

***Site Web Meteo* com Inteligência Artificial**

Licenciatura em Engenharia Informática

Edgar Filipe da Silva Mendes

João Rafael Freitas Oliveira

Leiria, julho de 2023



***Site Web Meteo* com Inteligência Artificial**

Licenciatura em Engenharia Informática

Edgar Filipe da Silva Mendes

João Rafael Freitas Oliveira

Trabalho de Projeto da unidade curricular de Projeto Informático realizado sob a orientação do Professor João da Silva Pereira e do Professor Rui Vasco Guerra Baptista Monteiro

Leiria, julho de 2023

# Agradecimentos

Gostaríamos de expressar a nossa mais sincera gratidão a ambos os nossos orientadores, Professor Doutor João da Silva Pereira e Professor Doutor Rui Vasco Monteiro, pelo apoio inestimável que nos proporcionaram ao longo da elaboração do nosso projeto informático. Sem a orientação e o conhecimento dos mesmos, certamente não teria sido possível alcançar esse marco significativo.

Agradecemos também aos nossos amigos e família pelo apoio dado ao longo da realização deste trabalho.

# Resumo

Todos reconhecemos que a meteorologia e as condições climáticas têm uma influência significativa em todas as atividades humanas, quer sejam realizadas em espaços interiores ou ao ar livre. Além de influenciar as decisões mundanas de alguém antes de sair de casa, a previsão de precipitação é crucial em áreas como aviação, navegação, agricultura, indústria, comércio e turismo.

Em Portugal, o Instituto Português do Mar e da Atmosfera (IPMA), uma organização governamental, analisa e faz a previsão do estado do tempo para períodos longos até 10 dias e com um certo grau de incerteza. Por esse motivo, o objetivo deste projeto é alcançar uma alta taxa de acerto na identificação de padrões de precipitação para que o algoritmo seja confiável em contextos reais.

Para atingir esse objetivo, foram utilizados os dados do IPMA que procedeu à implementação de captura de imagens do estado meteorológico, imagens de radares que serão utilizadas para prever a precipitação em Leiria. Neste projeto serão testadas técnicas de inteligência artificial com redes neuronais artificiais em *Deep Learning*.

Ao chegar ao fim do trabalho, foi observado uma certa falha por parte da rede neuronal na previsão de momentos com chuva intensa, devido à falta de dados deste tipo para o treino do mesmo. É ainda considerado que, caso seja possível obter mais dados do IPMA, quer sejam fornecidos pelo próprio, quer sejam obtidos pelo *developer* em períodos com chuva frequente e intensa, o algoritmo atingiria uma boa taxa de precisão.

**Palavras-chave:** *Deep Learning*, CNN, Meteorologia, IPMA, Redes Neuronais Artificiais

# Abstract

We all recognize that meteorology and weather conditions have a significant influence on all human activities, whether indoors or outdoors. In addition to influencing someone's everyday decisions before leaving home, precipitation forecasting is crucial in areas such as aviation, navigation, agriculture, industry, trade, and tourism.

In Portugal, the Portuguese Institute for the Sea and Atmosphere, a governmental organization, analyses and predicts the weather conditions for longer periods of up to 10 days with a certain degree of uncertainty. For this reason, the objective of this project is to achieve a high accuracy rate in identifying precipitation patterns so that the algorithm can be reliable in real-world contexts.

To achieve this goal, data from IPMA was used, which implemented radars to capture images of the weather conditions. These images will be used to predict precipitation in Leiria. In this project, artificial neural networks in deep learning will be tested using artificial intelligence techniques.

Upon reaching the end of the project, it was observed that the neural network had a certain failure rate in predicting moments of heavy rainfall due to the lack of data of this type for its training. It is also considered that if it is possible to obtain more data from IPMA, whether provided by the organization itself or obtained by the developer during periods of frequent and intense rainfall, the algorithm would achieve a good accuracy rate.

**Keywords:** Deep Learning, CNN, Meteorology, IPMA, Artificial Neural Networks

Índice

[Agradecimentos iii](#_Toc139902923)

[Resumo iv](#_Toc139902924)

[Abstract v](#_Toc139902925)

[Lista de Figuras viii](#_Toc139902926)

[Lista de tabelas x](#_Toc139902927)

[Lista de siglas e acrónimos xi](#_Toc139902928)

[1. Introdução 1](#_Toc139902929)

[2. Enquadramento teórico 3](#_Toc139902930)

[2.1. *Deep* *Learning* 3](#_Toc139902931)

[2.1.1. O neurónio 4](#_Toc139902932)

[2.1.2. A rede neuronal 5](#_Toc139902933)

[2.1.3. Aplicações 6](#_Toc139902934)

[2.2. *Convolutional* *Neural* *Network* 8](#_Toc139902935)

[*2.2.1.* *Convolution* *Layer* 8](#_Toc139902936)

[*2.2.2.* *Pooling Layer* 9](#_Toc139902937)

[*2.2.3.* *Fully* *connected* *layer* 10](#_Toc139902938)

[3. Descrição do processo 11](#_Toc139902939)

[3.1. Ferramentas utilizadas 11](#_Toc139902940)

[3.1.1. *GitHub* *Desktop* 11](#_Toc139902941)

[3.1.2. *Jupyter* *Notebook* 12](#_Toc139902942)

[*3.1.3.* *Visual Studio* *Code* 12](#_Toc139902943)

[3.2. *Dataset* utilizado 13](#_Toc139902944)

[3.2.1. *Data* *augmentation* 14](#_Toc139902945)

[3.2.2. Normalização dos dados 17](#_Toc139902946)

[3.3. Construção da rede neuronal 18](#_Toc139902947)

[3.3.1. Arquitetura da rede neuronal 18](#_Toc139902948)

[3.3.2. Parâmetros extra 21](#_Toc139902949)

[3.4. Treino do modelo 21](#_Toc139902950)

[3.4.1. Primeira abordagem 21](#_Toc139902951)

[3.4.2. Segunda abordagem 22](#_Toc139902952)

[3.4.3. Terceira abordagem 23](#_Toc139902953)

[3.4.4. Abordagem final 24](#_Toc139902954)

[4. *Website* desenvolvido para aplicar o modelo 26](#_Toc139902955)

[4.1. API em *Flask* 26](#_Toc139902956)

[4.2. *Web* *Meteo* 29](#_Toc139902957)

[4.2.1. Implementação 29](#_Toc139902958)

[4.3. Instalação e uso 38](#_Toc139902959)

[4.3.1. Requisitos 38](#_Toc139902960)

[4.3.2. Passos 38](#_Toc139902961)

[5. Análise de resultados 39](#_Toc139902962)

[6. Conclusão 41](#_Toc139902963)

[Bibliografia ou Referências Bibliográficas 42](#_Toc139902964)

# Lista de Figuras

[Figura 1 – Rede neuronal com quatro camadas. 3](#_Toc139644113)

[Figura 2 - O neurónio artificial. 4](#_Toc139644114)

[Figura 3 - Representação de uma imagem como matriz de *pixels.* 8](#_Toc139644115)

[Figura 4 - *Convolutional Layer.* 9](#_Toc139644116)

[Figura 5 - *Pooling Layer types.* 9](#_Toc139644117)

[Figura 6 - *GitHub Desktop.* 11](#_Toc139644118)

[Figura 7 - *Jupyter Notebook.* 12](#_Toc139644119)

[Figura 8 - Código para obter imagem para o *dataset.* 13](#_Toc139644120)

[Figura 9 - Função para remover *pixels* pretos das imagens. 14](#_Toc139644121)

[Figura 10 - Código da função para deslocar a imagem para baixo. 15](#_Toc139644122)

[Figura 11 - Código da função para deslocar a imagem para a esquerda. 16](#_Toc139644123)

[Figura 12 - Código da função para deslocar a imagem para a direita. 16](#_Toc139644124)

[Figura 13 - Código da função para rodar a imagem. 17](#_Toc139644125)

[Figura 14 - Código da função de normalização do valor de precipitação. 18](#_Toc139644126)

[Figura 15 - Código da função de redimensionamento da imagem. 21](#_Toc139644127)

[Figura 16 – Código da função *get\_radar\_image*(). 25](file:///C:\Users\joao\Desktop\WebMeteoPI\Relatório\Relatorio%20by%20Edgar%20e%20Joao.docx#_Toc139644128)

[Figura 17 - Código do endpoint *process\_image*(). 26](#_Toc139644129)

[Figura 18 - *Website* *Web Meteo.* 27](#_Toc139644130)

[Figura 19 - Código do *template* do *app.vue*. 28](#_Toc139644131)

[Figura 20 - Código do *script* do *app.vue*. 29](#_Toc139644132)

[Figura 21 - Código do *template* do *Clock.vue*. 29](#_Toc139644133)

[Figura 22 - Código do *script* do *Clock.vue*. 30](#_Toc139644134)

[Figura 23 - Código do *template* do *ImagePrediction.vue*. 31](#_Toc139644135)

[Figura 24 - Código do *script* do *ImagePredictions.vue*. 32](#_Toc139644136)

[Figura 25 - Código do *template* do Mapa.vue. 33](#_Toc139644137)

[Figura 26 - Código do *script* do Mapa.vue. 34](#_Toc139644138)

[Figura 27 - Código css do Mapa.vue. 35](#_Toc139644139)

# Lista de tabelas

[Tabela 1 - Total de dados obtidos 18](#_Toc139921441)

[Tabela 2 - Quantidade de dados a ser usada na primeira abordagem. 22](#_Toc139921442)

[Tabela 3 - Resultados da primeira abordagem. 23](#_Toc139921443)

[Tabela 4 - Quantidade de dados a ser usada na segunda abordagem. 23](#_Toc139921444)

[Tabela 5 - Resultados da segunda abordagem. 24](#_Toc139921445)

[Tabela 6 - Quantidade de dados a ser usada na terceira abordagem. 24](#_Toc139921446)

[Tabela 7 - Resultados da terceira abordagem. 25](#_Toc139921447)

[Tabela 8 - Resultados da abordagem final para uma hora de diferença entre a imagem e a label 25](#_Toc139921448)

[Tabela 9 - Resultados da abordagem final para duas horas de diferença entre a imagem e a label 25](#_Toc139921449)

[Tabela 10 - Resultados da abordagem final para três horas de diferença entre a imagem e a label 26](#_Toc139921450)

# Lista de siglas e acrónimos

|  |  |
| --- | --- |
| API | *Application Programming Interface* |
| CNN | *Convolutional Neural Network* |
| DOM | *Document Object Model* |
| HTTP | *Hypertext Transfer Protocol* |
| IDE | *Integrated Development Environment* |
| IPMA | Instituto Português do Mar e da Atmosfera |
| JSON | *JavaScript Object Notation* |
| ML | *Machine Learning* |
| ReLU | *Rectified Linear Unit* |
| URL | *Uniform Resource Locator* |

# Introdução

A previsão do tempo é uma área multidisciplinar que combina observações atmosféricas, técnicas de modelagem e processamento de dados para estimar as condições climáticas futuras. Ela envolve a obtenção de informações de várias fontes, como satélites, radares, estações terrestres, entre outros, para entender o estado atual da atmosfera e fazer projeções sobre como ela irá evoluir nas próximas horas, dias ou até mesmo semanas.

Os dados recolhidos incluem informações como intensidade do vento, temperatura, humidade, pressão atmosférica, vento, precipitação e padrões climatéricos. Essas informações são alimentadas em modelos computacionais que utilizam equações matemáticas complexas para simular o comportamento da atmosfera. Com base nessas simulações, as previsões do tempo são geradas e refinadas à medida que mais dados são incorporados ao longo do tempo.

No contexto deste projeto, o foco está na previsão de curto prazo, com base nos dados dos radares meteorológicos do IPMA e técnicas avançadas de processamento de imagens. As redes neuronais convolucionais são um tipo de algoritmo de *machine learning* (ML) que se mostram eficazes no reconhecimento de padrões em imagens. No caso específico, a *Convolutional Neural Network* (CNN) é treinada para identificar o nível de precipitação numa escala de milímetros por hora com base nas imagens de radar.

A abordagem de usar tecnologias de ML e processamento de imagens visa melhorar a precisão das previsões meteorológicas. Com previsões mais confiáveis e detalhadas sobre a precipitação, é possível tomar decisões mais informadas em diversas áreas. Por exemplo, na agricultura, os agricultores podem ajustar os seus planos de irrigação, e relativamente aos meios de transporte, as pessoas podem planear melhor as suas rotas, escolher o meio de transporte adequado e preparar-se para possíveis condições adversas.

Além disso, as previsões meteorológicas precisas também têm um impacto direto na segurança, permitindo que as autoridades emitam alertas antecipados de tempestades, inundações ou outros eventos climatéricos extremos. Isso dá às comunidades tempo suficiente para se prepararem e tomarem as devidas medidas preventivas, reduzindo assim os danos e riscos associados a esses eventos.

No geral, o uso destas tecnologias avançadas, combinadas com dados meteorológicos precisos, tem o potencial de fornecer previsões meteorológicas mais confiáveis e detalhadas, beneficiando tanto a sociedade em geral, quanto diversos setores económicos, que dependem dessas informações para tomar decisões estratégicas.

# Enquadramento teórico

## *Deep* *Learning*

*Deep Learning* é uma técnica poderosa no campo da inteligência artificial que visa replicar o funcionamento do cérebro humano. Para explicar o *Deep Learning* de forma simples, envolve a construção de redes neuronais com três ou mais camadas (Figura 1). Essas redes neuronais são projetadas para analisar grandes quantidades de dados e “aprender” com eles.

A picture containing diagram, line, circle, screenshot

Description automatically generated

Figura – *Rede neuronal com quatro camadas.*

O objetivo principal do *Deep Learning* é permitir que as máquinas façam previsões precisas e realizem tarefas sem a intervenção humana. Ao imitar o comportamento do cérebro humano, os algoritmos de *Deep Learning* podem processar e interpretar padrões e características complexas nos dados, permitindo-lhes identificar e classificar informações com uma precisão cada vez maior.

A implementação da tecnologia de *Deep Learning* tem levado a avanços significativos em várias aplicações e serviços. Produtos e serviços do dia a dia, como assistentes virtuais, comandos de TV por voz e sistemas de deteção de fraude em cartões de crédito, dependem de algoritmos de *Deep Learning*. Além disso, tecnologias emergentes, como carros automáticos, utilizam amplamente o *Deep Learning* para a sua operação.

Ao aproveitar o potencial do *Deep Learning*, as máquinas tornam-se capazes de realizar tarefas analíticas que anteriormente exigiam intervenção humana. Esta tecnologia tem revolucionado a automação, abrindo o caminho para sistemas mais inteligentes e eficientes em diferentes setores.

### O neurónio

Os neurónios artificiais são inspirados pela biologia e tentam replicar o funcionamento do nosso cérebro. O nosso cérebro possui mais de cem mil milhões de células neuronais que nos ajudam a interpretar os chamados sinais ou sinapses.

Pode-se imaginar um sinal como qualquer coisa que varia desde um raio de luz que atinge os nossos olhos até um pensamento nosso de que queremos mover o braço. Os neurónios ajudam-nos a interpretar esses sinais de forma correta. Por exemplo, quando um raio de luz atinge os nossos olhos, um neurónio saberá que esse raio de luz pode ser interpretado como azul, verde ou amarelo.

Agora, neurónios artificiais são uma abordagem para transferir esse princípio para um computador, replicando um neurónio biológico em forma de código. (Figura 2)

A picture containing black, darkness

Description automatically generated

Figura - O neurónio artificial.

Para replicar este processo surgiu um novo conceito, os pesos. Um neurónio artificial começa por receber um input, de seguida atribui um peso a esse input consoante a sua importância e multiplica o input pelo peso gerado, criando assim um novo valor.

No contexto de um neurónio artificial, o *bias* refere-se a um parâmetro adicional que é incorporado no cálculo da saída do neurónio. Pode ser considerado como um valor constante que é adicionado à soma ponderada das entradas antes de passar por uma função de ativação.

O termo *bias* permite ao neurónio artificial ter influência na sua saída, mesmo quando todos os valores de entrada são zero. Ele fornece ao neurónio a capacidade de deslocar o limiar da função de ativação e afetar a resposta geral do neurónio. Ajustando o *bias*, o neurónio pode se tornar mais ou menos propenso a ativar, mesmo na ausência de sinais de entrada fortes. O peso do *bias* é ajustado durante o processo de treino juntamente com os outros pesos das conexões para otimizar o desempenho do neurónio artificial.

De seguida, a soma do peso multiplicado pelo input e do *bias* passa por uma função de ativação.

Uma função de ativação em uma rede neuronal é responsável por determinar a saída ou ativação de um neurónio artificial com base na entrada recebida. Ela define se o neurónio deve ser ativado ou não, influenciando se a informação será transmitida para os neurónios subsequentes na rede.

Em geral, uma função de ativação compará a entrada recebida com um limite ou *threshold*. Se a entrada exceder esse limite, o neurónio é ativado e sua saída é propagada para os neurónios seguintes. Caso contrário, se a entrada for menor ou igual ao limite, o neurônio permanece inativo e não transmite informações.

O *threshold* determina o nível mínimo necessário para que o neurónio dispare, ou seja, para que ele seja ativado. É uma medida de sensibilidade do neurônio em relação aos estímulos que recebe. O *threshold* pode variar dependendo da função de ativação escolhida e das necessidades do problema em questão.

A função de ativação pode assumir diferentes formas e características e cada uma delas possui propriedades distintas que influenciam o comportamento do neurónio e a capacidade da rede neuronal de aprender e representar relações complexas.

Portanto, a função de ativação desempenha um papel crucial na tomada de decisão dos neurónios artificiais, permitindo que a rede neuronal processe informações, aprenda padrões e faça previsões com base nos dados de entrada.

### A rede neuronal

Uma rede neuronal é composta por um conjunto de camadas interligadas. Cada camada é formada por neurónios que recebem parâmetros de entrada e passam por um processo de atribuição de pesos e aplicação de uma função de ativação.

O processo começa na camada de entrada, onde os neurónios recebem os valores dos parâmetros de entrada. Em seguida, cada neurónio dessa camada passa o seu valor para todos os neurónios da próxima camada, acontecendo dessa forma a *forward propagation*.

À medida que os valores dos neurónios são propagados pela rede, eles passam pelo processo de atribuição de pesos e função de ativação em cada camada subsequente. Os pesos determinam a importância relativa dos valores de entrada para cada neurónio, enquanto a função de ativação determina o valor de saída do neurónio com base nos valores ponderados de entrada.

Esse processo continua até que os valores atinjam a camada final da rede. Nessa camada, os neurónios aplicam o mesmo processo descrito anteriormente, atribuindo pesos aos valores de entrada e aplicando uma função de ativação. Os resultados obtidos são utilizados como as saídas da rede neuronal.

Dependendo da tarefa em questão, pode-se utilizar uma função de ativação específica na camada de saída. Por exemplo, em problemas de classificação binária, é comum utilizar a função *sigmoid*, que produz uma saída entre 0 e 1, representando a probabilidade de pertencer a uma classe específica. Outra função de ativação usada é a *softmax*, na qual o *output* de cada neurónio da camada final representa a probabilidade de o *input* pertencer à classe associada a esse neurónio.

Em resumo, uma rede neuronal é uma estrutura composta por camadas interligadas de neurónios. Cada neurónio realiza o processo de atribuição de pesos e função de ativação, passando os valores para os neurónios da próxima camada. Esse processo repete-se até a camada final, onde são geradas as saídas da rede neural. A escolha da função de ativação adequada para a tarefa em questão é fundamental para obter resultados precisos e adequados ao problema.

### Aplicações

As aplicações em *Deep Learning* estão presentes no nosso quotidiano, embora, na maioria dos casos, estão tão bem integradas nos produtos e serviços que os utilizadores não têm conhecimento do complexo processamento de dados que ocorre.

Alguns exemplos destas aplicações são:

* Aplicação na área de segurança pública:

Os algoritmos *Deep Learning* podem analisar e aprender com dados transacionais para identificar padrões perigosos que indiquem possíveis atividades fraudulentas ou criminosas. Aplicações de reconhecimento de voz, visão computacional e outras áreas do *Deep Learning* podem melhorar a eficiência e eficácia da análise investigativa, extraindo padrões e evidências de gravações de som e vídeo, imagens e documentos. Isto ajuda as forças de segurança a analisar grandes quantidades de dados de forma mais rápida e precisa.

* Aplicação na área de serviços financeiros:

As instituições financeiras utilizam regularmente análises preditivas para impulsionar a negociação algorítmica de ações, avaliar riscos empresariais para a aprovação de empréstimos, detetar fraudes e auxiliar na gestão do crédito e carteiras de investimento para os clientes.

* Aplicação na área de atendimento ao cliente:

Muitas organizações incorporam tecnologia com *Deep Learning* nos seus processos de atendimento ao cliente. Os *chatbots*, amplamente utilizados em várias aplicações, serviços e portais de atendimento ao cliente, são uma forma de inteligência artificial. Os *chatbots* tradicionais usam linguagem humana e, até mesmo, reconhecimento visual, geralmente encontrados em centrais de atendimento.

No entanto, soluções de *chatbot* mais sofisticadas tentam determinar, se existem múltiplas respostas para perguntas indeterminadas. Com base nas respostas recebidas, o *chatbot* tenta responder a essas perguntas diretamente ou encaminhar a conversa para um utilizador humano.

Assistentes virtuais, como a *Siri* da *Apple*, *Alexa* da *Amazon* ou o *ChatGPT* da *OpenAI*, ampliam a ideia de um *chatbot*, permitindo a funcionalidade de reconhecimento de voz. Isto cria um novo método para envolver utilizadores de forma personalizada.

* Aplicação na área da saúde:

A indústria da saúde tem se beneficiado imensamente das capacidades do *Deep Learning* desde a digitalização dos registos e imagens hospitalares. Aplicações de reconhecimento de imagens podem apoiar especialistas em imagiologia médica e radiologistas, ajudando-os a analisar e avaliar mais imagens em menos tempo.

Estas são apenas algumas das aplicações reais que usam *Deep* *Learning* que estão a transformar diferentes setores da sociedade, melhorando a eficiência, precisão e qualidade dos serviços prestados. Com o avanço contínuo da tecnologia, espera-se que o campo do *Deep Learning* continue a evoluir e a encontrar novas formas de melhorar a nossa vida diária.

## *Convolutional* *Neural* *Network*

Uma *Convolutional Neural Network*, também conhecida como CNN ou *ConvNet*, é uma classe de redes neuronais que se especializa no processamento de dados e que possuem uma topologia em forma de grade, como uma imagem.

Uma imagem digital é uma representação binária de dados visuais. Ela contém uma série de pixels dispostos numa estrutura semelhante a uma matriz, onde os valores dos pixels indicam o brilho e a cor de cada *pixel* (Figura 3).



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

As CNN têm vários tipos de camadas específicas que organizadas de uma certa forma têm a capacidade de reconhecer padrões e características simples, como bordas e texturas e até mesmo coisas mais complexas como objetos ou pessoas. De seguida será explicado explicado as camadas mais importantes deste tipo de rede.

### *Convolution* *Layer*

A camada convolucional, ou *Conv2D*, é a base de todas as CNN’s. É boa prática começar uma CNN com uma camada deste tipo, pois reduz os dados significativa e eficientemente, passando para as camadas que a sucedem os dados resumidos, mas pertinentes, diminuindo assim o custo computacional da rede, mas não afetando a sua performance.

O parâmetro mais importante duma camada convolucional é o seu filtro, também conhecido como *kernel*. Este filtro tem o seu tamanho predefinido, sendo normalmente 2x2, 3x3 ou 5x5 e é o responsável por gerar o mapa de características, sendo esse o output desta camada.

Após receber uma matriz de *pixels*, o filtro vai deslizando ao longo da matriz. Para cada posição do *kernel*, os valores dos pixels cobertos pelo mesmo, são multiplicados pelos seus próprios valores. O resultado dessas multiplicações é somado dando assim origem ao novo valor do mapa de características. Ao fazer isto o mapa de características é capaz de detetar os pontos mais relevantes de uma certa imagem (Figura 4).



Figura - *Convolutional Layer.*

O mapa de caraterísticas pode ainda ser subjugado a uma função de ativação para extrair ainda mais os pontos mais importantes de uma certa imagem.

### *Pooling Layer*

A *Pooling layer*, normalmente subsequente à *Convolutional* *Layer*, tem como objetivo reduzir o tamanho do mapa de características, gerando como *output* uma versão reduzida do que recebeu.

Tal como a *Convolutional* *Layer*, a *Pooling* *Layer* também recebe um filtro, mas desta vez conhecido como *Pool* *size*, que passará pelo mapa de caraterísticas. Existindo dois tipos de camadas de *Pooling* o filtro tem um funcionamento diferente em cada uma delas.

No caso da *Max* *Pooling* *Layer*, o filtro seleciona o valor mais alto de dentro dos que estão a ser cobertos por ele e passa para o novo mapa de características.

Já na *Average Pooling Layer*, o filtro faz a média dos valores que estão a ser cobertos por ele, sendo o resultado, o novo valor no mapa de características (Figura 5).



Figura - *Pooling Layer types.*

Esta camada é utilizada especialmente para reduzir o número de parâmetros da rede de forma a otimizar ainda mais o modelo. No entanto também têm alguns efeitos adversos como a perda de informação detalhada.

### *Fully* *connected* *layer*

A *Fully connected layer* não tem nenhum requisito para o parâmetro de entrada e usualmente é utilizada como a última camada de uma CNN. A estrutura da *Fully connected layer* pode variar dependendo do objetivo da rede.

No caso de classificação binária, onde a rede deve determinar se uma imagem pertence a uma determinada classe ou não, a camada consistirá em apenas um neurónio com a função de ativação *sigmoid*. Esse neurônio calculará um valor entre 0 e 1, representando a probabilidade de a imagem pertencer à classe em questão. Um valor próximo de 1 indica que a rede está confiante de que a imagem pertence à classe, enquanto um valor próximo de 0 indica que a rede acredita que a imagem não pertence à classe.

Por outro lado, no caso de classificação para várias classes, onde a rede deve atribuir uma imagem a uma de várias classes possíveis, a camada terá tantos neurónios quanto o número de classes existentes. Cada neurónio estará associado a uma classe específica e calculará a probabilidade de a imagem pertencer a essa classe. Para obter uma distribuição de probabilidade correta, é comum aplicar a função de ativação *softmax* nos valores de saída desses neurónios. A função *softmax* normaliza os valores de saída para que eles somem 1, fornecendo assim uma probabilidade para cada classe.

# Descrição do processo

## Ferramentas utilizadas

Para o desenvolvimento deste projeto de grande magnitude, foram selecionadas ferramentas essenciais que desempenharão um papel fundamental em todo o processo. O uso do *GitHub Desktop*, do *Jupyter Notebook* e do *Visual Studio Code* foi cuidadosamente decidido, levando em consideração sua eficiência, facilidade de uso e recursos avançados.

### *GitHub* *Desktop*

O *GitHub* *Desktop* é uma *graphical user interface* desenvolvida pelo *GitHub* para facilitar o uso do *Git*, um sistema de controlo de versões amplamente utilizado para gerir o código-fonte de projetos de *software*. O *GitHub* *Desktop* fornece uma maneira fácil e visualmente intuitiva (Figura 6) de trabalhar com repositórios *Git* no *GitHub*.

Com o *GitHub*, a equipa terá a capacidade de seguir as alterações feitas no projeto, analisar e discutir propostas, e fornecer *feedback* de forma eficiente. Além disso, o *GitHub* oferece recursos para gestão de problemas e tarefas, permitindo que a equipa mantenha o controlo do progresso do projeto (Figura 6).

A screenshot of a computer

Description automatically generated

Figura - *GitHub Desktop.*

### *Jupyter* *Notebook*

O *Jupyter Notebook* é uma *open-source web application* que permite criar e compartilhar documentos interativos que contêm código, visualizações e texto explicativo. Ele é amplamente utilizado em ciência de dados, pesquisa científica e educação, pois oferece uma maneira conveniente de combinar código executável com elementos narrativos (Figura 7).

A principal característica do *Jupyter Notebook* é o suporte a células, que são unidades independentes onde é possível escrever e executar código, bem como adicionar texto, imagens e gráficos. Isso torna o processo de desenvolvimento iterativo e exploratório, permitindo que seja executado e visualizado os resultados do código em tempo real.

As células podem conter código em várias linguagens de programação e quando são executadas, o código dentro dela é processado e o resultado é exibido imediatamente abaixo da célula. Isso facilita ao testar e a depurar o código, pois é possível ver os resultados intermédios à medida que avança.



Figura - *Jupyter Notebook.*

### *Visual Studio* *Code*

O *Visual Studio Code* é um *Integrated Development Environment* (IDE) criado pela *Microsoft*, que ganhou uma enorme popularidade entre os programadores. Com a sua *user* *interface* amigável e extensibilidade, o *Visual Studio Code* oferece suporte a uma ampla variedade de linguagens de programação. Ele possui recursos essenciais, como realce de sintaxe, formatação de código e depuração, para facilitar o desenvolvimento de aplicações de qualquer escala. Além disso, o *Visual Studio Code* é altamente personalizável por meio de extensões, o que permite aos utilizadores adicionar funcionalidades extras, integração com ferramentas externas e suporte a *frameworks* específicas.

Uma característica notável deste IDE é o seu terminal integrado, que elimina a necessidade de alternar entre o editor e uma janela do terminal separada. Além disso, o IDE oferece recursos avançados de controlo de versões, sendo até possível integrar o *GitHub* dentro do mesmo, permitindo aos desenvolvedores gerir repositórios *Git* diretamente dentro do ambiente de desenvolvimento.

### *Flask*

O *Flask* é uma *framework* *web* leve e flexível desenvolvido em *Python*. Ele também é conhecido como uma *micro-framework*, o que significa que fornece apenas o básico necessário para criar aplicações web, tornando-o fácil de aprender e usar.

O *Flask* possui uma *Application Programming Interface* (API) minimalista e simples, permitindo que seja possível criar aplicações web com menos código e menos complexidade. Ela oferece um *routing* *system* que mapeia *Uniform Resource Locators* (URLs) para funções específicas, determinando como a aplicação deve responder aos seus pedidos. Também é uma *framework* altamente extensível, o que significa que é possível adicionar funcionalidades extras através de extensões ou bibliotecas de terceiros.

Além disso, o *Flask* possui uma documentação abrangente e bem escrita, que fornece exemplos claros e tutoriais passo a passo para ajudar os desenvolvedores a começar e aprofundar seus conhecimentos.

Em resumo, o *Flask* é um *framework* web leve, flexível e fácil de usar, que permite que sejam criadas aplicações web usando a linguagem *Python*. Ele é uma ótima opção para projetos pequenos a médios, onde a simplicidade e a facilidade de aprendizado são prioridades.

### Vue.js

O Vue.js é uma *framework* JavaScript que permite criar interfaces de utilizador de forma simples, eficiente e versátil. Baseia-se em HTML, CSS e JavaScript padrão e oferece um modelo de programação declarativo e baseado em componentes que facilita o desenvolvimento de interfaces, sejam elas simples ou complexas. Além disso, o Vue.js é progressivo, o que significa que pode ser adotado de forma incremental, integrando-se com outras bibliotecas ou *frameworks* conforme a necessidade.

Algumas das vantagens sobre o Vue.js são:

1. **Simplicidade**

O Vue.js é fácil de aprender e usar, pois requer apenas conhecimentos básicos de HTML, CSS e JavaScript. Não é necessário aprender outras linguagens ou ferramentas específicas, como acontece com outras *frameworks* como Angular e React. Além disso, o Vue.js tem uma sintaxe clara e intuitiva, que permite escrever aplicações com poucas linhas de código.

1. **Arquitetura baseada em componentes**

O Vue.js segue uma arquitetura baseada em componentes, que consiste em dividir o código da aplicação em partes independentes e reutilizáveis. Cada componente contém um *template*, a lógica e os estilos relacionados à sua funcionalidade. Os componentes podem ser compostos entre si para formar a aplicação completa. Essa abordagem facilita a organização, a manutenção e o teste do código.

1. **Reatividade**

O Vue.js possui um sistema de reatividade que permite atualizar automaticamente a interface de utilizador quando o estado da aplicação muda. Isto é feito através de um mecanismo de observação e notificação que deteta as mudanças nos dados e as reflete no DOM (*Document Object Model*). Assim, não é necessário manipular manualmente o DOM ou usar bibliotecas externas para isso.

1. **Desempenho**

O Vue.js é uma *framework* leve e rápida, que ocupa cerca de 20KB comprimido. Usa um DOM virtual para renderizar os componentes, o que reduz o número de operações no DOM real e melhora o desempenho da aplicação. Além disso, o Vue.js utiliza técnicas de compilação e otimização que permitem gerar código eficiente e adaptado ao navegador.

1. **Ecossistema**

O Vue.js conta com um ecossistema rico e diverso, que oferece soluções para diversas funcionalidades avançadas, como *routing*, gestão de estado, validação de formulários, renderização no lado do servidor, geração de *websites* estáticos, entre outros. Essas soluções são mantidas pela equipa oficial do Vue.js ou pela comunidade de programadores, que contribui com projetos *open source* e recursos educacionais.

## *Dataset* utilizado

Para o desenvolvimento deste projeto, foi necessário criar um conjunto de dados próprio, devido às restrições de disponibilidade de dados.

As imagens utilizadas foram obtidas (Figura 8) a partir dos radares de precipitação do Instituto Português do Mar e da Atmosfera, sendo disponibilizadas apenas imagens com duração máxima de cerca de um mês.



Figura - Código para obter imagem para o *dataset.*

Já os dados sobre os valores de precipitação das respetivas estações que o IPMA dispõe, são fornecidos os das 3 horas antecedentes à hora certa mais recente. Por exemplo, se for feito um pedido à APIàs 14:00h, serão obtidos os dados das 11:00h, 12:00h e 13:00h de todas as estações.

Ainda assim, dado que ao longo do desenvolvimento do projeto existiram algumas alterações no formato *standard* das imagens fornecidas pelo IPMA, sendo que, inicialmente as imagens vinham com as fronteiras delineadas a preto, mas, após um certo período, essas fronteiras foram removidas, e, eventualmente, voltaram a ser colocadas, nós desenvolvemos uma função capaz de remover os *pixels* pretos de uma imagem fazendo com que o *dataset* construído seja mais uniforme (Figura 9).

A picture containing text, screenshot, font, software

Description automatically generated

Figura - Função para remover *pixels* pretos das imagens.

Dado que as imagens fornecidas pelo IPMA contêm o mapa de Portugal Continental na sua integra e a API fornece dados relativos a várias estações, para obter mais dados para treino, foi feito o recorte do mapa para cada distrito atribuindo a cada imagem o valor de precipitação da estação mais próxima da capital de distrito.

Foi feito um recorte de duzentos por duzentos *pixels* de forma a cobrir o distrito na sua totalidade.

Apesar disto, apenas era possível obter um máximo de 432 elementos para o *dataset* diariamente, então, por parte dos orientadores, foi recomendado serem realizadas técnicas de *data augmentation.*

### *Data* *augmentation*

*Data augmentation* é uma técnica utilizada para aumentar artificialmente o *dataset* de treino, criando cópias modificadas do *dataset* original.

É uma técnica bastante utilizada em problemas de *deep* *learning*, por ser a única solução para um dos maiores problemas que as redes neuronais ainda apresentam, que é a dependência sob dados de treino para atingir o seu bom funcionamento.

Para *datasets* baseados em imagens algumas das técnicas mais famosas são realizar transformações geométricas nas imagens, alterar as cores das mesmas, apagar partes aleatórias, entre outras, mas sendo que estes dados são mais ‘delicados’ e não seria lógico dentro do contexto do problema fazer grandes alterações às imagens foi tida uma abordagem mais suave.

Essas alterações foram as seguintes:

1. Deslocamento da imagem em 20 *pixels* para baixo, copiando a “linha” superior da imagem para as 20 novas linhas criadas (Figura 10);

A picture containing text, screenshot, font

Description automatically generated

Figura - Código da função para deslocar a imagem para baixo.

1. Deslocamento da imagem em 20 *pixels* para a direita, copiando a “linha” da esquerda da imagem para as 20 novas colunas criadas (Figura 11);

A picture containing text, screenshot, font

Description automatically generated

Figura - Código da função para deslocar a imagem para a esquerda.

1. Deslocamento da imagem em 20 *pixels* para a esquerda, copiando a “linha” da direita da imagem para as 20 novas colunas criadas (Figura 12);

A picture containing text, screenshot, font

Description automatically generated

Figura - Código da função para deslocar a imagem para a direita.

1. Rotação da imagem original em 1 grau (Figura 13);

A picture containing text, screenshot, font, software

Description automatically generated

Figura - Código da função para rodar a imagem.

1. Rotação da imagem gerada pelo primeiro método (deslocamento para baixo) em 1 grau;
2. Rotação da imagem gerada pelo segundo método (deslocamento para a direita) em 1 grau.
3. Rotação da imagem gerada pelo terceiro método (deslocamento para a esquerda) em 1 grau.

Estas técnicas de aumento de dados permitem criar variações das imagens originais, ampliando o conjunto de dados disponível para o treino do modelo. Com uma maior diversidade de imagens, espera-se que o modelo seja capaz de aprender padrões mais abrangentes e generalizáveis relacionados à precipitação.

Apesar de todos os métodos empregues, não foi possível obter imagens de chuva intensa devido ao estado meteorológico predominante em Portugal Continental ter sido, maioritariamente, ameno.

### Normalização dos dados

Já para os valores de precipitação, com o objetivo de obter um treino mais estável e preciso, optou-se por normalizar o valor da precipitação com base no maior valor já registado em Portugal pelo IPMA, que foi de 220 milímetros cúbicos por hora, em Penhas da Saúde no dia catorze de janeiro de mil novecentos e setenta e sete. Ainda assim, decidiu-se aumentar esse valor para proporcionar uma margem de segurança para possíveis valores extremos que possam ocorrer.

A fórmula utilizada para essa normalização foi a seguinte:

Como a medida preventiva, optou-se por arredondar o valor obtido. Além disso, caso o valor fosse negativo, substituía-se por zero. Essa medida foi implementada devido a alguns pedidos que continham o valor de “-99.0”, possivelmente, por de erros no lado do IPMA (Figura 14).

A black background with white text

Description automatically generated with low confidence

Figura - Código da função de normalização do valor de precipitação.

Sendo assim, devido à escassez de dados, após a normalização dos dados, apenas foram obtidos dados com os valores de 0, 1, 2, 3 e 5. Os valores finais de precipitação (Tabela 1) obtidos foram os seguintes:

Tabela - Total de dados obtidos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Valor normalizado 0 | Valor normalizado 1 | Valor normalizado 2 | Valor normalizado 3 | Valor normalizado 5 |
| 21592 | 408 | 168 | 16 | 8 |

Estas imagens e valores foram utilizados tanto para o treino quanto para os testes dos modelos construído.

## Construção da rede neuronal

### Arquitetura da rede neuronal

A versão final do modelo é constituída por um total de treze camadas. Primeiramente, para receber o *input*, existe uma camada Convolucional 2D com dezasseis neurónios, um *max pooling size* de (2,2), a *Rectified Linear Unit* (ReLU) como função de ativação e, por fim, um *input* *shape* de 100100 *pixels* com quatros canais, dado que, as imagens fornecidas pelo IPMA, vêm no formato *Red-Green-Blue-Alpha*. De seguida, outra camada Convolucional 2D, também com dezasseis neurónios, um *filter siz*e de 3, o *stride* com tamanho de 1, o *padding* com o valor ‘*same*’ e também a ReLU como função de ativação. Foi decidido começar o modelo com duas camadas Convolucional 2D, pois, ao seguir essa mesma abordagem, o modelo tem capacidade de detetar características mais complexas e abstratas. A primeira camada é responsável por capturar características mais simples como bordas, formas básicas, entre outras, enquanto a segunda camada tem como intuito capturar características mais complexas que se integram nas características da camada precedente, revelando assim, os padrões mais significativos das imagens.

Após estas duas camadas está uma cada de *BatchNormalization*. Este tipo de camada tem como objetivo melhorar a estabilidade e velocidade do treino. Esta camada é responsável por normalizar o seu *input,* ajustando o mesmo a uma certa escala

Após isto, foi posta uma cada de *Dropout* com o valor de 0.6. Esta camada, é responsável por substituir 60% dos valores de *input* por 0. A mesma foi adicionada por ser a camada indicada para evitar o *overfitting*, um problema muito comum em redes neuronais.

De seguida está uma camada de *MaxPooling* com o *pool* *size* de (2,2), e outra camada de *BatchNormalization*. A camada *MaxPooling* reduz a dimensionalidade do *input,* sendo que, de acordo com o pool size definido, a camada analisa partes de 2x2 *pixels* da imagem de *input* e apenas passa por *output* o valor mais elevado dos quatro *pixels.* Esta camada é bastante eficaz para extrair as *features* mais salientes, e o *pool size* pequeno é indicado para extrair o máximo dessas mesmas *features*.

Após a mesma, existe a última camada Convolucional do modelo, com trinta e dois neurónios, um *filter size* de 3, o *padding* com o valor ‘*same’* e também a ReLU como função de ativação. Esta camada foi adicionada para extrair as características mais importantes do *input,* dado que o mesmo já sofreu várias alterações desde que esta camada foi utilizada.

Terminando, foi utilizada uma camada *Flatten*. Esta camada transforma os dados num *array* com apenas uma dimensão. O surgimento desta camada representa o começo do fim da rede neuronal, dado que ao transformar os dados em valores de apenas uma dimensão operações como as realizadas nas camadas de *MaxPooling* e *Convolutional2D* não poderão ser mais aplicadas. No entanto, ao transformar os dados num *array 1D*, permite que o modelo possa realizar operações específicas que preparem os dados para a camada final.

Após isso, existe uma camada *Dense* com 64 neurónios e a ReLU como função de ativação uma última camada de *BatchNormalization* seguida de uma camada de *Dropout* também com 0.6 de *dropout* *value*. Já a última camada do modelo, é uma camada *Dense* com 100 neurónios com a *softmax* como função de ativação. O número de neurónios deve-se a ser esse o número de classes existentes após a normalização dos dados.

Ao longo do modelo a *ReLU* foi a função de ativação de eleição por introduzir não-linearidade ao modelo, ou seja, capacitar o mesmo de ser capaz de capturar padrões complexos que não podem ser representados por uma função linear e promover o *sparsity* que faz com que uma certa porção dos neurónios do modelo inativos, aumentando a eficiência computacional e reduzindo o *overfitting*. A ReLU apenas foi substituída na última camada, onde a função de ativação utilizada foi a *softmax.* Esta função é, indiscutivelmente, a mais indicada para problemas de classificação de várias classes, pois, realiza uma distribuição de probabilidades ao longo dos neurónios da própria camada, ou seja, atribui a cada neurónio, a probabilidade de o input pertencer a uma certa classe.

De seguida, apresenta-se o código do modelo (Figura 15) e uma representação gráfica do mesmo (Figura 16).

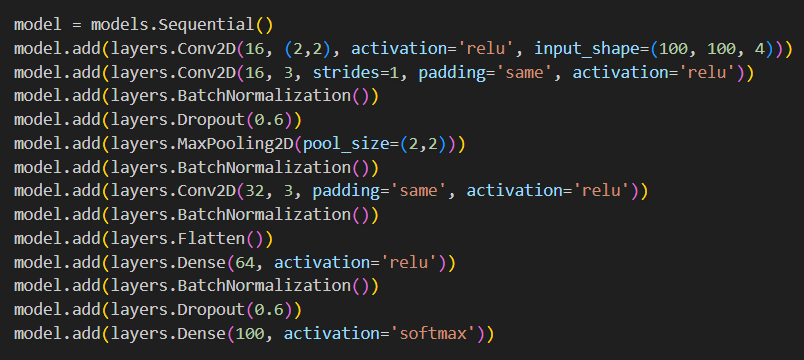


Figura - Código da arquitetura da rede neuronal

A diagram of a graph

Description automatically generated

Figura - Representação gráfica da rede neuronal

### Parâmetros extra

Para este problema em específico, foram testados vários *optimizers* e *loss* *functions* mas os escolhidos no final, por apresentarem os melhores resultados, foram o *adam* e a *sparse* *categorical* *crossentropy*, respetivamente. Foi ainda escolhido a *accuracy* como métrica principal ao compilar o modelo.

A accuracy é, frequentemente, a métrica escolhida para uma CNN por várias razões. Primeiramente, oferece uma interpretação intuitiva sobre a performance da rede neuronal ajudando bastante ao decidir se uma alteração na arquitetura do modelo melhorou ou não a performance do mesmo. A sua versatilidade e aplicabilidade é útil para avaliar diferentes modelos devido ao facto de considerar tanto os acertos quanto os erros do modelo, fornecendo uma avaliação abrangente do desempenho geral do mesmo.

Foram ainda desenvolvidas várias funções *callback* a serem utilizadas durante o *fit*. Primeiramente o *early* *stopping*, que monitorizando a métrica *val\_loss* a partir do *epoch* 30, caso não haja uma melhoria no valor dessa mesma métrica durante 15 ciclos seguidos parará a execução. De seguida o *model* *checkpoint* que monitorizando a métrica val\_*accuracy*, guarda os pesos de cada neurónio num ficheiro específico quando esta métrica atinge o seu valor mais alto registado. Finalmente, como terceira função, existe o *TerminateOnNaN* que não recebe nenhum parâmetro de entrada, apenas termina a execução quando um valor não numérico é encontrado.

## Treino do modelo

### Primeira abordagem

Primeiramente, já que seria impossível a máquina ter em memória mais de vinte e duas mil imagens, foram selecionadas aleatoriamente trezentas e vinte imagens de cada *label* (Tabela 2)*,* com o seu tamanho original de duzentos por duzentos *pixels*, com um *test size* de 45%.

Tabela - Quantidade de dados a ser usada na primeira abordagem.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Valor normalizado 0 | Valor normalizado 1 | Valor normalizado 2 | Valor normalizado 3 | Valor normalizado 5 |
| 320 | 320 | 168 | 16 | 8 |

De seguida, com um *batch size* de 32 foi realizado um treino onde os seus resultados (Tabela 3) foram os seguintes.

Tabela - Resultados da primeira abordagem.

|  |  |  |  |
| --- | --- | --- | --- |
| *Loss* | *Accuracy* | *Validation* *loss* | *Validation* *accuracy* |
| 4.3722 | 0.2713 | 4.2314 | 0.3760 |

Com esses resultados, foi assumido que o problema estaria na discrepância na quantidade de dados de cada valor passando assim, para uma abordagem mais balanceada nesse aspeto.

### Segunda abordagem

Para esta abordagem foi tentado usar a mesma quantidade de imagens para cada valor possível. Para fazer isso de forma que fosse facilmente alterável foi criado um conjunto de variáveis.

Foram criados dois dicionários com chaves de zero a cem, onde, o dicionário denominado de *counter dictionary,* tem todos os seus valores a zero e o dicionário denominado de *limit dictionary* tem todos os seus valores com a quantidade máxima de cada valor.

Ao ler as imagens do *dataset*, é verificado antes qual é o valor da imagem, e, usando esse valor como chave para ambos os dicionários, a imagem apenas é adicionada à lista de imagens para treino se, o valor no *counter dictionary* for inferior ao valor do *limit dictionary*.

Para esta abordagem foram usadas apenas vinte imagens de cada valor, sendo a quantidade de valores a seguinte (Tabela 4).

Tabela - Quantidade de dados a ser usada na segunda abordagem.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Valor normalizado 0 | Valor normalizado 1 | Valor normalizado 2 | Valor normalizado 3 | Valor normalizado 5 |
| 20 | 20 | 20 | 16 | 8 |

De seguida, foi realizado um treino onde os seus resultados foram os seguintes (Tabela 5).

Tabela - Resultados da segunda abordagem.

|  |  |  |  |
| --- | --- | --- | --- |
| *Loss* | *Accuracy* | *Validation* *loss* | *Validation* *accuracy* |
| 0.1181 | 1.0000 | 2.8125 | 0.2895 |

Sendo que os resultados não foram promissores, a abordagem tomada foi reduzir o tamanho das imagens para 100100 *pixels*.

### Terceira abordagem

Para esta abordagem, foi criada uma função (Figura 15) que reduz o tamanho da imagem no momento da adição da mesma à lista de imagens para treino.

A picture containing text, screenshot, font

Description automatically generated

Figura - Código da função de redimensionamento da imagem

Para esta abordagem foram usadas apenas trinta e duas imagens de cada valor, sendo a quantidade de valores a seguinte (Tabela 6).

Tabela - Quantidade de dados a ser usada na terceira abordagem.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Valor normalizado 0 | Valor normalizado 1 | Valor normalizado 2 | Valor normalizado 3 | Valor normalizado 5 |
| 32 | 32 | 32 | 16 | 8 |

De seguida, foi realizado um treino onde os seus resultados foram os seguintes (Tabela 7).

Tabela - Resultados da terceira abordagem.

|  |  |  |  |
| --- | --- | --- | --- |
| *Loss* | *Accuracy* | *Validation* *loss* | *Validation* *accuracy* |
| 0.0290 | 1.0000 | 0.2586 | 1.0000 |

Sendo a terceira abordagem um sucesso, a abordagem final foi alterar os dados de forma que pudesse ser feita uma previsão em relação ao futuro, para isso, foram feitos três treinos a mais.

De forma a ser possível fazer uma previsão ao ser adicionada uma imagem para treino a sua *label* foi o valor da precipitação uma, duas e três horas após.

De forma a realizar treinos eficazes dada a grande diferença de quantidade de valores, para cada hora, primeiro foi realizado um treino apenas com cento e noventa imagens com o valor normalizado de 0, 1 e 2, os três níveis de precipitação mais comuns no *dataset*, e com um *test* *size* de 40% e após isso foram utilizadas a maior quantidade de imagens possível de modo a que houvesse um *dataset* balanceado com todos os valores normalizados obtidos.

Com uma hora de diferença, os resultados foram os seguintes (Tabela 8).

Tabela - Resultados da abordagem final para uma hora de diferença entre a imagem e a label

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *Loss* | *Accuracy* | *Validation* *loss* | *Validation* *accuracy* |
| Com valores 0,1 e 2 | 0.2747 | 0.9231 | 0.6504 | 0.9048 |
| Com todos os valores | 1.2858 | 0.7273 | 0.8015 | 0.8889 |

Já para o treino com duas horas de diferença, os resultados foram os seguintes (Tabela 9).

Tabela - Resultados da abordagem final para duas horas de diferença entre a imagem e a label

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *Loss* | *Accuracy* | *Validation* *loss* | *Validation* *accuracy* |
| Com valores 0,1 e 2 | 0.0017 | 1.0000 | 0.6732 | 0.8750 |
| Com todos os valores | 0.0536 | 1.0000 | 1.0226 | 0.8750 |

Finalmente, para o treino com três horas de diferença, os resultados foram os seguintes (Tabela 10).

Tabela - Resultados da abordagem final para três horas de diferença entre a imagem e a label

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *Loss* | *Accuracy* | *Validation loss* | *Validation accuracy* |
| Com valores 0,1 e 2 | 0.1037 | 0.9655 | 0.4542 | 0.9138 |
| Com todos os valores | 0.7306 | 0.7500 | 0.6122 | 0.9500 |

# *Website* desenvolvido para aplicar o modelo

No website desenvolvido, para que fosse possível a aplicação do modelo de aprendizagem, foi necessário, numa primeira fase, desenvolver uma API em *Flask* para enviar os dados para o *frontend* a cada 10 minutos com as previsões climatéricas de Portugal, que fornecem estimativas de precipitação a cada estação mais perto do centro de cada distrito de Portugal. A API desenvolvida em *Flask* utiliza o mesmo modelo criado e treinado anteriormente, utilizando os pesos guardados dos melhores testes realizados na fase de treino.

## API em *Flask*

Para poder apresentar os resultados da previsão do modelo no website, foi fundamental o desenvolvimento de uma API em *Flask*. Esta API teve como ponto de partida a receção da imagem mais recente, fornecida pelos radares do IPMA, através de *web scrapping* realizado no *website* da própria organização. Para isso, foi necessário construir um URL com a data e hora atualizadas, seguindo um formato específico.

O processo de obtenção da imagem foi o seguinte:

1. Obter a data e hora em *UTC*, usando a função *datetime.utcnow*.
2. Arredondar a data e hora para o múltiplo de 5 minutos mais próximo, subtraindo o resto da divisão por 5.
3. Subtrair mais 10 minutos, para garantir que a imagem estivesse disponível na API do IPMA.
4. Formatar a data e hora de acordo com o padrão “%Y-%m-%dT%H%M”[[1]](#footnote-1).
5. Construir o URL para fazer o pedido *Hypertext Transfer Protocol* (HTTP) à API do IPMA, usando uma parte fixa e uma parte variável (data e hora).
6. Obter a imagem dos radares de Portugal com o conteúdo da resposta do pedido HTTP.
7. Retornar uma mensagem de erro no formato *JavaScript Object Notation* (JSON), caso ocorresse algum erro durante o pedido HTTP.

A screenshot of a computer program

Description automatically generated with medium confidence

Figura 18 – Código da função *get\_radar\_image*().

A imagem obtida será então usada no *endpoint* *process\_image* que recebe um pedido GET e retorna um objeto no formato JSON com as previsões de precipitação para cada estação meteorológica e para cada hora de diferença.

O *endpoint* da previsão de precipitação faz o seguinte processo:

1. Obter a imagem atual dos radares a partir da função *get\_radar\_image*.
2. Verificar se a imagem foi obtida com sucesso. Se não, retornar uma mensagem de erro.
3. Criar um dicionário onde as *keys* são as horas de previsão (1, 2 e 3).
4. Obter a lista dos ficheiros de pesos do modelo que estão guardadas numa pasta com a ajuda da função *listdir*.
5. Para cada id na lista de ids das estações, fazer as seguintes operações:
   1. Recortar a imagem do radar com base nas coordenadas fornecidas pelo dicionário *station\_box\_dict* com a ajuda da função *crop* do módulo *PIL*.
   2. Redimensionar a imagem recortada pela função *resize\_image*.
   3. Converter a imagem para um array com a função *img\_to\_array*
   4. Normalizar o *array* da imagem com a função normalize do *TensorFlow*, que recebe o *array* e o eixo com argumentos e retorna um novo *array* normalizado.
   5. Fazer a previsão com o valor desnormalizado da precipitação obtido pela função *desnormalize\_precipitation\_value*.
6. Adicionar a previsão ao dicionário
7. Retornar o dicionário completo no formato JSON.

A screen shot of a computer program

Description automatically generated with medium confidence

Figura - Código do endpoint *process\_image*().

## *Web* *Meteo*

O *website* desenvolvido permite ao utilizador aceder aos resultados das previsões de precipitação em Portugal, em cada distrito logo quando o *website* é carregado. A data e hora da previsão é a atual, mas com 10 minutos de atraso e arredondada por múltiplos de 5. Os resultados da previsão aparecem todos numa única tabela.

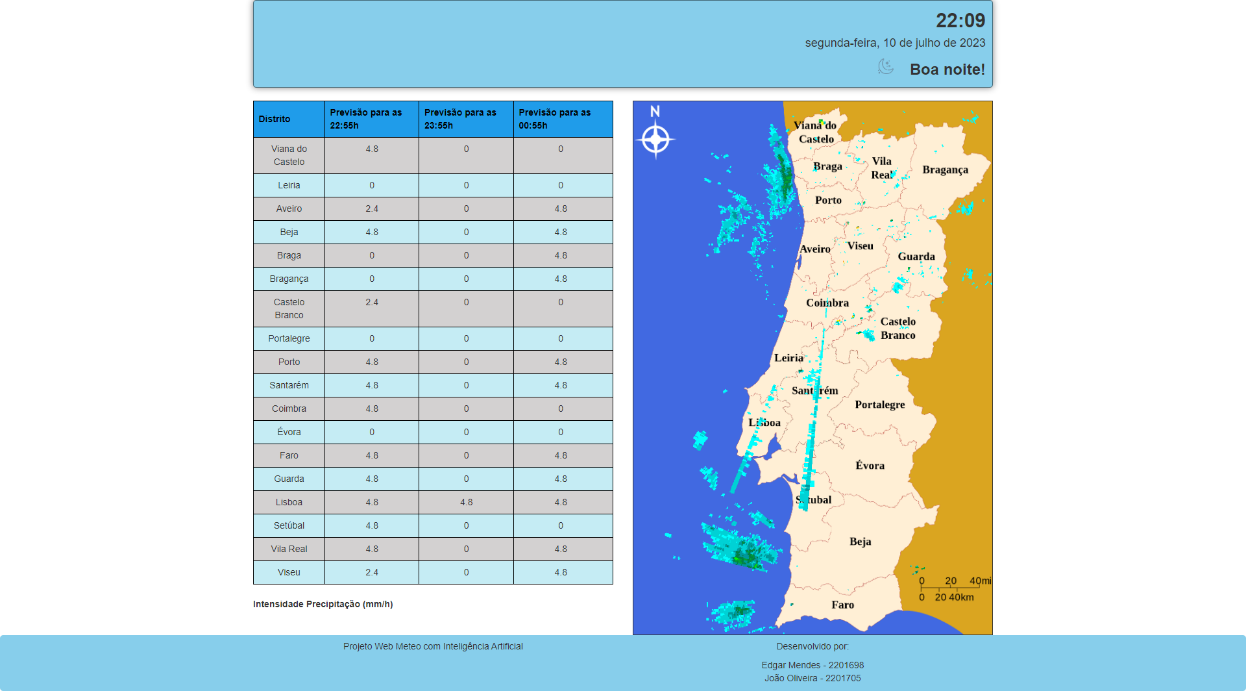


Figura - *Website* *Web Meteo.*

### Implementação

O *website* consiste num ficheiro principal *app.vue* e outros três componentes: *Clock.vue*, *ImagePrediction.vue* e *Mapa.vue*.

O *app.vue* (Figura 19) é responsável por criar uma *navbar*, onde apresenta o relógio, a data e uma saudação de acordo com a hora do dia, importado a partir do componente *Clock.vue*. Também importa o *ImagePrediction.vue* e *Mapa.vue* que os coloca no conteúdo do website em 2 colunas separadas. Além disso, o *app.vue* define um intervalo de tempo de 5 minutos para chamar as funções principais do *ImagePrediction.vue* e do *Mapa.vue*, que são responsáveis por atualizar os dados da previsão meteorológica e da imagem do radar, respetivamente (Figura 20). O *app.vue* também cria um rodapé com o título e os autores do projeto.

A screen shot of a computer program

Description automatically generated

Figura - Código do *template* do *app.vue*.

A screen shot of a computer code

Description automatically generated

Figura - Código do *script* do *app.vue*.

O *Clock.vue* é um componente que mostra o relógio, a data e uma saudação de acordo com a hora do dia (Figura 21). O *Clock.vue* usa o objeto *Date* do JavaScript para obter a hora e a data atuais, e as formata usando o método *toLocateString* com a opção ‘pt-PT’. O *clock.vue* também usa uma condição para determinar se é de manhã, tarde ou noite, e mostra uma imagem do sol ou da lua e uma saudação correspondente. O *Clock.vue* atualiza o relógio a cada minuto usando o método *setInterval* (Figura 22).

A computer screen with text on it

Description automatically generated

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

A screen shot of a computer program

Description automatically generated

Figura - Código do *script* do *Clock.vue*.

O *ImagePredictions.vue* é um componente que mostra uma tabela com as previsões da precipitação para os distritos de Portugal, baseadas numa imagem do radar (Figura 23). O *ImagePredictions.vue* usa o módulo *axios* para fazer um pedido GET à API Flask, que é responsável por processar a imagem do radar e retornar um objeto JSON com os dados das previsões. O *ImagePredictions.vue* então formata os dados recebidos num objeto que pode ser usado para preencher a tabela, usando um *loop* v-for. O *ImagePredictions.vue* também mostra a hora em que as previsões são feitas, que é obtida usando o objeto *Date* do JavaScript e arredondada para o múltiplo de 5 minutos mais próximo subtraindo mais 10 minutos (Figura 24).



Figura - Código do *template* do *ImagePrediction.vue*.

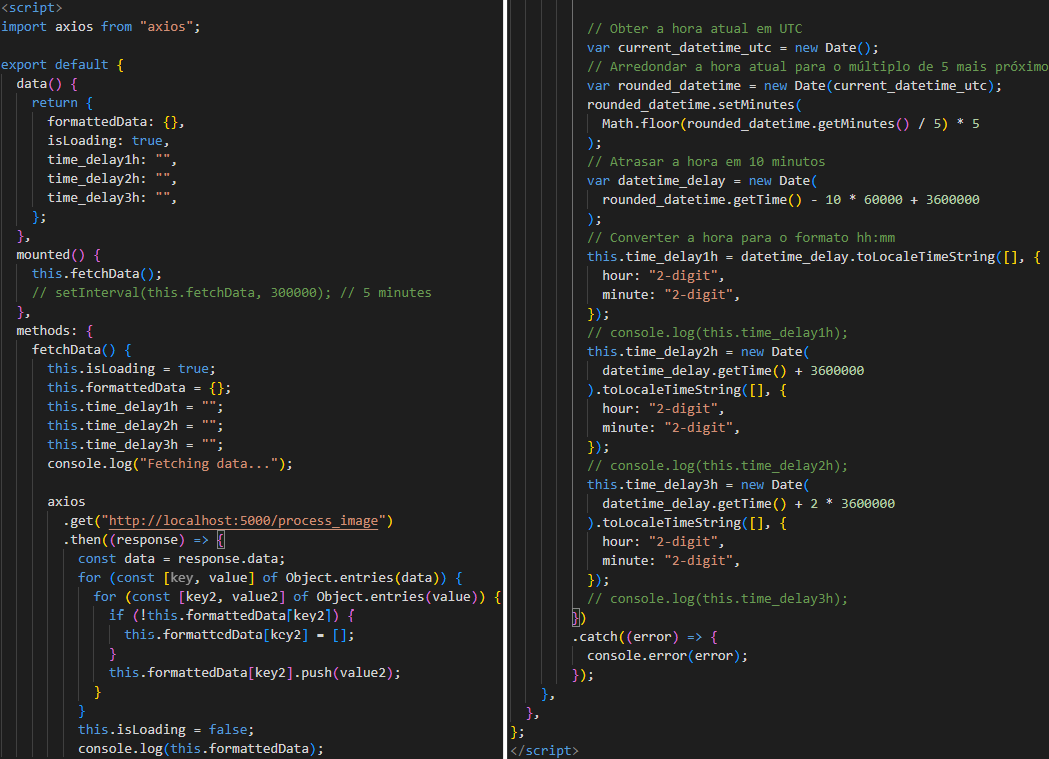


Figura - Código do *script* do *ImagePredictions.vue*.

O Mapa.vue é um componente que mostra uma imagem do mapa de Portugal com a imagem do radar encaixada pelos limites da imagem do mapa e sobreposta (Figura 25). O Mapa.vue usa o módulo *axios* para fazer um pedido GET à API Flask, que é responsável por obter a imagem do radar mais recente e retorná-la em formato base64 (Figura 26). O Mapa.vue então usa o atributo src da tag img para mostrar a imagem do radar, usando uma posição absoluta e um tamanho ajustado para se sobrepor ao mapa de Portugal, que é uma imagem estática na pasta *assets* (Figura 27).

A screen shot of a computer screen

Description automatically generated

Figura - Código do *template* do Mapa.vue.

A screen shot of a computer program

Description automatically generated

Figura - Código do *script* do Mapa.vue.

A screen shot of a computer code

Description automatically generated

Figura - Código css do Mapa.vue.

## Instalação e uso

O projeto *WebMeteo* consiste numa aplicação *web* que mostra as previsões da precipitação em cada estação referente a cada distrito de Portugal após 1, 2 e 3 horas. A aplicação *web* é feita com *vue.js* e os dados são fornecidos por uma API feita com *Flask*. As instruções seguintes explicam como executar o projeto.

### Requisitos

Antes de executar o projeto, é preciso ter instalado os seguintes programas:

* *Pyhton*
* *Flask*
* *Node.js*
* *Npm*

### Passos

Para executar o projeto é necessário realizar os seguintes passos:

1. Num terminal a diretoria deverá ser alterada para a seguinte pasta WebMeteoPI/FlaskFramework.
2. Será também necessário instalar as seguintes bibliotecas para o código *Flask* funcionarcorretamente, usando os seguintes comandos:
   * pip install flask
   * pip install tensorflow
   * pip install Pillow
   * pip intall flask-cors
3. Após isso será necessário executar a API, usando o comando py app.py run.
4. Noutro terminal a diretoria deverá ser alterada para a seguinte pasta WebMeteoPI/vue-client.
5. Será também necessário instalar as dependências necessárias para o *vue.js*, usando o comando npm install.
6. Para executar o *website* deverá ser usado o comando npm run dev.
7. Já no *browser*, para ver a aplicação *web* a funcionar, é necessário aceder a este link <http://localhost:5173/>

# Análise de resultados

A capacidade de uma rede neuronal identificar algo é uma coisa que tem recebido muita atenção e esforço por parte de toda a gente que se dedica ao ramo da inteligência artificial. Ferramentas presentes no nosso dia-a-dia como, reconhecimento facial, que se encontra na maioria dos telemóveis atuais, reconhecimento de voz, entre muitos outros exemplos têm todos como base redes neuronais.

Estas redes ou modelos podem ser avaliadas de várias formas, sendo que, neste projeto, a métrica considerada mais importante, foi a *validation* *accuracy*. O valor desta métrica é a taxa de acerto do modelo para imagens nunca vistas pelo modelo, mas que estão devidamente classificadas. Com as ferramentas que utilizámos o valor desta métrica está compreendido entre 0 e 1 tendo de ser multiplicado por 100 para dar um valor em percentagem.

Mesmo assim, também foi dada atenção a outras métricas sendo elas também importantes. Primeiramente a accuracy, que se traduz na taxa de acerto para as imagens com que o próprio modelo foi treinado, sendo assim, o valor da accuracy pode às vezes dar resultados muito bons, mas, na prática, o modelo não ter uma boa capacidade de classificação. Isto acontece quando existe um dos problemas mais comuns, no entanto dos mais perigosos para o bom funcionamento de um modelo, o overfitting. Este problema acontece quando o modelo ‘viu’ demasiadas vezes as mesmas imagens então perde a sua capacidade de generalização, ou seja, ao ver uma imagem da mesma classe, mas que seja significativamente diferente, o modelo não será capaz da classificar corretamente.

Para evitar o *overfitting* ao máximo, dado que, ao longo do projeto, existiu uma grande carência de dados por fatores externos tanto aos alunos quanto aos professores, durante o processo de treino foram realizadas duas etapas. Na primeira etapa, foi realizado um treino onde foram utilizados apenas dados que existiam em ‘grande’ quantidade, ou seja, como mencionado na descrição do processo, imagens cujo a sua classificação fosse o valor normalizado de 0, 1 ou 2. Já na segunda e última etapa, foi realizado um treino com imagens de todas as classificações, mas com *dataset* equilibrado, ou seja, utilizando o mesmo número de imagens para cada classe, sendo assim, o treino foi realizado com apenas 8 imagens de cada classe dado que esse era o número total de imagens pertencentes a essa classe. Sendo assim, ainda que possa ter acontecido, inevitavelmente, um certo *overfitting* para as classes mais escassas, para as classes onde existiam bastantes dados foram utilizados uma grande quantidade de dados diferentes em cada treino sendo que para essas, é considerado que o modelo tem uma boa capacidade de generalização.

Relativamente aos resultados em si, após o treino por etapas e o *data augmentation*, foi possível notar uma melhoria na previsão realizada, sendo que para as previsões das próximas uma, duas e três horas foi obtido um valor de *validation* *accuracy* de 88,89%, 87,5% e 95% respetivamente.

Ainda assim, o modelo não está preparado para imagens de radar de chuva intensa dado que esses dados não estiveram disponíveis no treino.

# Conclusão

O propósito deste projeto foi a criação de um *website* que fosse capaz de mostrar a precipitação num futuro próximo, prevista por uma rede neuronal.

Em termos de precisão, o modelo ficou um pouco aquém da expectativa devido à falta de dados pelas limitações do IPMA, e pelo mesmo ter recusado o nosso pedido de fornecer mais. Ainda assim, o projeto tem implementado todas as funções necessárias para esses dados serem obtidos por um futuro utilizador do mesmo.

Mesmo assim, com o desenvolvimento deste projeto foi possível criar uma ferramenta com grande potencial, tendo até capacidades de interpretar detalhes e padrões que não seria possível analisar de outra forma, para esta área que já se provou ser algo que tem uma grande influencia nas decisões de alguém.

Apesar de todos os obstáculos que tivemos ao longo do projeto, tanto os que conseguimos ultrapassar como o que não conseguimos, aprendemos bastante com todos os erros e problemas, podendo dizer que o projeto ficou próximo do seu pico de qualidade, ignorando os problemas externos a nós, tendo ainda, como sempre, espaço para melhorias.

# Bibliografia ou Referências Bibliográficas

|  |  |
| --- | --- |
| [1] | IBM, “What is deep learning?,” [Online]. Available: https://www.ibm.com/topics/deep-learning. |
| [2] | Viceri, “Arquiteturas de Redes Neurais Convolucionais para reconhecimento de imagens,” 29 julho 2020. [Online]. Available: https://viceri.com.br/insights/arquiteturas-de-redes-neurais-convolucionais-para-reconhecimento-de-imagens/. |
| [3] | Vue School, “Vue.js,” [Online]. Available: https://vuejs.org/guide/introduction.html. |
| [4] | Monocubed, “What are the Advantages of Vue js Framework in Web Development?,” 5 agosto 2021. [Online]. Available: https://www.monocubed.com/blog/advantages-of-vue-js/. |

1. %Y – ano; %m – mês; %d – dia; %H – hora; %M – minuto; [↑](#footnote-ref-1)