráfico comparativo da *validation accuracy* no modelo final

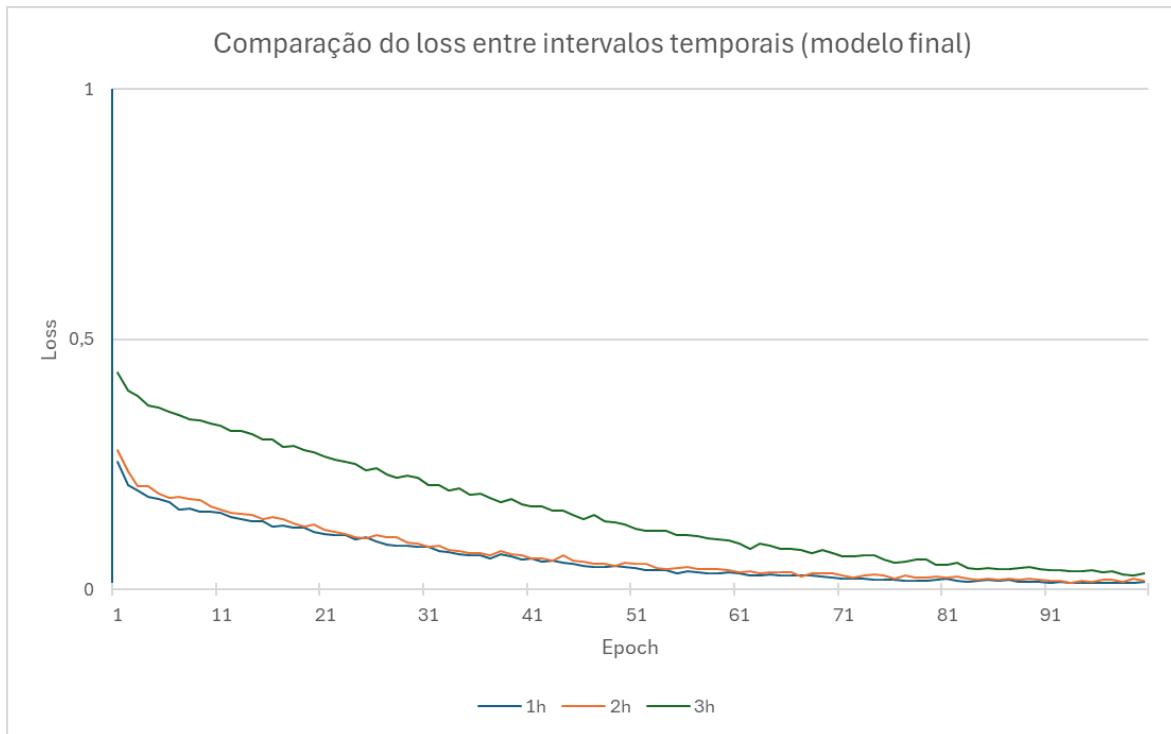


Figura 59 - Gráfico comparativo da loss no modelo final

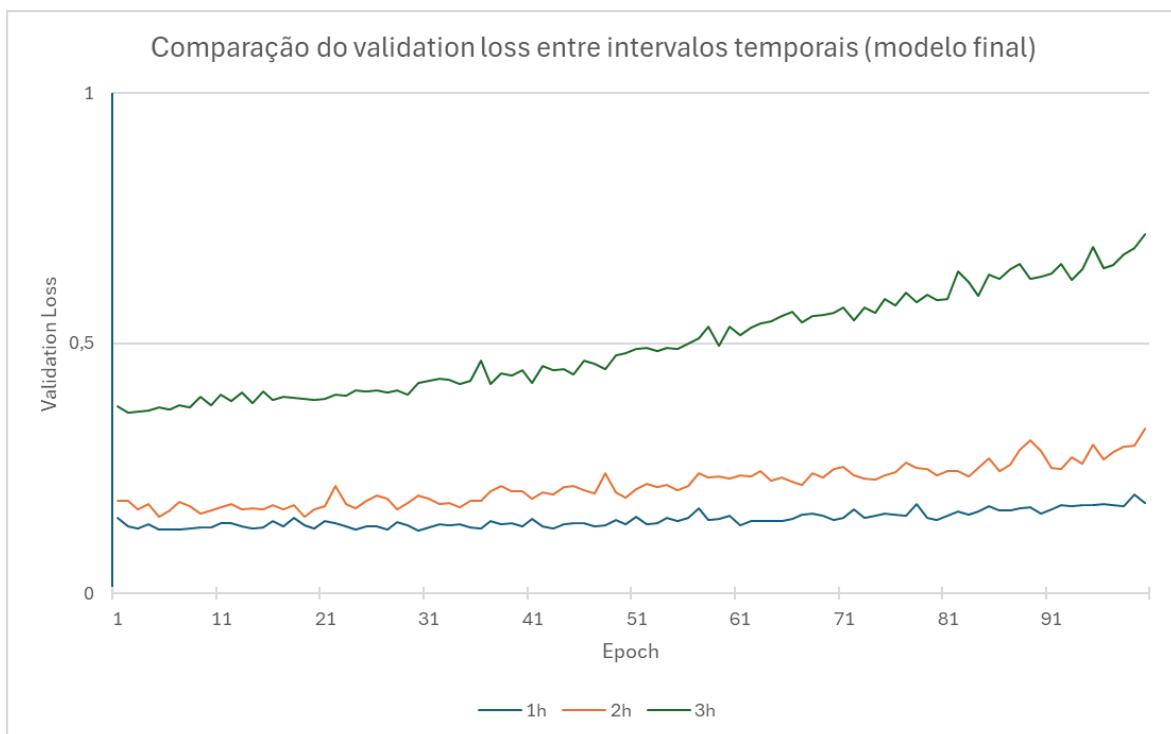


Figura 58 - Gráfico comparativo da validation loss no modelo final

Os resultados obtidos reforçam a eficácia do modelo, confirmando a sua capacidade de generalização mesmo com diferentes intervalos temporais entre a imagem e o valor de precipitação:

- Para 1 hora de antecedência, o modelo atinge uma *validation_accuracy* de 97.4%, o que o torna ideal para aplicações práticas;
- A previsão com 2 horas de antecedência mantém também resultados excelentes (96.78% de *validation_accuracy*), com apenas uma ligeira degradação em relação ao intervalo anterior;
- No caso mais desafiante, com 3 horas de antecedência, embora se observe uma descida da *validation_accuracy* para 88.75%, os resultados continuam sólidos. Este valor indica que mesmo com uma janela de previsão mais alargada, o modelo continua a captar padrões relevantes.

Estes resultados validam a viabilidade da rede híbrida desenvolvida para utilização real, nomeadamente no contexto do website interativo, onde previsões com 1h, 2h ou até 3h de antecedência podem ser disponibilizadas de forma eficaz.

6. Conclusão

O presente projeto teve como principal objetivo desenvolver um sistema capaz de prever a ocorrência de precipitação em Portugal Continental com base em imagens de radar meteorológico, recorrendo a técnicas avançadas de *Deep Learning*, com especial foco nas *Liquid Neural Networks* (LNN). Para além disso, procurou-se comparar o desempenho destas redes com o modelo desenvolvido no ano letivo anterior, baseado em *Convolutional Neural Networks* (CNN), e explorar a integração das capacidades de extração de características espaciais das CNN com a forte aptidão das LNN para análise de dados temporais.

Consideramos que os objetivos propostos foram alcançados. O projeto permitiu demonstrar o potencial das LNN na tarefa de previsão temporal, embora os modelos finais tenham ficado ligeiramente aquém das expectativas, sobretudo devido à quantidade limitada de amostras recolhidas este ano – um problema particularmente evidente em situações de precipitação intensa.

Apesar destas limitações, foi possível explorar diversas particularidades das LNN e desenvolver um modelo com robustez suficiente para sustentar uma aplicação prática funcional. Esta aplicação permite o acesso, em tempo real, a previsões de precipitação próximas da realidade, demonstrando o valor aplicado do trabalho realizado.

Para implementações futuras, será essencial ultrapassar a limitação dos dados, o que poderá passar por parcerias com o Instituto Português do Mar e da Atmosfera (IPMA) ou outras entidades meteorológicas. Adicionalmente, recomenda-se a inclusão de diferentes tipos de dados de entrada, como imagens de infravermelhos, valores de temperatura e níveis de humidade, de forma a enriquecer o contexto disponível para o modelo e facilitar o reconhecimento de padrões meteorológicos mais complexos.

Em suma, este projeto contribuiu significativamente para a compreensão e exploração das *Liquid Neural Networks*, uma tecnologia ainda recente, mas com enorme potencial no campo do *Deep Learning*, e culminou no desenvolvimento de uma aplicação funcional, evidenciando o valor prático da inteligência artificial no domínio da meteorologia.

Bibliografia

- | H. Hewamalage, C. Bergmeir e K. Bandara, “Recurrent Neural Networks for Time
- 1] Series Forecasting: Current status and future directions,” 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0169207020300996>. [Acedido em 11 Junho 2025].
- | G. Boesch, “What are Liquid Neural Networks?,” 17 Maio 2024. [Online].
- 2] Available: <https://viso.ai/deep-learning/what-are-liquid-neural-networks/>. [Acedido em 11 Junho 2025].
- | M. Lechner, R. M. Hasani e R. Grosu, “Neuronal Circuit Policies,” 22 Março
- 3] 2018. [Online]. Available: <https://arxiv.org/abs/1803.08554>. [Acedido em 11 Junho 2025].
- | Instituto Português do Mar e da Atmosfera, “Radar - Mapa dinâmico Continente,”
- 4] 2025. [Online]. Available: <https://www.ipma.pt/pt/otempo/obs.remote/>. [Acedido em 11 Junho 2025].
- | J. D. S. Pereira, “Deterministic ANN PS method for initializing the weight of
- 5] neurons from an artificial neural network with perfect sequences”. Patente pendente (2024).
- | S. Walczak e N. Cerpa, “Artificial Neural Networks,” 2003. [Online]. Available: <https://www.sciencedirect.com/topics/earth-and-planetary-sciences/artificial-neural-network>. [Acedido em 11 Junho 2025].
- | M. Weaver, “Perceptron 101: The building blocks of a neural network,” 21 Julho
- 7] 2023. [Online]. Available: <https://pub.aimind.so/perceptron-101-the-building-blocks-of-a-neural-network-496f6b9b3826>. [Acedido em 11 Junho 2025].
- | Geeks for Geeks, “Backpropagation in Neural Network,” 25 Maio 2025. [Online].
- 8] Available: <https://www.geeksforgeeks.org/machine-learning/backpropagation-in-neural-network/>. [Acedido em 11 Junho 2025].

- | G. Paaß, “Deep Learning: How do deep neural networks work?,” 21 Abril 2021.
- 9] [Online]. Available: <https://lamarr-institute.org/blog/deep-neural-networks/>. [Acedido em 11 Junho 2025].
- | Amanatullah, “Vanishing Gradient Problem in Deep Learning: Understanding,
- 10] Intuition, and Solutions,” 12 Junho 2023. [Online]. Available: <https://medium.com/@amanatulla1606/vanishing-gradient-problem-in-deep-learning-understanding-intuition-and-solutions-da90ef4ecb54>. [Acedido em 11 Junho 2025].
- | P. Raghav, “Understanding of Convolutional Neural Network (CNN) — Deep
- 11] Learning,” 4 Março 2018. [Online]. Available: <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neuralnetwork-cnn-deep-learning-99760835f148>. [Acedido em 11 Junho 2025].
- | Ø. Kolås, “Digital image representation,” 2007. [Online]. Available: https://pippin.gimp.org/image-processing/chap_dir.html. [Acedido em 11 Junho 2025].
- | S. Saha, “A Guide to Convolutional Neural Networks — the ELI5 way,” 20
- 13] Novembro 2023. [Online]. Available: <https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/>. [Acedido em 11 Junho 2025].
- | K. O’Shea e R. Nash, “An Introduction to Convolutional Neural Networks,” 2
- 14] Dezembro 2015. [Online]. Available: <https://arxiv.org/abs/1511.08458>. [Acedido em 11 Junho 2025].
- | A. H. Reynolds, “Convolutional Neural Networks (CNNs),” 2019. [Online].
- 15] Available: <https://anhreynolds.com/blogs/cnn.html>. [Acedido em 11 Junho 2025].
- | H. Yingge, I. Ali e K.-Y. Lee, “Deep Neural Networks on Chip - A Survey,” 2020.
- 16] [Online]. Available: https://www.researchgate.net/publication/340812216_Deep_Neural_Networks_on_Chip_-_A_Survey. [Acedido em 11 Junho 2025].

- | R. Qayyum, "Introduction To Pooling Layers In CNN," 16 Agosto 2022. [Online].
- 17] Available: <https://towardsai.net/p/l/introduction-to-pooling-layers-in-cnn>. [Acedido em 11 Junho 2025].
- | K. Doshi, "Batch Norm Explained Visually – How it works, and why neural networks need it," 18 Maio 2021. [Online]. Available: <https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739/>. [Acedido em 11 Junho 2025].
- | H. Yadav, "Dropout in Neural Networks," 5 Julho 2022. [Online]. Available: <https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9/>. [Acedido em 11 Junho 2025].
- | SuperDataScience, "Convolutional Neural Networks (CNN): Step 3 - Flattening," 20] 18 Agosto 2018. [Online]. Available: <https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening>. [Acedido em 11 Junho 2025].
- | P. Srivatsavaya, "Flatten Layer — Implementation, Advantage and Disadvantages," 4 Outubro 2023. [Online]. Available: <https://medium.com/@prudhviraju.srivatsavaya/flatten-layer-implementation-advantage-and-disadvantages-0f8c4ecf5ac5>. [Acedido em 11 Junho 2025].
- | D. Unzueta e B. Whitfield, "Fully Connected Layer vs. Convolutional Layer: Explained," 24 Fevereiro 2025. [Online]. Available: <https://builtin.com/machine-learning/fully-connected-layer>. [Acedido em 11 Junho 2025].
- | M. Verma, "Binary Classification Using Convolution Neural Network (CNN) Model," 8 Maio 2022. [Online]. Available: <https://medium.com/@mayankverma05032001/binary-classification-using-convolution-neural-network-cnn-model-6e35cdf5bdbb>. [Acedido em 11 Junho 2025].
- | S. K. R. Meruva, V. G. S. Tulasi, N. Vinnakota e B. V, "Risk Level Prediction of Diabetic Retinopathy based on Retinal Images using Deep Learning Algorithm," 2022. [Online]. Available: https://www.researchgate.net/publication/366722281_Risk_Level_Prediction_of_Diabetic_Retinopathy

- betic_Retinopathy_based_on_Retinal_Images_using_Deep_Learning_Algorithm.
[Acedido em 11 Junho 2025].
- | R. T. Q. Chen, Y. Rubanova, J. Bettencourt e D. Duvenaud, “Neural Ordinary
25] Differential Equations,” 14 Dezembro 2019. [Online]. Available:
<https://arxiv.org/abs/1806.07366>. [Acedido em 11 Junho 2025].
- | M. Lechner e R. Hasani, “Neural Circuit Policies (for PyTorch and TensorFlow),”
26] 2021. [Online]. Available: <https://github.com/mlech261/ncps>. [Acedido em 11 Junho
2025].
- | R. Hasani, M. Lechner, A. Amini, D. Rus e R. Grosu, “Liquid Time-constant
27] Networks,” 14 Dezembro 2020. [Online]. Available: <https://arxiv.org/abs/2006.04439>.
[Acedido em 11 Junho 2025].
- | R. Hasani, M. Lechner, A. Amini, L. Liebenwein, A. Ray, M. Tschaikowski, G.
28] Teschl e D. Rus, “Closed-form Continuous-time Neural Models,” 2 Março 2022.
[Online]. Available: <https://arxiv.org/abs/2106.13898>. [Acedido em 11 Junho 2025].
- | A. Rosales, “Liquid Neuronal Networks using Pytorch,” 22 Maio 2024. [Online].
29] Available: <https://medium.com/@andrea.rosales08/liquid-neuronal-networks-using-pytorch-0d0bef41d504>. [Acedido em 11 Junho 2025].
- | A. A. Awan, “A Complete Guide to Data Augmentation,” 9 Dezembro 2024.
30] [Online]. Available: <https://www.datacamp.com/tutorial/complete-guide-data-augmentation>. [Acedido em 11 Junho 2025].
- | K. Studios, “Image Augmentation for Computer Vision Tasks Using PyTorch,” 22
31] Setembro 2023. [Online]. Available: <https://www.comet.com/site/blog/image-augmentation-for-computer-vision-tasks-using-pytorch/>. [Acedido em 11 Junho
2025].
- | J. Murel Ph.D e E. Kavlakoglu, “What is data augmentation?,” 7 Maio 2024.
32] [Online]. Available: <https://www.ibm.com/think/topics/data-augmentation>. [Acedido em 11 Junho 2025].

- | S. Galli, “Exploring Oversampling Techniques for Imbalanced Datasets,” 20
33] Março 2023. [Online]. Available: <https://www.blog.trainindata.com/oversampling-techniques-for-imbalanced-data/>. [Acedido em 11 Junho 2025].
- | C. Ellis, “Oversampling vs undersampling for machine learning,” 21 Janeiro 2023.
34] [Online]. Available: <https://crunchingthedata.com/oversampling-vs-undersampling/>.
[Acedido em 11 Junho 2025].
- | T. Mucci, “Overfitting vs. underfitting: Finding the balance,” 11 Dezembro 2024.
35] [Online]. Available: <https://www.ibm.com/think/topics/overfitting-vs-underfitting>.
[Acedido em 11 Junho 2025].
- | Geeks for Geeks, “ML | Underfitting and Overfitting,” 27 Janeiro 2025. [Online].
36] Available: <https://www.geeksforgeeks.org/machine-learning/underfitting-and-overfitting-in-machine-learning/>. [Acedido em 11 Junho 2025].
- | “Jupyter Notebook,” LF Charities, [Online]. Available: <https://jupyter.org/>.
37]
- | “Flask,” Pallets, [Online]. Available: <https://flask.palletsprojects.com/en/stable/>.
38]
- | “Vue.js,” Evan You, [Online]. Available: <https://vuejs.org/>.
39]
- | “TensorFlow,” Google, [Online]. Available: <https://www.tensorflow.org/>.
40]
- | “Keras,” Google, [Online]. Available: <https://keras.io/>.
41]
- | “GitHub,” GitHub, Inc., [Online]. Available: <https://github.com/>.
42]
- | “Microsoft Excel,” Microsoft, [Online]. Available:
43] <https://www.microsoft.com/pt-pt/microsoft-365/excel>.

- | Instituto Português do Mar e da Atmosfera, “Critérios de Emissão dos Avisos Meteorológicos,” 2025. [Online]. Available: <https://www.ipma.pt/pt/encyclopedia/otempo/sam/index.html?page=criterios.xml>. [Acedido em 11 Junho 2025].
- | Instituto Português do Mar e da Atmosfera, 2025. [Online]. Available: https://www.ipma.pt/pt/educativa/faq/meteorologia/previsao/faqdetail.html?f=/pt/educativa/faq/meteorologia/previsao/faq_0033.html. [Acedido em 11 Junho 2025].
- | E. F. d. S. Mendes e J. R. F. Oliveira, “Site Web com Inteligência Artificial,” 2023.
- | R. d. S. Crespo e M. M. Melro, “Site Web Meteo com Deep Learning,” Instituto Politécnico de Leiria, Leiria, 2024.