NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE INFORMÁTICA

JOÃO HENRIQUE D'OLIVEIRA PALMEIRO

Degree in Computer Science

PLATAFORMA PARA TREINO DE MODELOS DE APRENDIZAGEM AUTOMÁTICA RELACIONADOS COM AGRICULTURA

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa

Abril, 2024

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE INFORMÁTICA

PLATAFORMA PARA TREINO DE MODELOS DE APRENDIZAGEM AUTOMÁTICA RELACIONADOS COM AGRICULTURA

JOÃO HENRIQUE D'OLIVEIRA PALMEIRO

Degree in Computer Science

Orientador: Carlos Augusto Isaac Piló Viegas Damásio
Professor Associado, Universidade NOVA de Lisboa

Coorientadores: Fernando Pedro Reino da Silva Birra
Professor Auxiliar, Universidade NOVA de Lisboa

João Carlos Gomes Moura Pires
Professor Associado, Universidade NOVA de Lisboa

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
Abril, 2024

Plataforma para Treino de Modelos de Aprendizagem Automática relacionados com Agricultura

Este documento foi gerado com o processador (pdf/Xe/Lua)LaTeX e o modelo NOVAthesis (v7.1.18) [44].

Agradecimentos

Em primeiro lugar, gostaria de agradecer ao professor Carlos Damásio e aos professores Fernando Birra e João Moura Pires por me terem acompanhado e ajudado ao longo de todo o processo de desenvolvimento desta dissertação.

Tenho também a agradecer aos meus amigos, tanto aos que conheci durante o curso que agora termina, como àqueles que conheço desde sempre. A sua ajuda foi da maior importância e a preocupação demonstrada pelas recorrentes perguntas "Como vai a tese?" mantiveram-me focado durante o processo. Finalmente posso responder: "A tese está acabada".

Quero também deixar uma palavra de grande agradecimento à minha namorada, Marina, que me acompanha e apoia há praticamente seis anos. A sua alegria e boa disposição ajudaram-me durante todo o curso e ainda mais durante o desenvolvimento desta dissertação.

Por último, agradeço profundamente à minha família: aos meus pais, Gabriela e João, à minha irmã, Alice, à minha tia, Teresa, e aos meus avós, Irene e João. Os seus esforços para me proporcionarem uma excelente educação e todas as condições para que pudesse ter sucesso devem ser aqui reconhecidos. Vou continuar a esforçar-me para que tudo isso não tenha sido em vão.

Resumo

Atualmente, a monitorização de pragas e doenças, bem como a identificação de espécies em campos agrícolas, implica custos elevados de deslocação aos terrenos por parte de profissionais especializados. Particularmente para pequenos produtores, isto pode ser um entrave ao diagnóstico de problemas como doenças e pragas em culturas agrícolas, dificultando o seu tratamento atempado.

Para tentar minimizar este problema, é benéfico fazer uso dos recursos tecnológicos atuais, criando modelos de aprendizagem automática capazes de identificar problemas a partir de fotografias tiradas no local. Além do mais, é importante garantir que se conseguem partilhar esses dados entre os técnicos agrícolas, para que o seu trabalho conjunto possa garantir a validade das classificações.

Por outro lado, o universo da aprendizagem automática tem vindo a crescer nos últimos anos, estando a ser criados modelos cada vez mais avançados e eficazes para determinadas áreas. Desta forma, pretende-se também garantir que a forma como a classificação é feita possa evoluir ao longo do tempo, permitindo a profissionais da área de informática o desenvolvimento ou adição de novos modelos de aprendizagem.

Assim, esta dissertação tem como objetivo o desenvolvimento, estudo e testagem de uma plataforma desenvolvida usando os serviços da *Google Cloud Platform* para treinar, visualizar e utilizar modelos de aprendizagem automática. Esta permite o armazenamento de imagens em repositórios, de uma forma organizada que permite a anotação automática dos dados. Por outro lado, permite o carregamento de templates de código por parte de profissionais qualificados, bem como a sua instanciação com parâmetros relevantes. Estes permitem desenvolver *pipelines* que serão responsáveis por toda uma cadeia de procedimentos capazes de treinar modelos de aprendizagem automática. A plataforma permite executar os templates instanciados de forma a desenvolver esses modelos, que ficam disponíveis para uso de qualquer interveniente e para os quais se podem visualizar métricas de avaliação relevantes.

Palavras-chave: Aprendizagem Automática, Computação em Cloud, Banco de Dados, Agricultura.

Abstract

Currently, the monitoring of pests and diseases, as well as the identification of species in agricultural fields, involves high costs of traveling to the land by specialized professionals. Particularly for small producers, this can be an obstacle to diagnosing problems such as diseases and pests in agricultural crops, making their timely treatment difficult.

To try to minimize this problem, it is beneficial to make use of current technological resources, creating machine learning models capable of identifying problems from photographs taken on site. Furthermore, it is important to ensure that this data can be shared between agricultural technicians, so that their joint work can guarantee the validity of the classifications.

On the other hand, the world of machine learning has been growing in recent years, with increasingly advanced and effective models being created for certain areas. In this way, it is also intended to ensure that the way in which classification is carried out can evolve over time, allowing IT professionals to develop or add new learning models.

Therefore, this dissertation aims to develop, study and test a platform developed using *Google Cloud Platform* services to train, visualize and use machine learning models. This allows the storage of images in repositories, in an organized way that allows automatic annotation of data. On the other hand, it allows the loading of code templates by qualified professionals, as well as their instantiation with relevant parameters. These allow the development of *pipelines* that will be responsible for an entire chain of procedures capable of training machine learning models. The platform allows you to run instantiated templates in order to develop these models, which are available for use by any stakeholder and for which relevant evaluation metrics can be viewed.

Keywords: Machine Learning, Cloud Computing, Datasets, Agriculture.

Índice

Índice de Figuras

Índice de Listagens

Siglas

API *Application Programming Interface (pp. 3, 34, 35, 37–39, 41, 43, 44, 47, 49, 53, 55, 58–60, 62, 63, 66–70, 72–74, 76, 79, 82, 83, 90–92, 98, 101–105, 107, 108, 112, 113, 124–127)*

CNN Redes Neuronais Convolusionais *(pp. 20–22, 25)*
CSS *Cascading Style Sheets (pp. 48, 49, 51, 72)*

DGAV Direção Geral de Alimentação e Veterinária *(p. 1)*
DNN *Deep Neural Networks (pp. 20, 21)*

FEEI Fundos Europeus Estruturais e de Investimento *(p. 2)*

GAE *Google App Engine (p. 90)*
GCD *Google Cloud Datastore (pp. 59, 84)*
GCP *Google Cloud Platform (pp. 58, 109, 112)*
GCS *Google Cloud Storage (pp. 59, 65, 86)*

HTML *HyperText Markup Language (pp. 48, 49, 51, 72)*
HTTP *HyperText Transfer Protocol (pp. 67, 68, 73, 92, 113)*
HTTPS *HyperText Transfer Protocol Secure (pp. 68, 91)*

SGD *Stochastic Gradient Descent (p. 20)*
SPA *Single Page Application (pp. 48, 49, 57, 102)*
SSE *Server Sent Events (pp. 53, 74, 93, 94, 105)*
SUS *System Usability Scale (pp. 108, 113–117, 120)*
SVM *Support Vector Machine (p. 25)*

1

Introdução

1.1 Contexto e Motivação

A Agricultura é uma atividade económica primária que desempenha e continuará a desempenhar um papel fundamental na obtenção de matérias primas para a alimentação. No decorrer dos anos, foram criadas técnicas cada vez mais sofisticadas para maximizar a produção e enfrentar os problemas pré-existentes ou os que foram surgindo ao longo do tempo. Atualmente, a Agricultura continua a enfrentar algumas ameaças, tais como as alterações climáticas, a urbanização ou as pragas e doenças causadas por espécies invasoras ou emergentes [78].

A globalização veio aumentar este último fator, difundindo vários tipos de pragas em redor do globo, sendo estimadas perdas de produção agrícola e florestal causadas por insetos ou patogénicos invasores de aproximadamente 40 mil milhões de dólares apenas nos EUA [56]. Portugal também tem registado um aumento do número de novas pragas e doenças das plantações agrícolas, sendo um problema cada vez mais atual e que apenas tem tendência a agravar-se [45].

A solução para controlar os malefícios provocados pelas doenças e pragas passa pela identificação rápida das mesmas, atuação imediata e prevenção da propagação para outras culturas agrícolas. Para tal, a Direção Geral de Alimentação e Veterinária (DGAV), de acordo com instruções do próprio Ministério da Agricultura e da legislação comunitária, tem mecanismos de incentivo à prospeção fitossanitária para deteção de pragas e doenças [79, 66]. Estas necessitam de recursos humanos especializados e deslocações aos terrenos agrícolas, incorrendo em custos financeiros avultados.

Para tentar não só diminuir os encargos financeiros para instituições e produtores, mas também para melhorar o diagnóstico, seria muito importante ter um sistema que pudesse servir de auxiliar a todos os intervenientes no processo de prospeção. Daí surge a motivação para esta dissertação, que aborda o desenvolvimento de uma plataforma capaz de guardar dados devidamente anotados e de os associar a tarefas de aprendizagem automática relevantes. Além do mais, todo o processo de acesso, anotação e aprendizagem deve resultar da colaboração entre os interessados no sistema, incluindo especialistas em

aprendizagem automática, produtores e técnicos agrícolas.

Esta dissertação está enquadrada no projeto "Smart Farm 4.0: soluções inteligentes para uma agricultura sustentável, preditiva e autónoma". Como o nome indica, o projeto pretende desenvolver soluções que permitam "otimizar a utilização dos recursos, a sustentabilidade e a resiliência dos sistemas agrícolas e a sua rentabilidade e valor acrescentado" [70]. O projeto conta com vários parceiros e entidades, sendo cofinanciado pelos Fundos Europeus Estruturais e de Investimento (FEEI).

A dissertação e sistema subjacente contou com o apoio financeiro do projeto atrás mencionado, através de uma bolsa de investigação para licenciado (BIL6), cuja referência é: "Smart Farm 4.0 (POCI-01-0247-FEDER-046078). Assim, o trabalho desenvolvido enquadra-se nos objetivos enunciados no parágrafo anterior, usando tecnologias 4.0 (incluindo Inteligência Artificial e Internet das Coisas - IoT) para aplicações no setor agrícola e de forma a assegurar a otimização de recursos, autonomia e rentabilidade dos sistemas de apoio. Ainda no âmbito deste projeto mobilizador, foi possível contactar profissionais na área da agricultura para melhorar a qualidade geral do sistema.

1.2 Problema

Atualmente, a prospeção de pragas ou doenças pode levar a custos financeiros difíceis de comportar, sobretudo para pequenos produtores que não consigam apoios de outras instituições ou organizações. Caso não haja uma identificação atempada das pragas ou doenças e o seu tratamento adequado e rápido, corre-se o risco de propagação para terrenos fronteiriços, sendo mais difícil fazer a contenção dos problemas e imprevisível a dimensão que estes podem alcançar.

Para além da prospeção em si, muitos profissionais da área agrícola poderiam beneficiar de um banco de dados único, onde fosse possível aceder a uma grande quantidade de conteúdo multimédia, disponibilizado por produtores e organizado de acordo com os melhores fatores para cada caso (localização, características fonológicas, nome de espécie...). Tal banco de dados permitir-lhes-ia analisar espécies agrícolas provenientes de vários locais e fazer a comparação entre elas, permitindo inferir o estado de disseminação das doenças. Além do mais, seria também mais fácil inter-relacionar características fonológicas com base em fatores variados, como a localização e as características do meio ambiente, percebendo semelhanças ou diferenças com base nestes critérios.

Por outro lado, a análise, mesmo que feita por profissionais, pode estar errada ou incompleta. Esta carece de um mecanismo automático que possa prevenir erros ou até fazer uma análise automática sem intervenção de qualquer ser humano. Para isso, poder-se-iam usar modelos de aprendizagem automática capazes de desempenhar este papel e servir de auxiliar à classificação feita pelos profissionais.

Por último, é sabido que a evolução da Informática tem sido crescente, em particular da área da Inteligência Artificial, com o surgimento de tecnologias cada vez mais avançadas. Todavia, um determinado tipo de tecnologia pode ser mais ou menos indicado consoante

o tipo de problemas e o momento em que surgem. Assim, é essencial garantir que os sistemas permanecem atualizados caso surjam outros métodos, possibilitando os melhores resultados possíveis.

1.3 Objetivos

O projeto "*Smart Farm 4.0*: soluções inteligentes para uma agricultura sustentável, preditiva e autónoma", onde esta dissertação está inserida, pretende tratar os problemas mencionados na subsecção anterior. Desta forma, este documento aborda o desenvolvimento de uma plataforma para o treino de modelos de aprendizagem automática preparados para lidar com conteúdo multimédia relacionado com a Agricultura.

A plataforma é então composta por uma componente de servidor e lógica subjacente, condensada numa *Application Programming Interface* (API) REST fiável, escalável e robusta, que garante alta disponibilidade para qualquer tipo de utilizador. É ainda capaz de armazenar uma grande quantidade de imagens de forma persistente, organizadas em repositórios editáveis pelos utilizadores e acessíveis de forma rápida a partir de qualquer ponto geográfico. Para tal, a API foi desenvolvida com recurso a serviços da *Google Cloud Platform*, aliando às características mencionadas um custo mais eficiente.

Os dados guardados podem ser anotados automaticamente pelo sistema com base no nome das pastas em que estão inseridos, sendo a organização dos conteúdos essencial para esse processo. Esta estratégia também permite automatizar o processo para conjuntos de dados maiores. Para complementar este tipo de anotação, a plataforma oferece algoritmos de aprendizagem pré-treinados, capazes de classificar automaticamente os dados sem intervenção externa. Para tal, são apresentados os *pipelines* de aprendizagem automática que foram usados na fase de treino, bem como os próprios modelos, associados a uma variedade de métricas de avaliação relevantes para a compreensão do seu funcionamento e eficácia.

Para garantir a continuidade da plataforma e a máxima capacidade de previsão automática, é possível adicionar modelos pré-treinados ou criar outros completamente novos. Isto é conseguido através do upload de código por parte de pessoal especializado, criando *templates* que possam ser instanciados com vários tipos de parâmetros. Estes podem ser transferidos para máquinas locais, estando preparados para executar num ambiente apropriado e com as credenciais certas, mas são especialmente destinados a uma execução automática num servidor específico para o efeito. Este servidor, por ter já instaladas as dependências necessárias ao funcionamento do código presente na plataforma, bem como as credenciais necessárias para desencadear a execução diretamente na Cloud, é o ambiente indicado para correr os *templates*.

Noutra perspetiva, e para que seja possível aceder ao sistema e executar os pedidos da API mencionada anteriormente, foi desenvolvida uma aplicação Web para realizar os pedidos nela disponíveis. Esta componente de *frontend* garante acesso à maioria das

funcionalidades do sistema, de forma a que haja uma forma de contacto mais amigável para qualquer tipo de utilizador.

Todo o sistema foi concebido de forma a garantir a colaboração entre três entidades: utilizadores, técnicos agrícolas e profissionais ou investigadores na área de inteligência artificial. Assim, utilizadores e produtores podem fazer o upload de imagens que serão guardadas de forma duradoura e organizada no sistema, podendo depois ser classificadas de forma automática. Os utilizadores podem ainda ter acesso ao conteúdo multimédia e reorganizá-lo em pastas diferentes, de acordo com os seus critérios e de forma a que o sistema possa fazer a anotação dos mesmos da forma mais correta possível. Especialistas em inteligência artificial podem colaborar entre si para criar os melhores modelos e *pipelines* de aprendizagem automática, fazendo a afinação dos mesmos ou simplesmente criando outros mais eficientes para um determinado tipo de problema.

1.4 Estrutura do Documento

Este documento de dissertação encontra-se organizado em 7 capítulos principais (excluindo este capítulo introdutório). Os três capítulos seguintes abordam o estado da arte da área de inteligência artificial e das tecnologias usadas no desenvolvimento da plataforma, estando os dois capítulos subsequentes reservados para aspetos de modelação, arquitetura e implementação. Os capítulos finais abordam a avaliação do sistema e as conclusões gerais do projeto e consequente dissertação. A estrutura mais detalhada é então a seguinte:

- **Estado da Arte em Aprendizagem Automática**: capítulo em que são introduzidos alguns conceitos de Aprendizagem Automática e Profunda, relevantes para o projeto desenvolvido. Assim, há uma explicação do processo de aprendizagem, falando-se depois nas técnicas de aprendizagem mais eficazes para as tarefas que se pretendem tratar nesta dissertação;

- **Desenvolvimento Cloud e Trabalho Relacionado**: capítulo em que se aborda o conceito de computação em Cloud, incluindo uma análise de fornecedores de serviços e a escolha de um em particular. Por último, descrevem-se plataformas ou sistemas com funcionalidades semelhantes àquele que se pretende vir a desenvolver, identificando possíveis pontos de melhoria ou inovação;

- **Desenvolvimento Frontend e Tecnologias Relevantes**: capítulo em que se tratam os conceitos fundamentais de desenvolvimento Web, passando depois para as melhores ferramentas e tecnologias nesta área. São ainda mencionadas outras tecnologias relevantes para o projeto desenvolvido, abordando genericamente a forma como foram usadas;

- **Modelação da Plataforma**: capítulo em que se enuncia a metodologia de trabalho e os passos necessários para o desenvolvimento do projeto. Para tal, faz-se o

levantamento dos requisitos funcionais e *User Stories*. Por outro lado, discute-se a forma como a tecnologia abordada nos capítulos anteriores se pode interligar para formar a arquitetura global do projeto, terminando depois com a exemplificação de funcionalidades no sistema;

- **Detalhes de Implementação**: capítulo que apresenta detalhes de implementação no que toca ao *backend*, *frontend* e outras tecnologias usadas. Abordam-se casos concretos de uso e cuidados tomados no design e na implementação do sistema, acabando com uma visão global do mesmo;

- **Avaliação**: capítulo em que se analisa a forma como o sistema foi avaliado, incluindo resultados obtidos em testes de utilização e desempenho. Discute-se também a forma como os resultados foram interpretados e algumas conclusões que podem ser tomadas em relação ao trabalho desenvolvido;

- **Conclusões e Trabalho Futuro**: capítulo final onde é feita a conclusão da dissertação e se aborda o projeto subjacente, referindo alguns pontos importantes com ele relacionados. Por fim, são apresentadas algumas temáticas de trabalho futuro, indicando possíveis melhorias do sistema.

2

Estado da Arte em Aprendizagem Automática

No seguimento do capítulo introdutório, esta dissertação baseia-se no desenvolvimento de uma plataforma para auxiliar o treino de modelos de aprendizagem automática para aplicação na Agricultura. Atualmente, existe tecnologia que permite fazer tarefas relacionadas, nomeadamente ferramentas capazes de realizar atividades de aprendizagem automática.

Não obstante, um dos objetivos desta dissertação e da plataforma subjacente é estudar o uso das tecnologias mais recentes da área de Inteligência Artificial. Para tal, é necessário partir de conceitos básicos e usar meios e serviços variados, passando depois a tecnologias e ferramentas mais sofisticadas.

Este capítulo faz o levantamento dos pressupostos teóricos por detrás das tecnologias relacionadas com aprendizagem automática e profunda, focando-se depois no que, dentro destas áreas, é mais relevante para a área da Agricultura. O resumo apresentado de seguida pretende guiar o leitor pelas bases de uma das tecnologias mais relevantes usadas na dissertação, abordando depois detalhes concretos daquelas que foram estudadas com maior ênfase nesta dissertação.

2.1 Aprendizagem Automática

2.1.1 Introdução à Aprendizagem Automática

A evolução tecnológica trouxe um grande aumento na quantidade de dados criados, quer diretamente por utilizadores, quer automaticamente por sistemas informáticos. Este tipo de dados pode encaixar-se em três categorias principais: estruturados, em que os dados estão num formato standardizado, com uma estrutura definida, obedecendo a um modelo de dados ordenado e acessível facilmente por humanos e máquinas [64]; semi-estruturados, em que os dados não estão inseridos num formato convencional nem obedecem a um esquema fixo, mas existem certos elementos estruturais como etiquetas ou metadados que os tornam mais fáceis de analisar [64]; não estruturados, aqueles que não obedecem

a um determinado esquema ou formato e que, portanto, não podem ser guardados em bases de dados relacionais convencionais. Este é o tipo de dados que mais se produz atualmente, incluindo determinados tipos de documentos, conteúdos de mensagens ou emails, imagens, vídeos ou ficheiros de áudio [64].

Para analisar e tentar compreender vários aspetos nestas grandes quantidade de dados, particularmente do tipo não estruturado, foram surgindo diversas técnicas e mecanismos, dos quais alguns dos melhores e mais consensuais são aqueles baseados na Aprendizagem Automática. Este é um ramo da Inteligência Artificial que estuda sistemas que possam aperfeiçoar-se a partir dos dados de forma automática, sem que sejam explicitamente programados para tal. Em termos genéricos, esta área usa e cria algoritmos que possam usar dados, analisá-los e gerar um resultado. Este processo envolve a repetição de etapas de treino e previsão, capazes de se aprimorar continuamente para melhorar o desempenho ao longo do tempo [4].

2.1.2 Tipos e Tarefas de Aprendizagem Automática

Os algoritmos e técnicas da Aprendizagem Automática foram e continuam a ser desenvolvidas de acordo com o tipo de dados e as tarefas que se têm em vista. Combinações de dados e problemas diferentes requerem abordagens e metodologias particulares. Assim, o tipo de aprendizagem encaixar-se-á numa das seguintes cinco categorias:

- **Supervisionada (*Supervised*)**: os dados de treino são conhecidos e etiquetados, havendo informação e metadados concretos para a sua análise. Estes conhecimentos são usados para que o algoritmo possa inferir uma função que mapeie os dados de entrada num resultado de saída. As duas principais tarefas supervisionadas são a classificação, que separa e identifica os dados de acordo com etiquetas classificativas, e a regressão, que ajusta uma função aos dados. Alguns algoritmos usados para problemas de classificação são as árvores de decisão, o *k-nearest neighbours*, as *redes de Bayes* e as redes neuronais. Já problemas de regressão costumam ser resolvidos através de regressões lineares, logísticas e polinomiais [64];

- **Não supervisionado (*Unsupervised*)**: os dados de treino não têm etiquetas concretas e são utilizados sem manipulação prévia, podendo estar num formato não estruturado. Os algoritmos de aprendizagem têm como principal objetivo a identificação de padrões sem recorrer a elementos identificativos expressos, como etiquetas ou outro tipo de anotação. As principais tarefas não supervisionadas são o *clustering* (agrupamento), a identificação de características, a redução dimensional ou a deteção de anomalias. Os principais algoritmos usados são os de agrupamento (como o *K-means*, *Fuzzy Means* ou o *clustering* hierárquico), o *Principal Component Analysis* ou o *DBSCAN* [64];

- **Semi-supervisionado (*Semi-Supervised*)**: combinação dos dois modelos anteriores, já que se usam dados com e sem etiquetas. Esta abordagem é particularmente

relevante já que os dados não anotados costumam ser muito mais frequentes que os anotados, abrindo possibilidade para que o treino do algoritmo beneficie de anotação existente e de uma maior quantidade de dados. As áreas de aplicação são assim parecidas às dos tipos anteriores, incluindo classificação, anotação automática ou deteção de fraude;

- **Por reforço (*Reinforcement*)**: os algoritmos fazem uso de agentes, análises de ambiente e medidas de performance para fazer uma avaliação automática da melhor decisão a tomar. A aprendizagem é feita através de recompensas e penalizações que incentivam o agente a atuar de forma a maximizar as recompensas e evitar os riscos. Os modelos baseados neste tipo de aprendizagem podem automatizar ou melhorar a eficiência de sistemas pré-existentes, nomeadamente na área dos jogos, condução não assistida ou robótica [64];

- **Auto-supervisionado (*Self-Supervised*)**: podendo ser visto como um sub-tipo de aprendizagem não supervisionada, os modelos/algoritmos usam maioritariamente dados não anotados, tendo como principal objetivo a reconstrução de valores de entrada (*input*) com base noutros valores de entrada. Por poderem ser treinados com conjuntos enormes de dados não anotados, e terem como objetivo previsões com base no contexto de dados existentes, os modelos são capazes de capturar vários tipos de padrões. Os modelos baseados neste tipo de aprendizagem estão no centro de alguns dos sistemas mais inovadores da atualidade, desempenhando tarefas como geração de texto, imagem ou som [11]. Os principais algoritmos usados estão relacionados com *Transformers* e *Encoders*, como é o caso dos *Large Language Models* (Grandes Modelos de Linguagem), de que são exemplo o BERT, os GPT-3 e GPT-4 ou o Gemini.

A categorização dos tipos de algoritmos e técnicas de aprendizagem enfatiza a necessidade de enquadrar a tarefa a desempenhar com os dados existentes, tendo em conta a sua quantidade, estruturação e informação relevante.

No contexto do projeto desenvolvido, houve uma prioridade natural para a aprendizagem supervisionada e semi-supervisionada, uma vez que existem anotações capazes de guiar os modelos durante a fase de treino. Já o tipo de dados usado é sobretudo não estruturado, nomeadamente imagens carregadas dinamicamente por parte dos utilizadores, cuja quantidade é variável no tempo, de acordo com a entrada de novas entidades no sistema.

No seguimento do exposto acerca dos principais tipos de aprendizagem, faz sentido apresentar também informação acerca de três tarefas que podem ser auxiliadas ou resolvidas com técnicas de aprendizagem automática, a saber:

- **Classificação**: definida como o processo de reconhecimento ou agrupamento através de etiquetas classificativas. O valor de saída (*output*) tomado por uma função de

classificação será as etiquetas com que se possa definir o valor de entrada (*input*). A título de exemplo, pode fazer-se a previsão de uma determinada etiqueta em imagens de frutos, ou seja, prever se uma determinada imagem apresenta uma maçã, uma pêra ou uma laranja.

A classificação pode ser dividida em 3 categorias: binária quando existem apenas duas classes como verdadeiro ou falso, podendo as classes ser vistas como uma determinada propriedade ou a representação de dois estados, um normal e outro anormal; multi-classe quando existem várias classes em vez de apenas duas, sendo escolhida uma determinada etiqueta de entre um conjunto maior de possibilidades; multi-etiqueta que surge como uma generalização da categoria anterior, mas é usada especificamente para os casos em que é apropriado associar mais do que uma etiqueta em simultâneo (já que as classes não são mutuamente exclusivas e podem ter uma estrutura hierárquica) [71].

- **Segmentação**: é um sub-ramo da visão computacional e de processamento de imagem que procura agrupar conjuntos de pixeis por similaridade. Desta forma, conseguem identificar-se determinadas regiões de uma imagem que respeitem determinadas características. Esta categoria é uma extensão da classificação, já que, para além da classificação em si, consegue localizar um determinado objeto e delimitar a sua fronteira [52]. Existem ainda dois tipos de segmentação: a segmentação semântica, que estabelece uma distinção consoante em classes determinadas, marcando todos os que sejam da mesma classe de igual forma e a segmentação de instância, aquela em que se faz uma distinção entre indivíduos concretos, ainda que alguns pertençam a uma mesma classe geral. Um exemplo do mencionado é apresentado na figura 2.1.

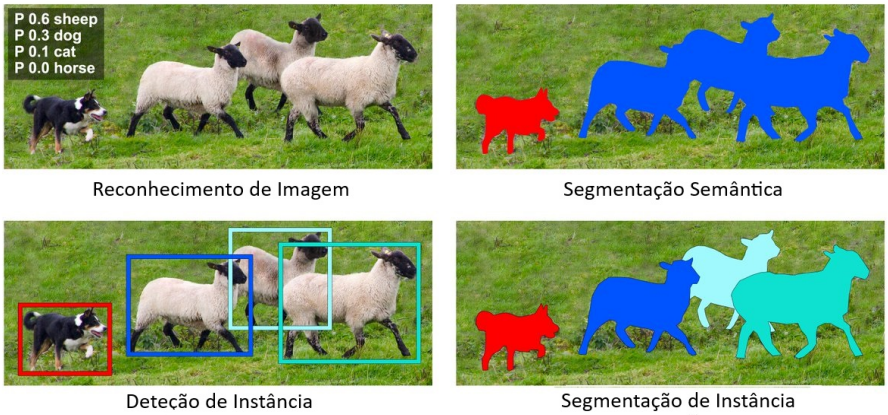


Figura 2.1: Tipos de Segmentação de Imagens (adaptado de [33])

- **Contagem**: é também um sub-ramo da visão computacional cujo objetivo é fazer uma contagem do número de ocorrências de um determinado objeto ou de uma determinada instância numa imagem [30]. Por exemplo, no contexto agrícola, pode

querer-se contar automaticamente o número de insetos em imagens de armadilhas. Existem várias técnicas de aprendizagem para resolver estes problemas, sobretudo explorando zonas de densidade nas imagens. Contudo, podem surgir dificuldades no processo, como os casos em que as instâncias não estejam espalhadas de forma uniforme, as imagens tenham distorções dependentes da localização e proximidade da câmara, a luminosidade seja má ou haja objetos em segundo plano ou semelhantes à instância a considerar [15].

Nos últimos anos, têm vindo a ser criadas técnicas para processar imagens e resolver os problemas referidos. As redes neuronais convolucionais são uma das técnicas que vieram revolucionar a forma como este processo se faz e a sua eficácia, permitindo um melhor processamento da imagem numa grande parte dos casos [25]. Não obstante, continua a ser necessária a ação humana para fazer uma contagem manual e anotar as imagens, o que acaba por ser uma tarefa complexa, morosa e sujeita a erros.

2.1.3 Processo de Aprendizagem

A aprendizagem automática é útil quando se tenta realizar uma tarefa em que se conhece conceptualmente o resultado correto, mas não se sabe exatamente o conjunto de passos para o obter. Por exemplo, o ser humano consegue distinguir pessoas inconscientemente olhando para fotografias, mas sem saber explicar exatamente as características que o levaram a fazer a distinção, já que estas podem ser apenas meros pormenores e não características gerais como a cor do cabelo ou dos olhos.

A aprendizagem automática vem resolver estas tarefas em que não é possível definir um algoritmo, mas há uma quantidade de dados suficiente para fazer uma boa aproximação. Um modelo de aprendizagem é precisamente um programa que consegue encontrar padrões e tomar decisões a partir de conjuntos de dados que ainda não tenha visto. No caso da aprendizagem supervisionada, os modelos são resultado de um processo de treino em que se guia repetidamente o programa a escolher os padrões certos para diferenciar os dados da forma pretendida. Os modelos são normalmente a implementação e instanciação de um algoritmo de aprendizagem, algo que pode ser definido como o processo matemático para encontrar padrões, normalmente resultado de métodos estatísticos e algébricos. [13]

No caso da aprendizagem supervisionada, os algoritmos costumam ter por base um conjunto de *features* (características) X, uma etiqueta Y e o objetivo é inferir uma função F, dependente de alguns parâmetros θ, que mapeie os valores de X em Y (2.1):

$$F(\theta, X) : X \rightarrow Y \tag{2.1}$$

Geralmente, o input do modelo é pré-processado, de forma a redefinir e uniformizar a escala dos dados para reduzir a variabilidade dos exemplos e facilitar a aprendizagem, num processo conhecido por normalização. Noutros casos, os valores de X são obtidos

na fase de pré-processamento, num método chamado de *feature extraction* (extração de características), que também pode ser feito recorrendo a outras técnicas de aprendizagem automática. Se a aprendizagem for bem sucedida, o modelo deverá conseguir prever o valor de Y para qualquer exemplo, ainda que não faça parte do mesmo conjunto de dados conhecido X (chamado de conjunto de treino), mas que respeite a sua distribuição de probabilidades.

O nome "aprendizagem supervisionada" é dado precisamente pelo facto de serem os valores conhecidos de Y a supervisionar o processo de aprendizagem, permitindo uma comparação empírica entre as novas previsões do modelo e o valor conhecido para cada caso. Esta comparação, conhecida como erro empírico ou erro de treino, é a medida de performance que permite perceber quão acertadas foram as previsões. Dependendo deste valor, pode ser necessário adaptar os parâmetros θ da função F, de modo a diminuir a diferença entre os valores previstos e os resultados reais, num processo chamado de treino.

Depois de definida, a função F pode ser usada para previsões noutro conjunto de dados ainda não utilizados (conjunto de teste) para medir o erro real fora do conjunto de treino. Se este erro for suficientemente baixo (o que por si só já é algo difícil de avaliar), pode dizer-se que o modelo consegue generalizar para além do conjunto de treino, tendo a capacidade de fazer previsões em relação a dados que ainda não tenha visto.

No entanto, caso o erro real seja significativamente maior que o de treino, pode ter ocorrido uma situação de *overfitting*. Neste caso, o modelo faz um sobre-ajustamento aos dados de treino, conseguindo valores de erro de treino muito baixos, mas perdendo a capacidade de generalização para outros conjunto de dados. Por outro lado, caso ambos os erros real e de treino sejam altos, o modelo não conseguiu ajustar-se aos dados de treino, sendo impossível instanciar qualquer hipótese que reflita a relação entre os atributos X e a(s) etiqueta(s) Y, num fenómeno chamado de *underfitting*. Assim, qualquer algoritmo ou técnica de aprendizagem supervisionada requer um equilíbrio entre a aproximação excessiva aos dados de treino ou a perda de capacidade de previsão em geral, incluindo para os de treino [4].

2.1.3.1 Regressão Linear e Polinomial

Para exemplificar os conceitos referidos, importa falar num dos algoritmos de aprendizagem mais conhecido. A regressão linear procura encontrar uma equação linear que minimize a soma das diferenças ao quadrado entre os valores previstos e os valores reais. Para exemplificar, pode imaginar-se um plano bidimensional (com coordenadas X e Y) com um conjunto de pontos. O objetivo será encontrar uma função que diga onde se irá localizar um próximo ponto, adicionado segundo a mesma distribuição de probabilidade dos restantes. Contudo, esta abordagem é muito limitada, sendo incapaz de se adaptar a dados que não estejam distribuídos de forma linear [25].

A regressão polinomial é uma extensão da anterior, permitindo uma equação polinomial capaz de se ajustar melhor aos pontos do plano. O treino é feito segundo o mesmo

mecanismo, mas, neste caso, o polinómio obtido é capaz de modelar uma relação não linear entre as variáveis (X a variável independente e Y a variável dependente). Esta não linearidade está intimamente ligada ao grau do polinómio, determinando a complexidade da curvatura da sua função no plano. Isto implica que, para valores maiores do grau, haverá também uma melhor adaptação ao conjunto de dados. Por outro lado, à medida que se aumenta o grau do polinómio, surge o risco de *overfiting*, indicado anteriormente, em que o polinómio se adapta em demasia aos dados de treino e perde a capacidade de generalização para outros conjuntos.

No entanto, um dos desafios associados à regressão polinomial reside na sua abordagem relativamente fixa à não linearidade. Isto significa que as capacidade para modelar a não linearidade estão limitadas às formas que os polinómios podem assumir. Assim, esta abordagem pode não ser suficiente para capturar relações não lineares no espaço original X, mas que poderiam ser lineares ou mais facilmente modeladas noutro espaço transformado X'. Para além do mais, as relações entre variáveis podem não ser apenas não lineares, mas também mudar de forma consoante o contexto, situações em que uma simples forma de não linearidade é incapaz de capturar corretamente relações intrinsecamente mais complexas. Outros algoritmos, como as redes neuronais, pretendem colmatar este problema, ao aprender representações mais complexas e flexíveis capazes de facilitar a inferência de relações entre os dados [13].

2.1.4 Mecanismos de Avaliação

A aprendizagem automática faz uso de conhecimento pré-existente para extrapolar ou prever algo noutros casos de estudo que ocorram em ambientes semelhantes. Como qualquer método de aprendizagem, é necessário errar para que se possa perceber o que melhorar no futuro, de forma a prevenir erros análogos. Quando aplicado a algoritmos e modelos de aprendizagem automática, este processo tem o nome de treino, e representa uma das fases mais importantes do seu desenvolvimento. Esta fase de treino carece da existência de medidas de performance capazes de avaliar o desempenho dos sistemas na fase de treino, mas também na fase de produção. As subsecções abaixo apresentam alguns dos mecanismos mais populares para o efeito.

2.1.4.1 Curvas de Treino ou Aprendizagem

Na secção 2.1.3 foram introduzidos os conceitos de erros de treino, de validação e de teste. Os dois primeiros são usados para medir e guiar o processo de treino, estando o último reservado para fazer uma estimativa final do desempenho do modelo. Estas três medidas podem ser usadas para desenhar curvas de treino ou aprendizagem, que refletem o erro ou uma percentagem de erro ao longo de uma certa medida de evolução, normalmente o número de treinos ou épocas. A figura 2.2 apresenta um exemplo do comportamento expectável das curvas de treino, relacionando o *Loss Error* com a evolução do treino para os conjuntos de treino e validação. Este tipo de medida de avaliação tem a grande vantagem

de permitir uma representação visual simples do processo de treino, útil para especialistas que tentem alterar os modelos com vista a melhorar a sua performance [73].

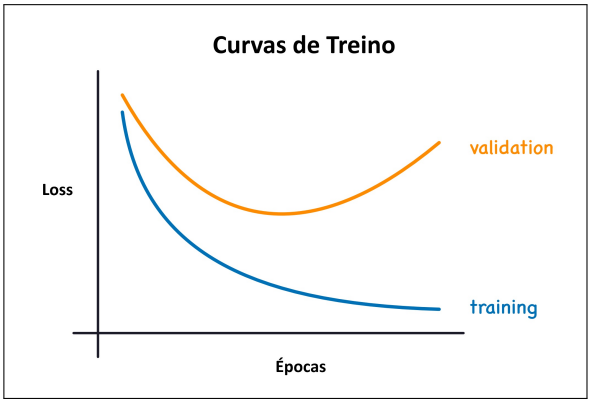


Figura 2.2: Curvas de Treino (adaptado de [77])

A progressão nos dados apresentados permite perceber não só a evolução do desempenho do modelo ao longo do treino, como também a sua capacidade de generalização para novos dados. Por exemplo, casos em que as curvas mostrem erros de treino notoriamente mais baixos que os de validação permitem considerar que o modelo está a sofrer um sobreajustamento aos dados de treino (*overfitting*), não sendo capaz de generalizar para conjuntos de dados independentes. Da mesma forma, casos em que as curvas mostrem erros de treino e validação convergentes e consistentes por um certo período de tempo podem ser sintoma de que o modelo está apto a generalizar para novos dados e que treino subsequente não trará melhorias significativas.

Importa ainda mencionar que, para além da fácil representação visual, outra das grandes vantagens das curvas de aprendizagem é a sua flexibilidade. Anteriormente, foram mostrados exemplos concretos em que a curva apresenta a evolução dos tipos de erros ao longo do treino. No entanto, é também possível usar outras métricas para desenhar este tipo de curvas, permitindo visualizar diferentes aspectos da performance do modelo durante a fase de treino. Algumas das métricas mais importantes são mencionadas na próxima subsecção.

2.1.4.2 Métricas de Avaliação Variadas

As medidas de perfomance para modelos de aprendizagem automática dependem da tarefa e do tipo de abordagem usada para os desenvolver, bem como dos critérios específicos usados para cada tipo de dados e cada modelo. Algumas destas medidas são [86]:

- *Log Loss*: também conhecida como perda logarítmica, é uma medida usada para avaliar a capacidade de classificadores binários. Por usar valores de probabilidade previstos, ao invés de apenas etiquetas (0 ou 1, por exemplo), esta métrica permite medir com maior precisão as divergências entre ambas, englobando a imprevisibilidade introduzida por usar valores concretos em lugar do valor das etiquetas.

Além do mais, por ter em conta a probabilidade dos resultados gerados, realça a importância não só do resultado estar correto, mas também de estar correto com maior confiança. Ao minimizar este valor, o modelo procura fazer previsões cada vez mais próximas dos valores reais;

- *Mean Square Error/Mean Absolute Error*: é habitualmente usado em problemas de regressão, medindo a média das diferenças entre os valores previstos e os reais. O erro quadrático, como o nome indica, multiplica o erro por si próprio, de forma a "personalizar" erros maiores;

- *Accuracy*: mede a proporção de resultados corretos face ao número total de previsões, pelo que é dedicada a problemas de aprendizagem supervisionada, nomeadamente os de classificação;

- *Precision*: mede a proporção entre o número de instâncias previstas como relevantes e o número real de instâncias relevantes;

- *Recall*: mede a proporção de instâncias corretamente classificadas como relevantes;

- *AUC-ROC*: abreviatura para *Area Under the Receiver Operating Characteristic Curve*, é uma métrica usada para problemas de classificação binária destinada a testar a capacidade para distinguir entre duas classes. Esta medida é apresentada como um gráfico que mostra a progressão dos valores comprovadamente positivos (*Recall*) face aos falsos positivos, ao longo de diferentes valores limiares;

- *F1-Score*: representa a média harmónica entre a Precisão e o *Recall*. Esta permite atribuir maior peso aos valores mais baixos das duas métricas, penalizando os desequilíbrios entre elas. Isto garante que apenas valores simultaneamente altos de ambas as métricas conseguem gerar um valor elevado de *F1-Score*;

- Matriz de Confusão: é a representação dos resultados através de uma tabela que inclui os verdadeiros e os falsos positivos, bem como os verdadeiros e os falsos negativos. Os verdadeiros representam previsões corretas, e os falsos previsões incorretas. No entanto, a diferenciação entre positivo e negativo permite perceber em que quadrante o modelo precisa de ser melhorado.

As métricas de avaliação apresentadas são apenas algumas das mais populares e usadas, sobretudo em relação a tarefas de aprendizagem supervisionada como é o caso da classificação, sendo as mais relevantes para esta dissertação. Por um lado, os erros quadrático e absoluto e a perda logarítmica são medidas geralmente usadas diretamente no treino, dando valores indicativos que o modelo deve reduzir de forma a aumentar a sua própria performance. Por outro lado, métricas como *Precision*, *Recall*, *F1-Score* ou *AUC-ROC* são usadas para abordagens mais pragmáticas, dando informações acerca dos modelos em fase de avaliação. Estas permitem resolver, por exemplo, situações em que os

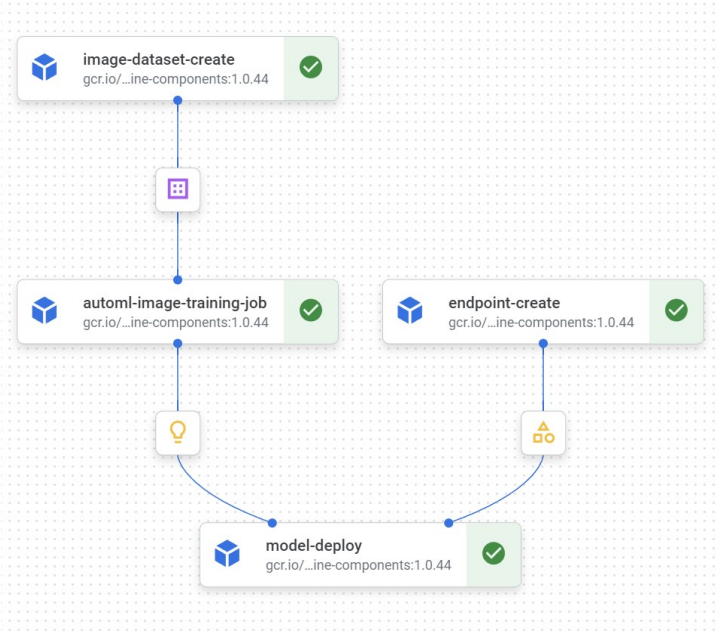


Figura 2.3: *Pipeline* de Aprendizagem Automática na *Google Vertex AI Pipelines*

conjuntos de dados são pouco balanceados, havendo uma quantidade muito mais significa de dados de determinadas classes em relação a outras. Esta diferença leva à necessidade de ter uma visão mais alargada dos resultados, de forma a equilibrar as diferenças de peso entre as diferentes classes, algo que uma medida objetiva como a *Accuracy* não permite.

2.1.5 *Pipelines* de Aprendizagem

Para concluir a secção acerca de aprendizagem automática, apresenta-se o conceito de *pipeline*, um dos mecanismos mais utilizados no desenvolvimento da plataforma abordada nesta dissertação. Um *pipeline* de aprendizagem é uma forma de automatizar a sequência de passos necessários para construir um modelo de aprendizagem automática. Assim, podemos vê-lo como um conjunto de passos que compõem o processo de aprendizagem, desde a extração e processamento dos dados, até ao treino e à implantação do modelo. Permitem compor as diferentes tarefas necessárias no processo de desenvolvimento, englobando quer os modelos individuais, quer o carregamento dos dados, numa abstração mais percetível e simples de usar [82].

Por outro lado, um *pipeline* também pode representar a divisão dos *workflows* de aprendizagem em módulos independentes e reutilizáveis de forma a serem ligados entre si para criar modelos mais complexos. Esta estrutura pode ser vista como um grafo em que se conectam diferentes partes do processo de aprendizagem, ou simplesmente como uma composição de passos puramente sequenciais, como é o caso do exemplo da figura 2.3. Este é um *pipeline* com 4 passos que foi treinado a partir da plataforma abordada nesta dissertação.

A importância desta abordagem é clara ao analisar o processo de aprendizagem normal, em que a extração, preparação e formatação dos dados, bem como a modelação, o treino

e a implantação do modelo fazem todas parte de um mesmo *script*. Isto implica correr o mesmo *workflow* várias vezes para várias versões do mesmo modelo, redundando na repetição dos mesmos passos iniciais de preparação dos dados. Da mesma forma, uma ligeira mudança nos dados ou na sua fonte pode implicar mudanças numa parte específica do código, que pode interferir indiretamente com outros pontos do processo.

Os *pipelines* vêm assim colmatar sobretudo problemas de escalabilidade dos modelos, permitindo chamar apenas certas componentes para lidar com um maior volume de dados em modelos iguais, reutilizar certas componentes de processamento para modelos diferentes ou concentrar as alterações de código numa única componente do processo, em lugar de alterar várias versões diferentes. Na mesma medida, vem permitir a contribuição de engenheiros especializados noutras sub-áreas diferentes para construir e estudar componentes específicas do processo. O seu trabalho pode ser direcionado para determinadas zonas independentes como processamento de dados, treino, avaliação ou implantação dos modelos [80].

2.2 Aprendizagem Profunda

2.2.1 Introdução à Aprendizagem Profunda

A Aprendizagem Profunda é um sub-ramo específico da aprendizagem automática em que são usados modelos de aprendizagem com um elevado número de camadas. Estas não são mais que níveis de abstração que aprendem diferentes representações intrincadas em grandes conjuntos de dados, usando a experiência e padrões reconhecidos em diferentes camadas para guiar a aprendizagem na direção certa. Estas técnicas tornam-se importantes à medida que os problemas se tornam mais complexos, quer a nível matemático, quer ao nível da dificuldade de análise de dados mais abstratos, volumosos e dispersos.

Tomemos como exemplo um classificador linear para separar duas classes usando uma regressão logística, cujo objetivo é traçar um hiperplano capaz de distinguir duas classes de pontos num plano cartesiano. Este tipo de classificador pode resultar caso as duas classes sejam linearmente separáveis, mas não em situações em que essa separabilidade não seja linear (aquelas em que não se consegue traçar uma linha reta para dividir os conjuntos) [39]. Exemplos deste tipo de conjuntos são apresentados na figura 2.4, havendo linhas preenchidas que tentam dividir linearmente o conjunto, e linhas picotadas que representam uma separação não linear hipotética dos mesmos.

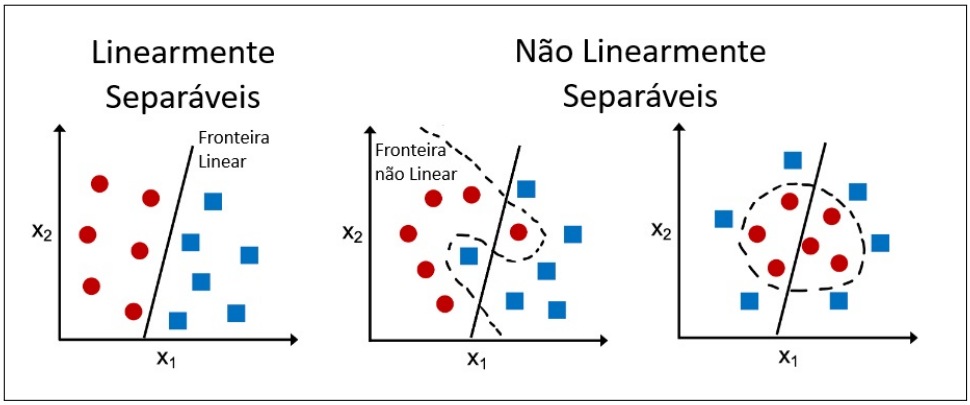


Figura 2.4: Separabilidade de pontos num plano cartesiano (Adaptado de [41])

Para lidar com situações como a apresentada no centro da imagem na figura 2.4, é possível remodelar os dados com uma transformação não linear, projetando-os num espaço dimensional superior. Neste novo espaço, a disposição dos dados é alterada, podendo revelar novos padrões de separabilidade linear e permitindo que um classificador linear os consiga separar corretamente (algo feito implicitamente pelos *Support Vector Machines* usando uma técnica chamada *Kernel Trick*, que estará fora do âmbito deste projeto), sendo capaz de traçar uma linha como aquela que é apresentada a picotado naquela mesma imagem. A estes classificadores dá-se então o nome de *Shallow Classifiers* [69].

Para dados mais complexos e abstratos, uma transformação não linear pode não ser suficiente, sendo necessária uma cadeia de transformações não lineares que operem sobre os resultados intermédios de cada transformação anterior. Por serem constituídos por várias camadas, estes classificadores são chamados de profundos. Ao contrário dos *shallow*, que aplicam um único conjunto limitado de transformações, os classificadores profundos aplicam várias séries de transformações cada vez mais complexas, capturando padrões em espaços dimensionais superiores [25].

Estas técnicas altamente não lineares tornaram-se particularmente úteis em conjuntos de dados não estruturados, revolucionando as áreas de processamento de imagem, vídeo, som ou texto. Em dados com este tipo de formato, as técnicas de aprendizagem automática tradicionais têm dificuldade em reconhecer a quantidade necessária de características relevantes. Este problema foi colmatado por técnicas mais profundas em que cada nível de abstração é capaz de criar uma representação única, focando aspetos diferentes e identificando detalhes cada vez mais "escondidos" nos dados [39]. Uma das técnicas mais bem sucedidas para executar este tipo de processamento são as redes neuronais profundas.

2.2.2 Redes Neuronais

As redes neuronais profundas são um dos principais modelos estudados e utilizados na aprendizagem profunda. O nome é inspirado na arquitetura do cérebro humano, cuja capacidade de reconhecimento de imagens, vídeos ou sons é muitas vezes superior

(ainda) à da tecnologia atual. O cérebro tem uma enorme rede de neurónios altamente interligados para conseguir realizar estas tarefas e espera-se que uma adaptação deste sistema em máquinas as possa fazer beneficiar das mesmas capacidades de análise. [4]

Por analogia, o *Perceptron* (Perceptrão) pode ser visto como um neurónio no cérebro, ou seja, o elemento base de processamento. De um ponto de vista matemático, o Perceptron pode ser representado como na imagem da figura 2.5.

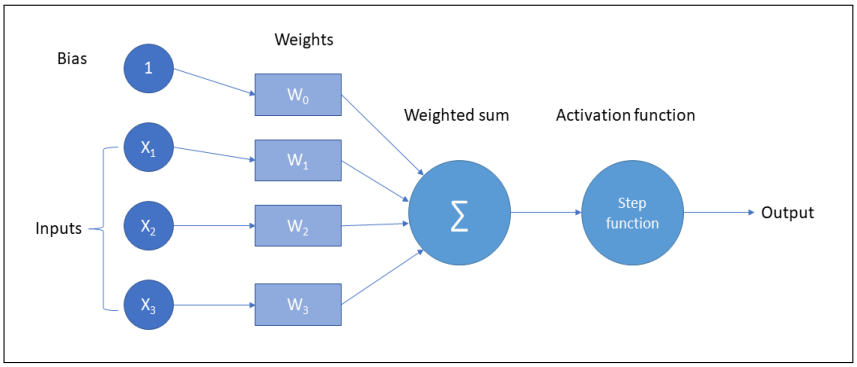


Figura 2.5: Representação visual de um *Perceptron* (Fonte: [72])

Os *inputs* (valores de entrada) podem fazer parte do ambiente ou ser o resultado do *output* (valor de saída) de outro tipo de processamento ou de outros perceptrões. Associado a cada *input* existem pesos de conexão ou sinápticos (noutra analogia ao cérebro), que determinam as suas importâncias relativas. Já o *output*, no caso mais simples, é uma soma pesada de cada um dos *inputs*, seguida da aplicação de uma função de ativação, numa operação descrita matematicamente como:

$$y = f\left(\sum_{i=1}^{n} W_i X_i + b\right) \tag{2.2}$$

Nesta equação, bem como na figura 2.5, W_0 é um valor que permite tornar o modelo mais geral, representando o peso de uma unidade de enviesamento adicional (X_0). Na aprendizagem supervisionada, o objetivo será aprender os melhores valores dos pesos (W) para que os *outputs* (y) sejam os desejados, treinando-se sucessivamente o modelo para obter os valores ótimos. Num exemplo simples, o treino pode ser feito atualizando os pesos do modelo lentamente, de forma a minimizar o erro quadrático entre a resposta e o objetivo, num processo que implica cálculos matemáticos como a derivada do erro em relação aos pesos.

Para tal, o perceptrão precisa ainda daquilo a que se chama uma *activation/threshold function* que, traduzida, significa função de ativação/limite. Como o nome indica, esta representa uma função matemática que define o valor de saída (*output*) do perceptrão de acordo com a soma pesada feita anteriormente [13]. No exemplo mais simples, é possível definir um hiperplano que separe o espaço em dois através do resultado, implementando um classificador linear que separe duas classes consoante o sinal (positivo ou negativo)

da soma. Desta forma, o valor de limite ou ativação é zero e representa o limiar a partir do qual o *output* se vai alterar.

No entanto, para casos mais complexos, em que o *output* possa variar entre uma gama maior de valores ou mesmo devolver uma probabilidade, podem e devem usar-se funções de ativação diferentes e mais complexas (como a sigmóide). Por ter um âmbito mais generalizado, estas estruturas passam a chamar-se de neurónios artificiais, referindo-se a qualquer função matemática que possa ser usada para imitar um neurónio biológico.

Uma rede neuronal não é mais que vários destes neurónios ligados entre si. Cada uma destas unidades básicas recebe o valor de saída de outras, processando-o e usando uma função de ativação para gerar o seu próprio valor de saída, que poderá depois ser passado a outro(s) neurónio(s) ou usado como parte do valor de saída da rede geral. Um exemplo de uma rede pode ser encontrado na figura 2.6, em que cada círculo representa um neurónio. O conjunto de ligações que se estabelecem entre estes permite formar camadas lógicas, que normalmente estão associadas a tarefas de processamento diferentes.

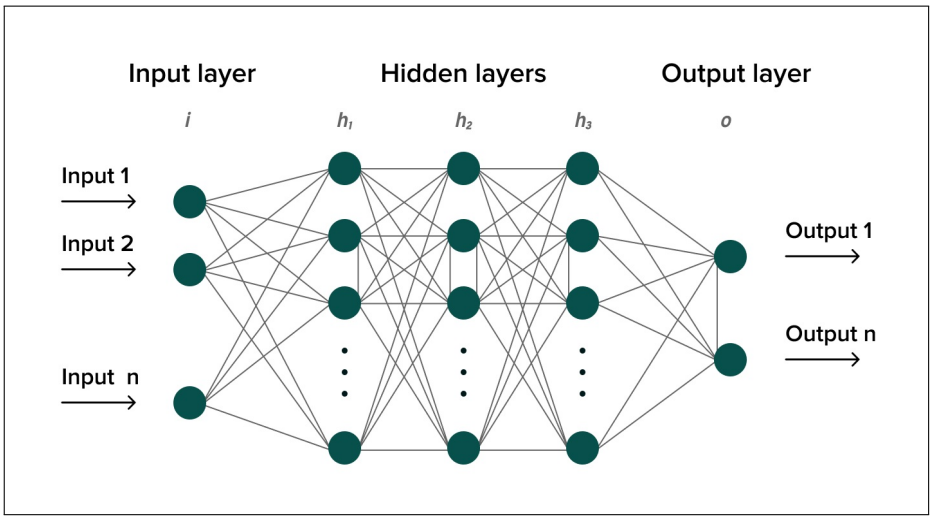


Figura 2.6: Exemplo de uma Rede Neuronal (Fonte: [42])

Geralmente, existem 3 conjuntos de camadas importantes: a camada de *input*, responsável por receber os dados e em que normalmente cada neurónio representa uma característica (*feature*); as camadas escondidas (*hidden*), responsáveis por receber os dados da camada de input, processá-los de determinadas formas e encaminhar os resultados para passos subsequentes e a camada de *output*, que produz o resultado final da rede, em que cada neurónio pode representar uma classe ou valor de *output*.

Dependendo do problema a resolver, as camadas e a forma como as ligações se estabelecem podem ser completamente reformuladas, podendo haver mais ou menos camadas, com um diferente número de neurónios em cada uma delas. Da mesma forma, as funções de ativação também podem variar entre neurónios (desde que, para cada camada, todos os neurónios tenham o mesmo tipo de função), nomeadamente na camada de *output*, que será responsável por devolver o(s) valor(es) da rede. Por exemplo, nesta

camada, pode usar-se a função sigmóide para problemas de classificação binária ou a função softmax para problemas de classificação multi-classe, responsável por converter um vetor de números reais numa distribuição de probabilidade. [25]

Ao contrário de outros algoritmos de aprendizagem automática tradicionais, as redes neuronais são capazes de aprender tipos diferentes de não-linearidade, permitindo automatizar o processo de obtenção de *features* que são depois passadas a modelos lineares. Este processo é possível através da aplicação de várias camadas que operem transformações não lineares sobre os dados, projetando-os num novo espaço [39]. Habitualmente, um grande número de camadas define uma maior profundidade da rede, alterando também o nome de rede neuronal para rede neuronal profunda (*Deep Neural Networks* (DNN)).

Relativamente ao processo de treino, o algoritmo *Stochastic Gradient Descent* (SGD) é um dos mais usados e populares. Este permite escolher uma amostra aleatória do conjunto de treino em cada passo e ajustar os pesos do modelo de modo a minimizar o gradiente, isto é, a diferença entre as previsões e os valores reais para aquele exemplo. Como o nome indica, o algoritmo pretende caminhar no sentido da minimização deste gradiente, obtido através de uma *loss function* (função de erro), como a *mean square error*, a *cross-entropy* ou a *log loss*.

Se para o caso de um simples neurónio se pode aplicar diretamente a função de erro entre o *output* e o valor alvo e calcular a sua derivada, no caso de uma rede neuronal o processo é mais complexo. Para tal, é usado o algoritmo de *Backpropagation*, responsável por calcular o gradiente do erro no que respeita aos parâmetros. Em termos simplistas, o *input* é propagado pela rede, sendo gerado um *output* a partir do qual se pode calcular o gradiente do erro relativamente ao alvo. Este tem de ser depois propagado para trás através da rede, usando derivadas parciais, para calcular o gradiente do erro relativamente aos parâmetros. O gradiente representa então o ritmo de mudança do erro para cada um dos parâmetros, sendo a base para os, de modo a reduzir o erro em iterações futuras usando o SGD. [13]

2.2.3 Redes Convolucionais

As redes neuronais convolucionais (Redes Neuronais Convolusionais (CNN)) são um dos mecanismos mais populares na área da visão computacional por serem especialmente desenhadas para o processamento de dados representados na forma de vetores. As imagens são um exemplo de um tipo de dados não estruturado que pode ser representado através de um vetor. Por exemplo, numa imagem de 10*10 píxeis, é possível representar todos os 100 píxeis num vetor. Para incluir as cores pode usar-se o sistema RGB (Red-Green-Blue), que permite representar as cores através dos três canais principais, com valores a variar entre 0 e 255, dependendo da intensidade. Para isto, a representação da imagem exemplificada pode ser feita através de um vetor unidimensional com 3 valores seguidos para cada pixel, ou 3 vetores unidimensionais para cada 100 píxeis, um para cada canal de cor principal.

Estes tipos de redes são semelhantes às DNN, mas usam convolução em pelo menos uma das suas camadas (em vez de multiplicações de matrizes tradicionais), numa estrutura que envolve não só camadas de convolução, mas também camadas de agrupamento (*pooling*). A função de convolução representa o integral do produto de duas funções em que uma é invertida e depois deslocada por um parâmetro t. Esta operação está na base matemática de uma operação mais complexa que permite a multiplicação da matriz representativa da imagem por outra mais pequena chamada de filtro ou *kernel*, segundo a equação definida em 2.3.

$$F(x, y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x + i, y + j) \cdot K(i, j) \tag{2.3}$$

Este filtro ou *kernel* pode ter diferentes dimensões ou valores e é deslocado sobre toda a matriz da imagem, multiplicando os seus valores pelos da parte da imagem coberta em cada iteração e somando-os. Este passo permite extrair características relevantes, gerando uma nova matriz denominada precisamente de mapa de características (*feature maps*). A esta é depois aplicada uma função de ativação não linear que facilita o reconhecimento de padrões (como a ReLU, uma função simples, mas que obtém bons resultados de forma rápida). Este sistema tira partido do facto de haver características correlacionadas em zonas próximas na imagem. Por outro lado, o uso dos mesmos pesos ao longo da matriz permite detetar padrões iguais em diferentes pontos da imagem, aproveitando o facto de grupos de características semelhantes poderem estar em mais que um local [39].

Se por um lado estas camadas são importantes para reconhecer padrões e características discriminatórias, é ainda necessário outro mecanismo complementar adicional. As camadas de agrupamento ajudam a preservar as características mais importantes de cada *feature map*, fundindo características semelhantes e reduzindo a dimensionalidade da imagem. Conseguem fazê-lo através de métodos como o *max-pooling*, em que se mantém o valor mais alto de cada *feature map*, que passa a estar disponível para uso em camadas subsequentes. [25]

As CNN são uma combinação destes estágios (convolução, não-linearidade e agrupamento), seguidos por um conjunto de camadas completamente conectadas (camadas densas). Estas últimas são responsáveis pela parte da classificação, processando o resultado das camadas anteriores e passando-o depois por uma função de ativação capaz de produzir as probabilidades de cada classe. [39].

2.2.4 Técnicas de Aprendizagem

Como já foi referido, as redes neuronais, especificamente as convolucionais, são muito úteis para o processamento de imagens. Estas são compostas por alguns dos conceitos fundamentais apresentados anteriormente, mas fazem uso de outras técnicas para melhorar a sua performance. Alguns dos métodos mais populares são o *Dropout* (desconexão de neurónios para introduzir aleatoriedade na rede) ou a *Batch Normalization* (normalização

das entradas das camadas para aceleração do treino). Além do mais, fazem também uso de otimizadores (*Adam*, *Adagrad* etc.) para melhorar a convergência do processo de treino e de técnicas de regularização (L2, L1 etc.) para evitar o problemas como *overfitting*.

Existem ainda outras técnicas de aprendizagem que importa mencionar. Os métodos de conjunto (como *Bagging* ou *Boosting*) são responsáveis por combinar vários modelos ou redes, com diferentes capacidades de previsão e atenção aos detalhes, de forma a reunir competências variadas e aumentar a robustez do modelo final. Por outro lado, as técnicas de *Data Augmentation* permitem gerar novos exemplos de treino a partir dos existentes (por exemplo, modificando as condições das imagens, a sua orientação, luminosidade, ou mesmo aplicando-lhes certas deformações), aumentando a quantidade de dados de treino e a generalização do modelo, ajudando na redução de problemas como o já mencionado *overfitting* [25].

Na área da segmentação, para além das CNN, existem outras redes como as FNC (*Fully Convolutional Networks*), que apenas usam camadas convolucionais. A segmentação semântica ou de instância também tem vindo a ganhar destaque, permitindo atribuir etiquetas a determinados píxeis, normalmente através de alterações nas CNN (por exemplo saltando camadas de forma estratégica). Por último, os *Conditional Random Fields* (CRF) conseguem modelar relações entre píxeis, podendo ser combinados com as CNN para melhorar a sua performance.

No âmbito desta secção, importa ainda referir um outro método de aprendizagem muito importante atualmente e que representa também uma técnica relevante nesta dissertação, o *Transfer Learning*.

2.2.4.1 *Transfer Learning*

Um dos principais problemas da aprendizagem supervisionada é a necessidade de uma grande quantidade de dados anotados explicitamente, sem os quais o treino pode não ser possível de todo ou então ficar altamente enviesado. Para ultrapassar este obstáculo, foram criadas técnicas de aprendizagem semi-supervisionada com o intuito de descobrir estruturas intrínsecas capazes de propagar a precisão das previsões em dados não anotados. Porém, muitos destes algoritmos ainda precisam de dados provenientes de fontes iguais (ou pelo menos gerados segundo distribuições de probabilidade semelhantes) e representados no mesmo espaço dimensional, tendo o mesmo número de *features* [57].

Para ajudar a resolver as dificuldades, surgiu a técnica de *Transfer Learning*, que, como o próprio nome indica, pretende utilizar dados ou conhecimento de domínios relacionados para ajudar os algoritmos a melhorar o seu desempenho no domínio de interesse. Esta técnica foca-se então na reutilização de modelos pré-treinados com uma quantidade enorme de dados de treino genéricos, cuja estrutura pode ser afinada usando dados concretos relacionados com o domínio de interesse. Para além do mais, usar apenas estes dados concretos disponíveis pode limitar a generalização dos modelos para outros domínios, que, apesar de muito semelhantes, possam acrescentar alguns pontos de diferença. [57]

Existem três tipos principais de "*Transfer Learning*":

- ***Fine-tuning***: significando afinação em português, esta abordagem permite partir de modelos pré-treinados e afiná-los iterativamente para que se adaptem melhor ao domínio de interesse, sendo uma abordagem especialmente útil quando se dispõe de poucos dados anotados. No caso das redes neuronais, preservam-se as primeiras camadas da rede, responsáveis pela aprendizagem de características mais genéricas e, portanto, comuns aos domínios de treino e interesse. Posteriormente, treinam-se as camadas finais, alterando a estrutura da rede para que esta se adapte melhor ao domínio de interesse;
- **Extração de Características**: como o nome indica, esta abordagem permite usar um modelo pré-treinado para extrair características nos dados. O objetivo é reutilizar as primeiras camadas da rede e usar o seu *output* para treinar um novo classificador, alterando as últimas camadas;
- **Adaptação de Domínio**: o objetivo desta técnica é adaptar um modelo pré-treinado com dados de um domínio relacionado ao de interesse, mas com algumas características diferentes. Assim, pode adaptar-se o modelo para que possa funcionar melhor nos dados do novo domínio de interesse, em vez de se adaptar em demasia aos do domínio de treino.

Para o caso do projeto desenvolvido, este tipo de abordagem tem especial interesse, já que é complicado construir um modelo de raiz com uma boa performance usando apenas os dados disponíveis (que, para além de serem em muito pouca quantidade, podem também não estar devidamente anotados). Esta técnica foi usada numa dissertação associada ao tema da Agricultura para desenvolver modelos de aprendizagem automática capazes de classificar espécies de plântulas infestantes [68]. Além de relevante do ponto de vista teórico, o estudo desta dissertação e código subjacente foi feito com o objetivo de adaptar um dos modelos resultantes para a plataforma desenvolvida.

2.2.4.2 *Auto Machine Learning*

A técnica de *Automated Machine Learning* (*AutoML*) tem vindo a mostrar resultados cada vez mais promissores na automatização do processo de desenvolvimento nesta área da aprendizagem automática. O *AutoML* pretende então determinar automaticamente a melhor estratégia, técnica e hiperparâmetros para resolver um determinado problema de aprendizagem. Desta forma, o sistema compõe todas as fases do processo de desenvolvimento de modelos, culminando na criação de um pipeline capaz de executar todas as tarefas, desde o pré-processamento dos dados até à fase de avaliação. Este tipo de solução torna a aprendizagem automática mais acessível para quem não tenha uma formação específica na área, por possibilitar a produção eficiente de modelos avançados com uma configuração mínima e um reduzido investimento de tempo e recursos [23].

A estratégia para decidir a melhor abordagem para cada tarefa ou tipo de aprendizagem deve ser baseada numa série de técnicas avançadas, responsáveis por usar vários tipos de métricas, análise estatística e algoritmos, a fim de explorar o espaço de soluções possíveis. Entre as técnicas iniciais mais bem sucedidas pode referir-se a *Automated Hyperparameter Optimization*, a *Meta-Learning* ou a *Neural Architecture Search*, embora se tenham vindo a desenvolver outras que tornam o regime de decisão mais abrangente e bem sucedido [53].

O facto de todo o processo de construção de modelos ser complexo, moroso e demorado é resultado direto do grande número de alternativas que podem ser usadas e do enorme grafo de dependências que pode resultar da sua composição. A figura 2.7 apresenta um esboço simplista deste processo, mas nem por isso mais fácil de compreender, onde são apresentadas algumas técnicas frequentemente utilizadas na construção de *pipelines* de aprendizagem. Também por isso, o *AutoML* tem vindo a ganhar relevância na automatização deste processo, fazendo com que grandes empresas como a Google, a Amazon e a Microsoft tenham passado a disponibilizar esta técnica nos seus serviços Cloud, simplificando a construção de modelos de aprendizagem automática [29].

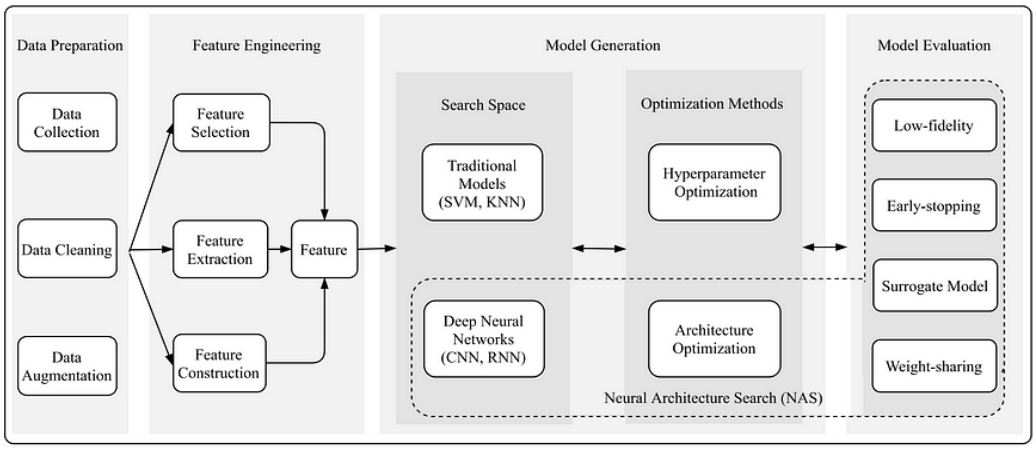


Figura 2.7: Visão de Geral de um Template para *Pipeline* AutoML (Fonte: [10])

O que foi referido anteriormente demonstra a utilidade desta técnica e o motivo pelo qual foi amplamente usada no projeto subjacente a esta dissertação. No capítulo seguinte far-se-á uma introdução à Cloud e às ferramentas usadas no âmbito do projeto, mas importa falar ainda nesta subsecção do uso do *Google AutoML* para treinar modelos de aprendizagem. Esta técnica conseguiu criar bons modelos de forma rápida e eficaz, permitindo visualizar resultados do treino e das métricas e avaliação associadas. Além do mais, permite disponibilizar a forma mais elementar de treinar modelos, destinada a utilizadores que apenas pretendam alterar alguns parâmetros de configuração e utilizar dados guardados na plataforma.

2.2.5 Aprendizagem na Agricultura

Atualmente existem várias aplicações e ferramentas de inteligência artificial que procuram resolver problemas de âmbito geral ou direcionadas para áreas específicas. Este projeto procura centrar-se na área da Agricultura em particular, desenvolvendo e utilizando ferramentas de aprendizagem automática otimizadas para este domínio. A análise de alguns dos métodos mais usados é feita em vários artigos mencionados ao longo do texto, mas importa lembrar que os casos de estudo são específicos para determinados problemas, pelo que os seus resultados podem não ser reproduzíveis no âmbito do projeto a desenvolver. Isto acontece porque as técnicas têm de ser adaptadas aos dados e às anotações (*Groundtruth*) disponíveis, tornando a análise de resultados mais complicada.

Particularmente no que toca ao processamento de imagens, especialmente na prospeção em culturas agrícolas, as técnicas que envolvem Redes Neuronais são muito eficazes. Dentro desta categoria, há que destacar a utilidade das CNN que, pelo facto de explorarem a profundidade e serem excelentes a lidar com imagens a cores, podem aumentar a performance dos modelos de aprendizagem. Nos casos em que existem poucos dados, a técnica de *transfer learning* mostra também uma taxa de sucesso elevada, já que faz uso de modelos complexos, treinados com uma quantidade de dados capaz de estabelecer uma base sólida de reconhecimento de padrões em vários domínios [67].

As técnicas de *Automated Machine Learning* (AutoML) (que permitem automatizar o processo de construção de modelos de aprendizagem em todas as suas vertentes, desde o processamento dos dados ao treino e à seleção de modelos) também se têm mostrado proveitosas nesta área [49]. Outras técnicas como os *Support Vector Machines*, úteis em tarefas de classificação, podem ser igualmente indicadas para determinados tipos de problemas de prospeção [12, 40].

Além do mais, são usados sistemas de Inteligência Artificial para fazer a gestão de recursos e otimizar a sua utilização, usando as técnicas de aprendizagem profunda referidas anteriormente para prever a necessidade de recursos como água e energia em função da fase de crescimento das plantações. A complementaridade dessas técnicas com a utilização de sensores capazes de reconhecer dados em tempo real contribui para maximizar os rendimentos ao mesmo tempo que minimiza o impacto ambiental, promovendo a agricultura de precisão [67].

Por outro lado, a adoção de *drones* equipados com câmaras de alta resolução, associados a algoritmos avançados de processamento de imagens, tem vindo a revolucionar o campo da prospeção e deteção de pragas. Aplicadas a grandes conjuntos de imagens, o uso das técnicas referidas (como as CNN e os *Support Vector Machine* (SVM)) são eficazes na identificação de problemas de larga escala, contribuindo para práticas agrícolas mais eficientes e demonstrando a importância que as novas tecnologias têm no campo da agricultura [62].

2.3 Conclusões

Este capítulo apresentou conceitos relevantes acerca de Aprendizagem Automática e Profunda, áreas que possibilitam as funcionalidades mais importantes do sistema desenvolvido. Neste sentido, foi necessário estudar, aprofundar e documentar aspetos teóricos relevantes desta área, desde os mais introdutórios ao estado da arte.

Por um lado, compreendeu-se que o desenvolvimento de modelos de aprendizagem automática envolve vários passos, cada qual com a sua relevância. Para além do uso de código especializado para o efeito, é necessário perceber as técnicas mais eficazes para completar todo o processo, desde o processamento dos dados à fase de implantação e invocação dos modelos. Para tal, a plataforma desenvolvida usa *pipelines* como forma de automatizar todo este processo, assegurando todos os passos auxiliares necessários para os conseguir executar.

Por outro lado, foi também usada a técnica de *AutoML* para construir modelos de aprendizagem automática com maior qualidade. Esta permite complementar uma série de técnicas relevantes para tornar o modelo mais eficiente e eficaz. Durante a fase de desenvolvimento foram ainda testadas outras técnicas, nomeadamente o uso de *Transfer Learning* para a extração de *features* e Redes Neuronais Densas ou *Support Vector Machine* para a afinação do modelo ao seu domínio de aplicabilidade.

Desta forma, o estudo dos conceitos apresentados permitiu compreender o estado da arte na área de Inteligência Artificial, de forma a escolher as técnicas mais eficazes para as tarefas a desempenhar, incluindo para a área da Agricultura. Para além do mais, assegurou uma base sólida para a compreensão de código e consequente capacidade de adaptação de modelos especializados para o sistema desenvolvido.

3

Desenvolvimento Cloud e Trabalho Relacionado

3.1 Desenvolvimento na Cloud

3.1.1 Introdução

Na sequência do capítulo acerca de aprendizagem automática, é importante abordar outro dos conceitos base do projeto subjacente a esta dissertação. A Computação na Cloud é uma peça fundamental no cenário tecnológico atual e transformou a forma de armazenamento, processamento e gestão de dados em diversos setores, incluindo na Agricultura[8].

Estas mudanças abriram caminho para inovações em várias áreas, incluindo na de aprendizagem automática, sendo uma tecnologia chave para alavancar os avanços tecnológicos com esta relacionados. Com todas as vantagens que lhe são inerentes, a Computação na Cloud forneceu uma base sólida para o desenvolvimento da plataforma em estudo, pelo que importa falar acerca dos seus conceitos básicos e da forma como estes se relacionam com este projeto.

Concretamente, os serviços Cloud foram usados para desenvolver o *backend* da plataforma, ou seja, a parte do servidor da aplicação, que esconde a lógica que permite ao utilizador interagir com a mesma corretamente. O *backend* trata de tudo o que envolve manipulação de dados do utilizador de forma segura, da lógica subjacente a pedidos mais complexos e da interação de tudo isto com outros serviços da Cloud. Normalmente, precisa de aceder a bases de dados ou serviços complementares para obter a informação necessária, acabando por ser uma "caixa negra" responsável por receber pedidos, processá-los e devolver uma resposta. Neste âmbito, este capítulo faz o levantamento dos pressupostos teóricos por detrás das tecnologias relacionadas com o desenvolvimento na Cloud, atalhando para escolhas que foram tomadas no que toca ao desenvolvimento da plataforma em estudo.

"Computação em Cloud refere-se tanto às aplicações que disponibilizam serviços na Internet como ao *hardware* e sistemas de *software* necessários em centros de dados para garantir esses serviços" [8]. Assim, estes centros de dados e os serviços que podem

disponibilizar são aquilo a que se chama a Cloud (Nuvem). Esta tem vindo a desenvolver-se cada vez mais nos últimos anos para responder ao aumento da procura e explorando três grandes fatores: o aumento do poder computacional a custo constante; o aumento da largura de banda na rede e as mudanças nos sistemas operativos, que passaram a permitir virtualização e, consequente, partilha de uma máquina por vários utilizadores em simultâneo [8]. A figura 3.1 apresenta um diagrama de barras que ilustra o crescimento vertiginoso dos gastos com infraestrutura Cloud, em oposição a uma aparente estagnação nos gastos com centros de dados tradicionais.

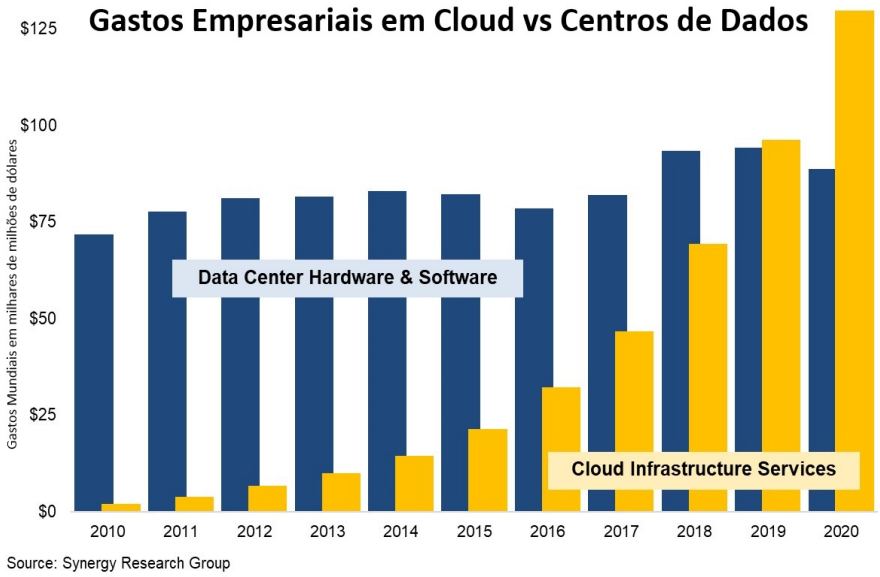


Figura 3.1: Crescimento do Mercado de Serviços Cloud na última década (Fonte: [51])

A Cloud garante recursos aplicacionais, servidores, armazenamento, ferramentas de desenvolvimento na rede e muito mais, tudo hospedado num centro de dados remoto gerido por um fornecedor de serviços em Cloud. Tudo isto garante a programadores e especialistas informáticos uma maior liberdade para desenvolver os seus produtos de forma personalizada, com o mínimo de preocupações de gestão ou resolução de problemas e sem interação direta com os fornecedores de serviços.

A computação em Cloud endereça várias questões de grande importância no âmbito programacional e de gestão de recursos. Em primeiro lugar, garante uma plataforma estável com grande disponibilidade, fiabilidade e tolerância a falhas, aspetos garantidos através de técnicas como replicação dos dados ou níveis de redundância, dividindo os pedidos e o armazenamento por mais que um centro de dados, potencialmente localizados em regiões geográficas completamente distintas. Todos os serviços disponibilizados estão protegidos por mecanismos de segurança apropriados para sistemas distribuídos e comunicações encriptadas, de forma a evitar ataques externos que causem violação de privacidade ou corrupção dos dados [37].

Por outro lado, a aparência de recursos computacionais infinitos sob procura permite

responder a períodos em que haja mais acessos aos recursos sem que seja preciso estudar e calendarizar a alocação dos mesmos consoante os períodos temporais. Da mesma forma, permite acompanhar o crescimento de uma aplicação, alocando mais recursos ao longo, consoante as necessidades.

Por último, os serviços têm custo eficiente, já que o pagamento é feito consoante os recursos que são usados ou reservados. Tendo em conta que, para certas aplicações e em certos casos, apenas é usado, em média, 10% do poder computacional dos servidores, o custo pode ser bem menor que a compra e manutenção de servidores independentes. Além do mais, a capacidade computacional escalável garante a possibilidade de resposta mesmo em picos de utilização, pagando apenas pelo que é de facto usado. Por outro lado, há um valor estratégico, já que se podem tomar opções de design sem o compromisso de compra expressa de máquinas, podendo usar mais ou deixar de usar de todo sem que se tenha incorrido em custos avultados [8].

3.1.2 Tipos de Serviços na Cloud

A programação em Cloud pode ser dividida em categorias de acordo com as características que apresenta. A ”Cloud pública” ocorre quando os serviços são disponibilizados para o público em geral, de forma a que apenas se pague o que se usa. Os recursos correm nos domínios do fornecedor, que também é responsável pela sua manutenção. Já o termo ”Cloud privada” é usado para referir centros de dados internos a determinadas organizações que não são disponibilizados para o público e são geridos pela organização em si ou por terceiros.

Os serviços na Cloud também podem ser catalogados de acordo com o tipo, havendo três categorias principais:

- ***Software as a Service*** **(Saas)**: o serviço corre numa infraestrutura Cloud, normalmente diretamente no *browser* do cliente e sem necessidade de downloads ou manutenção, que é deixada para os fornecedores do serviço. Esta categoria é de especial interesse quando não há tempo ou recursos para gerir ou desenvolver o *software*. Contudo, este tipo de serviços está muito ligado ao fornecedor, não permitindo customização da aplicação ou controlo na forma de guardar e lidar com os dados. Por outro lado, pode haver problemas na interoperabilidade e migração dos dados para tipos diferentes de serviços. São exemplos conhecidos deste tipo de serviço cloud a *Dropbox* ou o *Google Workspace*;

- ***Platform as a Service*** **(Paas)**: o consumidor pode implantar (*deploy*) a sua aplicação na infraestrutura cloud usando *frameworks*, serviços ou ferramentas disponibilizadas pelo fornecedor. Esta categoria foca-se na criação de software sem que o consumidor tenha de se preocupar com aspetos de baixo-nível como servidores, manutenção ou detalhes de armazenamento. Para além disto, este modelo reduz a quantidade de código, permite automatização das operações, garante fiabilidade e disponibilidade

do serviço e permite o uso de ferramentas de desenvolvimento ou teste. Apesar disso, pode haver problemas na integração dos dados noutros sistemas, demasiada dependência do fornecedor de serviços, poucas capacidades de customização ou limitações operacionais que não permitam determinados *workflows*/controlo por parte do consumidor. Exemplos mais conhecidos incluem *Google App Engine* ou *Microsoft Azure App Service*;

- ***Insfrastructure as a Service* (Iaas)**: o fornecedor de serviços é responsável por disponibilizar os recursos físicos (servidores, rede, sistemas operativos ou armazenamento) e capacidades de monitorização automatizadas onde o consumidor pode implantar *software* arbitrário. Estes recursos têm as mesmas capacidades que um centro de dados tradicional sem que o consumidor tenha de fazer a sua manutenção, sendo disponibilizados através de tecnologias de virtualização. O consumidor tem de gerir mais aspetos na aplicação, sistemas operativos, *middleware* e dados, delegando a gestão de servidores, discos, rede e armazenamento para o fornecedor. Alguns dos exemplos mais conhecidos são a *Amazon EC2* ou a *Google Compute Engine*.

  Entre as vantagens deste modelo estão a flexibilidade, controlo da infraestrutura, escalabilidade e dinamismo. Contudo, pode ser complicado integrar aplicações pré-existentes, comprar recursos e treinar desenvolvedores para gerir a infraestrutura, de forma a garantir que não haja erros de isolamento ou de segurança [61].

Para este projeto em específico, os modelos mais relevantes são o Paas e o Iaas, por serem aqueles em que é dado um maior grau de controlo ao desenvolvedor. Para poder responder aos requisitos funcionais, foi necessário um foque no detalhe e no controlo das funcionalidades que o Saas simplesmente não permite.

Ao programar com recurso e serviços disponibilizados por terceiros, é importante conhecer o que já existe e o que cada fornecedor oferece. Nas subsecções abaixo apresentam-se os principais fornecedores de serviços (Google, Amazon e Microsoft) e os pontos fortes de cada um, havendo uma subsecção adicional para discutir as diferenças e explicar a escolha que houve para este projeto em particular. Naturalmente, a limitação a estes três fornecedores acontece por serem de longe os mais populares, oferecendo as capacidades e ferramentas mais avançadas aos preços mais competitivos.

3.1.3 Google Cloud Platform

A Google Cloud Platform é um conjunto de serviços de computação em Cloud lançados em 2011 pela Google que corre nas mesmas infraestruturas que a empresa usa para as suas próprias aplicações e serviços (como *Google Search*, *Gmail* ou *Youtube*). A empresa detém ainda inúmeras infraestruturas onde outros serviços podem correr que, associados a um acompanhamento próximo da quantidade de recursos utilizados, permitem oferecer preços competitivos e que apenas dizem respeito ao que foi verdadeiramente usado [31].

Tratando-se de uma coleção de serviços, o utilizador pode usar alguma da infraestrutura da Google em si, incluindo requisição expressa de máquinas virtuais através da *Google Compute Engine* ou *Google Run*. A *Google Kubernetes Engine* permite usar *Cointainers*, uma forma mais flexível e moderna de virtualização que, em vez de recriar todo o servidor, consegue encapsular apenas os recursos que serão necessários. Esta abordagem evita a alocação de instâncias específicas dos recursos, baixando os custos e delegando ao serviço a capacidade para escolher o melhor ambiente para correr cada aplicação. Outros serviços como *Google App Engine* ou *Anthos* também agilizam o processo de implantação de aplicações, permitindo níveis de controlo diferentes no lado do servidor e integração com outros serviços [24].

O armazenamento de objetos e ficheiros é garantido através de serviços como a *Google Cloud Storage*, sendo a *Google Cloud Firebase* responsável por dados genéricos de menor dimensão. O serviço *BigQuery* implementa um sistema de bases de dados baseado em colunas, mais fácil de comprimir e indexar e com facilidade a escalar em proporção com o aumento do volume de dados.

No que toca à área de Aprendizagem Automática, a Google disponibiliza vários serviços que oferecem recursos para a construção e utilização de modelos de aprendizagem. É possível usar uma variedade de CPU e GPU de acordo com as necessidades e que garantem uma boa escalabilidade durante a fase de treino, flexibilidade e alguns dos algoritmos mais recentes e otimizados. A *Vertex AI* disponibiliza ferramentas para as várias funcionalidades que compõem um modelo de aprendizagem. No que toca à preparação dos dados, é possível carregar, gerir e anotar os dados de forma personalizada.

Em relação a modelos em si, é possível usar o *AutoML* de forma a não escrever o código expressamente e delegando para o sistema a escolha do melhor modelo, ou o *Custom Models* para código personalizado, permitindo a integração com *frameworks* e bibliotecas. É possível treinar ou implantar modelos pré-treinados e a *Feature Store* trata das questões de armazenamento. Existe ainda uma funcionalidade chamada *Deep Learning VM Images*, um conjunto de imagens de ambientes virtuais otimizadas para tarefas de aprendizagem profunda. É possível usar várias *frameworks* ou processadores para trabalhar com estas imagens, o que permite uma melhor otimização de todo o processo [34].

Existe também suporte para o uso de pipelines através da *AI Platform Pipelines* que disponibiliza uma plataforma para orquestrar os passos de *workflows* de aprendizagem, cujo processo fica facilitado pela integração simples com *frameworks* como a *Kubeflow Pipelines* ou a *TensorFlow Extended*(TFX), ambas open-source. A primeira é específica do conjunto de ferramentas geral *Kubeflow*, que permite correr modelos ou pipelines de aprendizagem em *Kubernetes*, possibilitando automatização, configuração e isolamento do processo [35].

3.1.4 Amazon Web Services

A Amazon foi a empresa pioneira no que toca à disponibilização de serviços na Cloud, lançando a Amazon Web Services no ano de 2006. Por ter entrado no ramo muito cedo, ainda num mercado com pouca competição, a Amazon conseguiu angariar muitos clientes e garantir a sua fidelização, mantendo-a através da constante inovação e pensamento no futuro. São ainda hoje o fornecedor de alguns dos serviços mais utilizados, tendo construído uma grande rede de parceiros e clientes importantes, além de uma enorme quantidade de serviços.

Como as restantes, a AWS oferece serviços de armazenamento, computação, rede e inteligência artificial, sendo possível usar diferentes níveis de abstração para armazenar os dados. Dentro dos serviços mais proeminentes estão a EC2, responsável pela disponibilização e gestão de máquinas virtuais, e a S3, um serviço de armazenamento altamente disponível e fiável. A AWS tem vários centros de dados, o que permite requisitar máquinas virtuais em diferentes locais e criar uma infraestrutura de serviços global, acessível de forma rápida a partir de qualquer ponto do globo [85].

Relativamente a funcionalidades relacionadas com Inteligência Artificial, a AWS oferece alguns serviços pré-treinados que podem ser integrados em novas plataformas. Da mesma forma, a *Amazon SageMaker* tem um conjunto variado de ferramentas para qualquer passo do processo de aprendizagem automática, permitindo opções de treino flexíveis e ajustáveis ao *workflow* [83]. Também permite o uso de *pipelines* de aprendizagem através da *Amazon SageMaker Pipelines*, com várias opções de personalização e implantação dos novos modelos [17].

3.1.5 Microsoft Azure

A Microsoft Azure é, como o nome indica, um serviço de computação em Cloud disponibilizado pela Microsoft especialmente importante para quem já depende de aplicações ou sistemas da Microsoft como o Office ou o próprio Windows. A Azure oferece várias ferramentas simples de usar, além de serviços facilmente integráveis entre si e com outros sistemas.

No que toca à infraestrutura, a Azure tem várias *availability zones* (zonas de disponibilidade) e vários centros de dados espalhados pelo mundo de forma estratégica para garantir proximidade ao cliente e resiliência, mesmo na presença de uma falha numa determinada região. Os pilares para a construção das aplicações assentam na computação, no armazenamento e na rede, sendo possível construir uma arquitetura personalizada usando o modelo IaaS e serviços como a *Azure Virtual Networks* ou a *Azure Disk Storage*. Por outro lado, é possível usar o modelo PaaS para uma arquitetura pré-definida e fácil de usar através de serviços como *Azure App Services* ou *Azure SQL*.

Os serviços disponibilizados pela Microsoft Azure incluem várias funcionalidades relacionadas com aprendizagem automática, entre as quais o uso de *pipelines* de aprendizagem, que automatizam o processo de construção e implantação dos modelos, dividindo-o

em vários passos distintos. Para tal, podem ser usados os *Azure Machine Learning components*, segmentos de código para um determinado passo ou funcionalidade dentro de um *workflow*. Estes têm metadados (nome, versão, tipo, etc.), interface (com especificações dos valores de entrada e saída, os seus tipos e valores pré-definidos) e comandos ou código necessário para correr o componente. [81]

3.1.6 Comparação e Escolha

Os fornecedores de serviços de computação em cloud apresentados nas subsecções anteriores têm os seus próprios pontos fortes e fracos. É possível fazer várias comparações entre estas, a vários níveis, algo que é apresentado nalguns artigos e estudos [60, 3].

No entanto, importa enquadrar as características de cada serviço com as necessidades da plataforma abordada neste documento. Em primeiro lugar, o seu desenvolvimento não prevê a intervenção de uma equipa, sendo o trabalho de investigação e programação desenvolvido por apenas um técnico. Em segundo, os requisitos de disponibilidade centram-se sobretudo em Portugal, pelo que as zonas de replicação e disponibilidade não desempenham um fator de decisão. Em terceiro, todas as plataformas apresentam serviços de aprendizagem automática, com integração rápida de novos modelos, suporte para *pipelines* e possibilidade de uso das *frameworks* mais populares na área. Por último, todas permitem integração com serviços de armazenamento e computação externos.

Assim, e tendo em conta que todas as plataformas providenciam as ferramentas necessárias para o projeto a desenvolver, decidiu optar-se pelos serviços da Google, que têm apresentado um bom ritmo de crescimento ao longo dos últimos anos. Apesar de serem o fornecedor mais recente, já têm provas dadas da fiabilidade dos seus produtos. Por outro lado, apresentam também os preços mais competitivos (ainda que por uma margem pequena), incluindo um crédito de 300 dólares que pode ser usado para experimentar os serviços de computação e aprendizagem durante três meses. Como fator adicional, esta é a plataforma com a qual havia maior familiaridade prévia, permitindo avançar mais rapidamente nas fases iniciais de desenvolvimento e saltando passos de aprendizagem dos serviços mais básicos como a *Google App Engine* ou a *Google Cloud Datastore*.

3.2 Tecnologias da Google Cloud

No seguimento da escolha da Google como fornecedor de serviços e da introdução feita à Google Cloud Platform em 3.1.3, importa detalhar os serviços Cloud usados na plataforma abordada nesta dissertação. Estando o mundo da programação em Cloud recheado de possibilidades, com inúmeras ferramentas e opções para o desenvolvimento de aplicações, foi importante escolher e agregar aquelas que permitissem cumprir os requerimentos pretendidos, sem comprometer a flexibilidade e simplicidade do sistema a desenvolver.

Assim, as subsecções que se seguem apresentam serviços concretos pertencentes à Cloud Platform, mencionando as suas características e abordando alguns aspetos em que

estes se relacionam com o projeto desenvolvido.

3.2.1 Google App Engine e Compute Engine

A Google App Engine e a Google Compute Engine são dois serviços de computação que vêm ao encontro de alguns dos conceitos abordados na secção 3.1.2 relacionados com PaaS e IaaS, respetivamente.

A App Engine é uma plataforma de computação *servless* que suporta o desenvolvimento e implantação de aplicações Web [7]. Por garantir uma infraestrutura gerida automaticamente pelo fornecedor de serviço, a App Engine é capaz de abstrair as complexidades subjacentes a tarefas de gestão e provisionamento de recursos. Tal garante não só maior liberdade ao programador, mas também a garantia de que a infraestrutura será fiável e escalável, colocando este tipo de serviço na categoria de PaaS.

É um serviço que suporta uma grande variedade de linguagens de programação, como Python, Java ou Node.js, garantindo versatilidade para vários tipos de projetos. Além do mais, a possibilidade de integração com outros serviços como a Storage, Datastore ou Vertex AI, associada a ferramentas de diagnóstico e controlo de versões, permite a construção de aplicações sofisticadas com o mínimo de configuração. Por último, a presença automática dos protocolos de rede mais avançados como SSL/TLS, bem como a integração com uma *firewall* específica, permitem que a aplicação e as comunicações estejam protegidas contra vários tipos de ataques informáticos.

Por outro lado, e contrariamente ao serviço já apresentado, o foco da Compute Engine é a customização e a flexibilidade no uso de recursos [16]. Se por um lado a App Engine gere automaticamente a maioria dos aspetos de infraestrutura, por outro a Compute Engine disponibiliza recursos computacionais que permitem correr código arbitrário. Esta funcionalidade é garantida através de sistemas de containerização suportados por máquinas virtuais, que permitem correr software existente sobre infraestrutura definida pelo utilizador, colocando este tipo de serviço na categoria de IaaS.

Por suportar esta capacidade de seleção e alocação de máquinas específicas, é possível construir um sistema mais flexível, que inclua máquinas virtuais com diferentes capacidades de processamento, memória e armazenamento. Esta flexibilidade permite configurar detalhadamente a infraestrutura no que toca ao sistema operativo, à capacidade do processador e às quantidades de memória persistente, RAM e gráfica, para melhor se enquadrarem nos requisitos necessários para cada serviço da aplicação. Além do mais, permite a migração para a Cloud de aplicações que estejam a correr sobre outros recursos. Esta alternativa beneficia de recursos pagos conforme a quantidade e tipo de instância usada, evitando investimento em máquinas e servidores próprios que têm necessariamente de ser geridos e mantidos.

As diferenças entre os serviços App Engine e Compute Engine são portanto evidentes, sendo o primeiro direcionado para desenvolvimento rápido e o segundo para a gestão direta da infraestrutura subjacente. Na entrada para o desenvolvimento da API que

compõe o *backend* da plataforma foi necessário ter em conta ambos os serviços, mas ambos acabaram por ser utilizados para a construção do sistema. A App Engine foi usada para a construção da API, enquanto a Compute Engine foi utilizada para um servidor adicional que suporta a execução automática de *templates* de código armazenados na plataforma. Desta forma, foi possível tirar vantagem dos pontos fortes de cada serviço, usando-os de forma complementar para satisfazer os requisitos do projeto no que toca a desempenho e funcionalidades.

3.2.2 Google Cloud Datastore

A *Google Cloud Datastore* é um serviço de bases de dados NoSQL oferecido pela Google focado na escalabilidade e robustez das opções de armazenamento para aplicações web [26]. Esta faz parte de um serviço mais abrangente chamado *Firebase* que tem dois modos distintos de utilização, o *Native Mode* e o *Datastore Mode*. O foco neste último modo é originado pelas próprias recomendações da Google para guardar dados em projetos de servidor, garantindo um elo de ligação entre o servidor e a aplicação no que toca ao armazenamento.

Por ser baseado num sistema NoSQL, a *Datastore* é capaz de armazenar dados de forma flexível e escalável, sem a necessidade de um esquema pré-definido como no caso das bases de dados relacionais. Isto permite o armazenamento de grandes quantidades de dados e consultas fáceis para a sua obtenção. Estas características, em conjunto com as capacidades de disponibilidade, replicação e mecanismos de transações e consistência dos dados, libertam o utilizador das preocupações de gestão da infraestrutura subjacente. O utilizador pode tomar decisões relacionadas com os modos de armazenamento, bem como configurar algumas opções de replicação e consistência, mas toda a manutenção é feita de forma automática pelo fornecedor de serviços.

Por último, a facilidade de integração com serviços como App Engine e Compute Engine permite associar os benefícios de ambos os serviços, garantindo sistemas robustos e fiáveis, quer no que toca a capacidade de resposta, quer no que toca à manutenção e gestão dos dados que farão parte dessas mesmas respostas. No âmbito do projeto subjacente a esta dissertação, a *Datastore* foi utilizada para guardar dados de utilizadores e sessões, que têm a necessidade de guardar atributos e dados de perfil. É também responsável por guardar dados referentes a templates de código, dos seus respetivos parâmetros e meta-informação acerca de conjuntos de dados para tarefas de aprendizagem automática. O serviço permite então aproveitar a capacidade para escalar à medida que o sistema cresce, mas também a facilidade de integração com o serviço App Engine, de forma a assegurar respostas rápidas aos pedidos de clientes.

3.2.3 Google Cloud Storage

No seguimento da subsecção anterior, foi apresentado que a *Datastore* está mais direcionada para maiores conjuntos de dados de menor dimensão, que sejam acedidos mais

frequentemente e precisem de escalar consoante o número de utilizadores e pedidos do sistema. Assim, é necessário a existência de um serviço complementar que consiga armazenar dados de maior dimensão, mas em menor número, como imagens e outro tipo de conteúdos estáticos. Este tipo de dados imutáveis e com formato arbitrário é denominado por objeto e está no centro de relevância de outro serviço chamado *Cloud Storage* [27].

Este é um serviço que garante boas propriedades de escalabilidade e performance, além de mecanismos de segurança que garantem a encriptação dos dados guardados e em trânsito. Através do conceito de *bucket*, é possível agrupar os objetos com base nas suas caraterísticas e configurar detalhadamente aspetos relacionados com controlo de acessos, localização, modo de replicação e ciclo de vida dos dados. Entre estes, é possível escolher classes de armazenamento como *Standard*, *Nearline*, *Coldline* ou *Archive*, como apresentado na figura 3.2. Este controlo permite escolher entre eficiência, latência ou redução de custos dependendo do tipo de dados a armazenar e das necessidades do sistema. O serviço beneficia ainda da gestão automática dos discos subjacentes por parte do fornecedor, garantindo que o utilizador não tem de se preocupar com a substituição de recursos em caso de falha.

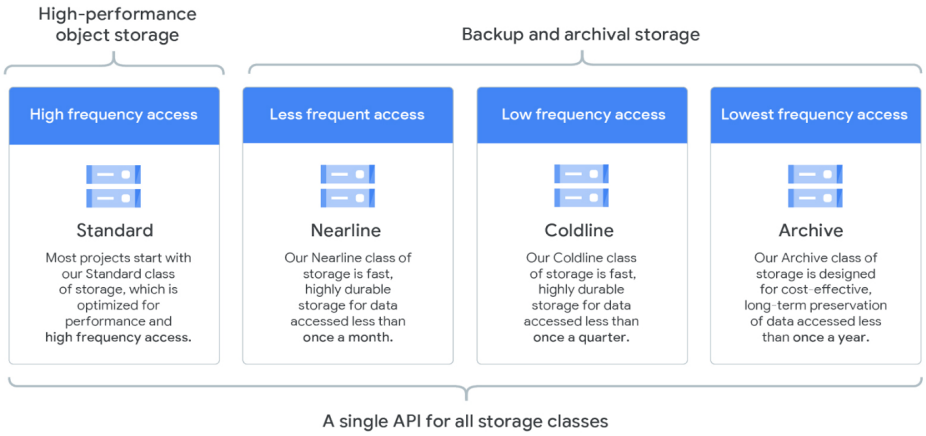


Figura 3.2: Classes de Armazenamento da *Google Cloud Storage* (fonte: [54])

A facilidade de integração da *Storage* com outros serviços da Google permite tirar partido de várias funcionalidades para a construção de aplicações mais robustas. No caso do projeto subjacente, este serviço foi combinado com o servidor implantado na *App Engine* para garantir o armazenamento de conteúdos multimédia, nomeadamente de imagens. Foram usados *buckets* diferentes para garantir não só uma melhor organização dos dados consoante os seus tipos, mas também a possibilidade de aplicar diferentes políticas de acesso dependendo das permissões e tipo de entidades que fazem os pedidos. Para além do mais, a *Storage* foi ainda usada para guardar anotações de imagens em conjuntos de dados e foi integrada com o serviço *Vertex AI* para armazenar dados relacionados com a criação de modelos de aprendizagem automática.

3.2.4 Google Vertex AI

A *Vertex AI* é uma plataforma da Google, integrada nos seus serviços Cloud, que pretende facilitar o desenvolvimento, gestão e implantação de modelos de aprendizagem automática. Este serviço unifica várias ferramentas relacionadas com inteligência artificial numa única interface, abrangendo todos os processos do ciclo de treino e implantação dos modelos de aprendizagem. Além do mais, é através desta interface que se pode ter acesso a mecanismos como *AutoML*, *Pipelines* e aos mais recentes modelos pré-treinados pela Google. Dentro destes, encontram-se alguns dos modelos generativos mais falados da atualidade como o *Gemini*, que é, segundo a própria Google, o melhor modelo generativo existente no mercado.

Além do ambiente capaz de facilitar o desenvolvimento, a plataforma oferece ainda suporte para as *frameworks* mais conhecidas como *TensorFlow*, *PyTorch* e *Scikit-Learn*. Associando isto à oferta de *Notebooks* e *Containers* especializados para o treino de modelos, o programador tem disponíveis virtualmente todas as opções para desenvolver modelos de aprendizagem automática. A facilidade de integração com outros serviços como a *Cloud Storage* e a *Compute Engine* permite ainda complementar várias ferramentas para utilizar dados, guardar informações do treino e disponibilizar os modelos.

No contexto do projeto subjacente à dissertação, a *Vertex AI* foi usada de várias formas. Por um lado, foi usada para testar ferramentas de *AutoML* diretamente na interface, treinando e invocando modelos com dados guardados em *buckets* da *Storage*, analisando custos e percebendo as funcionalidades disponíveis nesta modalidade. Por outro lado, foi usada para treinar modelos usando *pipelines* simples com código desenvolvido pelo autor e *pipelines AutoML* parametrizáveis através de adaptações de código pré-existente fornecido pela Google. O resultado deste treino originou modelos e métricas correspondentes, pelo que foi usada a API da *Vertex AI* para obter estes dados e invocar os modelos dinamicamente.

3.2.5 Google Artifact Registry

A *Google Artifact Registry* é uma ferramenta para guardar e gerir artefactos de software [9]. Num contexto informático, artefactos são definidos como o resultado da fase de desenvolvimento de software, ou seja, conjuntos de ficheiros que contenham código-fonte, documentação, bibliotecas ou outro tipo de dados gerados durante as fases de construção e implantação. Este serviço de *Registry* suporta vários tipos de formatos de artefactos, incluindo *Docker Images* ou *Maven Packages*, sendo suficientemente flexível para integrar vários tipos de software.

A principal vantagem deste tipo de serviço é a capacidade de gestão centralizada dos artefactos, ou seja, a possibilidade de gerir e configurar artefactos que possam ser usados noutros serviços da Google Cloud. Assim, é possível criar um único repositório de artefactos que podem ser usados por diferentes serviços, como é o caso da *App Engine*, *Compute*

Engine ou *Kuberneters Engine*. Este mecanismo providencia uma forma de criar repositórios de dependências e reutilizá-los de acordo com as necessidades, algo particularmente útil no desenvolvimento de sistemas baseados em microserviços. Além do mais, o facto de estar localizado na Cloud oferece melhores condições de segurança e escalabilidade, permitindo ainda controlo de acessos e de versões por parte do programador.

A relevância de artefactos é também evidente nos casos de aplicações containerizadas, onde é possível guardar de uma só vez todas as condições para a construção do sistema usando o sistema *Docker*. O conceito central deste mecanismo chama-se de *Image Container* e pode ser visto como um ambiente controlado para execução de software, incluindo para tal dependências como bibliotecas, *frameworks* e até o próprio código-fonte. Esta representa então a planta ou modelo que permite instanciar *Containers*, um pacote executável com tudo o que compõe a aplicação. As imagens são guardadas numa *Registry*, da qual a *Artifact Registry* é exemplo, podendo ser geridas e usadas conforme as necessidades. No caso do projeto desenvolvido, este sistema foi usado para guardar uma imagem de um *Docker Container* que contem código, dependências e variáveis de ambiente para executar um servidor que serve de complemento à API REST garantida pela *App Engine*.

3.2.6 Google IAM, Cloud Logging e Rede VPC

Para além dos serviços "principais" apresentados anteriormente, esta subsecção aborda outros serviços relevantes para o projeto, mas que não têm o mesmo tipo de importância. Os serviços que se seguem foram usados maioritariamente como forma de complementar funcionalidades ou requerimentos relacionados com o sistema.

3.2.6.1 Google IAM

O *IAM*, abreviatura de *Identity & Access Management*, que traduzido à letra significa "Identidade na Cloud e Gestão de Acesso", é um serviço que promove um conjunto de ferramentas para gerir acesso a recursos na Google Cloud [32]. Este serviço procura a boa conduta no acesso a outros serviços da Cloud, cuja má utilização pode ter efeitos devastadores, definindo de forma granular quem tem acesso a que recursos. Isto permite a administradores de projeto garantir que utilizadores/técnicos apenas possam aceder a recursos de acordo com o treino e formação que tenham para os usar. O serviço faz então uso de estratégias como a de menor privilégio, em que os utilizadores apenas têm acesso às permissões necessárias para completar as suas tarefas, de forma a garantir o uso correto do sistema.

Para além do mais, é ainda possível organizar pessoal em grupos e definir papéis específicos para aplicação automática de acessos ao longo da organização. A definição de contas de serviço permite perceber quem é responsável pelas ações dentro do sistema, além de garantir formas de acesso a recursos que estão limitados por definição. Foi precisamente nesta vertente que o serviço foi usado no projeto, para definir controlo de acessos a determinados recursos e permissões capazes de desbloquar funcionalidades

noutros serviços da Cloud. Entre esta utilização está o controlo de acessos a *Buckets* do *Cloud Storage*, a criação de contas de serviço e chaves de autenticação para acesso a funcionalidades na *Vertex AI* e a definição de permissões específicas para comunicação entre serviços Cloud diferentes.

3.2.6.2 Cloud Logging

A *Cloud Logging* faz parte do conjunto de ferramentas de gestão oferecidas pela Google para ajudar na monitorização de registos nos seus serviços. O seu principal objetivo é centralizar esta informação e permitir que os utilizadores/programadores possam ver registos de erros, avisos ou outro tipo de mensagens relacionadas com a execução de vários serviços. Esta monitorização é fundamental para diagnosticar problemas de programação, lógica ou dependências, de forma a melhorar a capacidade de resolução e o ambiente de desenvolvimento na Cloud.

No projeto desenvolvido, o *Cloud Logging* foi usado para monitorizar sobretudo registos relacionados com o servidor em execução na *App Engine*. Estes apresentavam vários elementos de interesse como dados do pedido, informações do cliente, estado e tempos de resposta, entre outros. Além do mais, o serviço foi usado várias vezes para diagnóstico de problemas relacionados com pedidos à API, vendo o tipo de resposta, mensagens de erros e exceções causadas pela lógica subjacente. Foi ainda usado nos testes de uso da *Vertex AI* para ver registos gerados em tempo real acerca de premissões de acesso e estado de execução de *Pipelines* de aprendizagem.

3.2.6.3 Rede VPC

A Rede VCP, sigla de *Virtual Private Cloud* (Rede Virtual na Cloud), é, como o nome indica, um serviço para criação e gestão de redes virtuais na Cloud. Este tipo de redes permite que as organizações providenciem e controlem endereços IP, sub-redes, portas de rede e outros aspetos relacionados com a receção e o redirecionamento de pedidos na rede. Isto permite controlar o tipo de acessos e monitorizar os clientes que tentam aceder aos pedidos, beneficiando das capacidades de segurança, flexibilidade e escalabilidade garantidas pela Cloud. Estas redes virtuais assemelham-se em tudo a redes tradicionais de centros de dados próprios, garantindo isolamento e segurança para os recursos instalados na Cloud.

O serviço é garantido automaticamente pela Google em vários dos serviços que impliquem o uso da rede, como é o caso da *App Engine* e da *Compute Engine*. No âmbito do projeto desenvolvido, a VCP foi usada para alterar de forma mínima algumas configurações pré-definidas na *firewall* de uma instância de servidor a correr na *Compute Engine*. Estas alterações foram feitas de forma a garantir que a porta 5000 não tinha restrições de acesso, facilitando a comunicação de outros serviços com o servidor em execução.

3.3 Trabalhos Relacionados

Esta secção apresenta trabalhos relacionados com a plataforma abordada nesta dissertação. Nas subsecções abaixo são apresentadas plataformas que usam ferramentas de aprendizagem automática para diversos fins, terminando com uma comparação entre elas e os possíveis pontos de melhoria.

3.3.1 Plataformas de Aprendizagem Automática

Nesta subsecção faz-se a análise e o levantamento das características de algumas plataformas estudadas e experimentadas durante a fase de investigação. Este estudo foi feito antes de iniciar o processo de desenvolvimento do projeto, pelo que algumas das funcionalidades podem estar desatualizadas, ou podem ter surgido novas plataformas mais aptas para os mesmos objetivos. Contudo, à entrada para a fase de planeamento e definição de objetivos, este estudo foi importante para saber o que o mercado disponibilizava em termos de serviços, custos e funcionalidades, ajudando a ter uma ideia das lacunas existentes e dos possíveis pontos de melhoria.

Relativamente ao estudo em si, é preciso ressalvar que os trabalhos aqui apresentados foram desenvolvidos num âmbito profissional, estando disponíveis para utilização por parte do público em geral. Desta forma, os serviços acabam por ter uma interface mais amigável, mas também um âmbito mais geral, não estando especificamente relacionados com domínios de interesse específicos. Existindo vários destes serviços disponíveis, a escolha dos que deveriam ser abordados teve com os seguintes critérios: facilidade para encontrar o serviço, uma vez que uma maior acessibilidade provavelmente indica um maior índice de utilização e popularidade; existência de características relacionadas com o trabalho a desenvolver, de forma a estudar pontos de melhoria ou encontrar ideias interessantes para adaptar; presença de diferenças com plataformas estudadas anteriormente, de forma a que o estudo abranja características e funcionalidades diferentes e complementares entre os diferentes sistemas.

3.3.1.1 Nyckle

A *Nyckle* [55] é uma plataforma com várias funcionalidades de inteligência artificial. Esta permite trabalhar sobretudo com três tipos de dados: imagens, texto e tabelas. Particularmente para as imagens, existem quatro funções: classificação, procura (retornando imagens com características inseridas em texto ou noutras imagens), deteção de objetos e extração de texto.

Particularmente quando usada a funcionalidade de classificação, podem-se acrescentar imagens para treinar o modelo, havendo uma funcionalidade automática de filtragem de formatos das imagens. Durante a fase de treino, a plataforma pede para anotar novamente certas imagens para as quais não haja concordância entre a etiqueta introduzida manualmente e a etiqueta prevista. A anotação é rápida, simples e intuitiva, mas não é

possível acrescentar mais que uma etiqueta de cada vez, dificultando o processo quando se querem incluir várias categorias hierárquicas. Depois do treino, é possível invocar o modelo, que tem várias funções de *output*, permitindo mostrar as probabilidades de cada imagem pertencer a qualquer uma das etiquetas presentes durante o treino. Um exemplo da listagem destas funções, que foram criadas e estudadas durante a dissertação, pode ser encontrado na figura 3.3, que apresenta a página correspondente da plataforma.

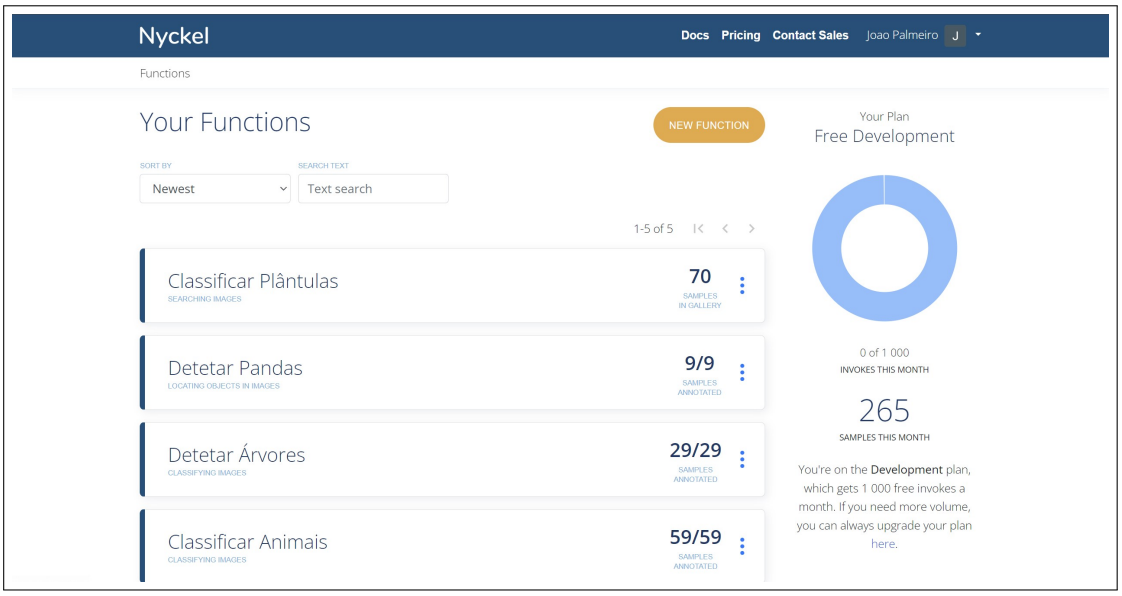


Figura 3.3: Secção das funções da Plataforma *Nyckle* [55]

Não há nenhuma funcionalidade particular de segmentação de imagens, embora isso seja provavelmente feito internamente pelo sistema. No entanto, a plataforma tem uma funcionalidade de deteção de objetos em imagens. Esta funcionalidade não segmenta a imagem e não consegue identificar o objeto inteiro, apenas consegue determinar se um determinado padrão se encontra naquela imagem.

A plataforma tem uma API pública detalhada com informação acerca dos tipos de dados, funcionalidades e os *endpoints* específicos a estes associados. Trata-se de uma API REST com os URL para acesso, bem como o tipo, *Headers*, *QueryParameters* e *Body* associados ao pedido, havendo ainda informação acerca das respostas esperadas para cada pedido, tudo devidamente documentado.

Noutras características interessantes, existe a possibilidade de catalogar os processos de aprendizagem para treinos diferentes com outras imagens, ficando organizados em secções diferentes. O design está atualizado e é intuitivo, sendo fácil usar o sistema, mesmo para utilizadores que não tenham conhecimentos técnicos da área de aprendizagem automática. Por outro lado, não existem funcionalidades de modelação específicas, sendo impossível escolher os modelos utilizados para o treino ou adaptar certos parâmetros para melhorar o desempenho dos mesmos.

3.3.1.2 SentiSight

A *SentiSight* [65] é uma plataforma de aprendizagem que pode ser usada numa vertente ligeiramente diferente, contendo mais funcionalidades de modelação no que respeita aos métodos de aprendizagem. A interface é pouco amigável para o utilizador e ainda mais confusa para quem não tenha conhecimentos na área.

Relativamente aos dados, tem uma forma de fazer o *upload* automático de uma pasta de imagens inteira, filtrando automaticamente aquelas cujos formatos não são suportados. É possível criar repositórios separados para treinos ou anotações diferentes, ficando guardados e acessíveis mais tarde. A parte de anotação não é a melhor, sendo preciso etiquetar uma imagem de cada vez. No entanto, durante o processo, é possível acrescentar mais que uma etiqueta de cada vez a cada imagem, acelerando o processo.

Esta plataforma é mais voltada para o programador/desenvolvedor. Tem parâmetros específicos para definir alguns aspetos do treino, nomeadamente o ritmo de aprendizagem, os tamanhos dos conjuntos de treino, validação e teste (em percentagem), etc. No entanto, não é possível escolher modelos em específico e os pré-existentes não parecem estar especialmente afinados, precisando de pelo menos 20 imagens com uma etiqueta para que esta possa ser incluída no treino. Cada treino tem um certo custo que varia de acordo com a quantidade de imagens e etiquetas usadas, num processo que pode demorar bastante tempo. Existe a funcionalidade de previsão multi-etiqueta e outra de segmentação de imagem. No entanto, o treino desta função é demorado (sendo este valor editável, mas nunca podendo ser menor que 60 minutos) e tem um custo financeiro relativamente elevado. A figura 3.4 apresenta o exemplo dos resultados do treino de um modelo de classificação de imagens na plataforma abordada.

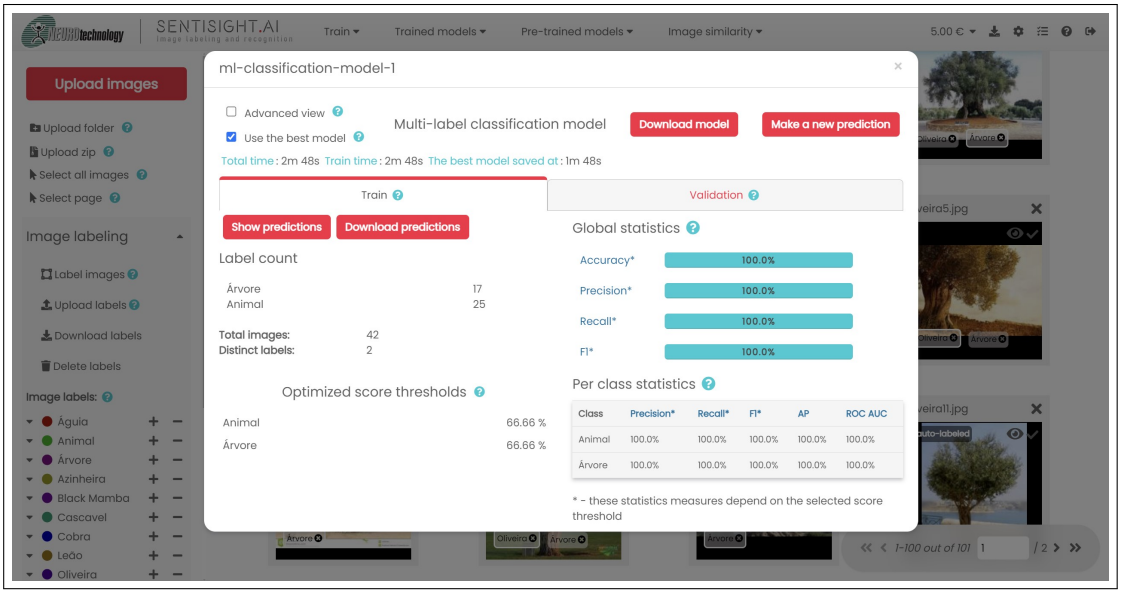


Figura 3.4: Resultados de um modelo de classificação na Plataforma *SentiSight* [65]

No que toca à documentação, existe uma quantidade elevada de tutoriais acerca da

melhor forma de usar cada funcionalidade, incluindo imagens ou vídeos do processo. Para além disso, é possível fazer os pedidos diretamente através da API REST sem utilizar diretamente a interface gráfica da plataforma, sendo incluídos na documentação segmentos de código para o fazer em várias linguagens. Existem também funcionalidades colaborativas, ainda que venham com um custo monetário acrescido, incluindo partilha de repositórios, capacidade de anotação de um mesmo repositório por mais do que uma pessoa, uso de modelos pré-treinados para determinados dados ou criação e treino de novos modelos.

3.3.1.3 Lobe

A Lobe [43] é uma aplicação voltada para os utilizadores que não tenham qualquer conhecimento técnico de informática, tendo uma interface muito amigável e simples de usar, além de uma documentação com explicações intuitivas do processo de aprendizagem. É uma aplicação que pode ser transferida para a máquina onde será usada e que tem a possibilidade de funcionamento *offline*. É também muito mais simples, tendo apenas funcionalidades de classificação de imagens.

Tal como as outras plataformas, permite colocar imagens e anotá-las manualmente, seguindo-se o uso de um modelo de treino desconhecido para prever automaticamente as etiquetas de novas imagens. É possível criar projetos diferentes para guardar determinadas imagens e fazer depois o treino de forma independente. A anotação é feita de forma mais expedita, tendo uma funcionalidade para selecionar várias imagens e anotá-las todas de uma vez, o que permite acelerar esse processo. Um exemplo desta anotação é apresentada na figura 3.5, apresentando uma divisão por etiquetas, e em que algumas das anotações já foram feitas de forma automática por um modelo treinado no sistema.

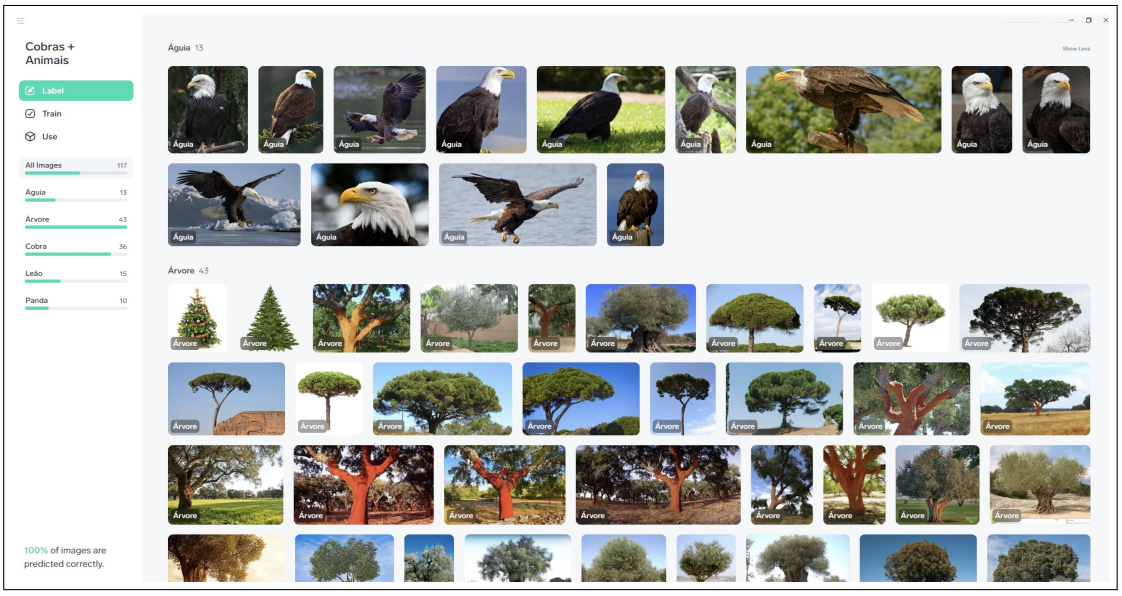


Figura 3.5: Anotação de imagens na aplicação *Lobe* [43]

Depois do treino do modelo, há uma nova previsão das etiquetas das imagens, sendo pedido ao utilizador para anotar novamente algumas etiquetas em que tenha havido uma discórdia entre a anotação manual e a previsão do modelo. Depois de ajustar (ou confirmar) as etiquetas, a plataforma reavalia e retreina o modelo, aumentando a percentagem de sucesso na atribuição de novas etiquetas e garantindo que novas chamadas à função retornam previsões com maior grau de certeza.

É possível exportar os modelos de cada projeto para que estes possam ser usados posteriormente, permitindo a sua integração em aplicações ou serviços externos. Contudo, durante toda a fase de treino, não é possível escolher um determinado modelo ou alterar qualquer tipo de parâmetros, sendo todos estes fatores escolhidos automaticamente pela plataforma e escondidos do utilizador. Por outro lado, a aplicação consegue fazer a análise de vídeo ou de uma sequência de imagens, incluindo em tempo real, tendo a capacidade de analisar vídeos captados através da câmara da máquina onde está instalada.

3.3.1.4 Outras Plataformas

A par das plataformas referidas anteriormente, foram estudadas outras com funcionalidades semelhantes ou complementares.

A Microsoft Vision é um serviço de inteligência artificial da Microsoft para analisar o conteúdo de imagens e vídeos que pode ser integrado no desenvolvimento de aplicações na Azure. À entrada para a dissertação, foi estudada uma API exemplificativa deste serviço, disponível com uma interface gráfica muito simples, onde é possível usar modelos/*pipelines* de aprendizagem pré-treinados e já preparados para responder a pedidos. Por se tratar apenas de um exemplo de utilização genérico, os modelos estão treinados para duas categorias específicas, sendo apenas possíveis invocá-los com imagens relacionadas. Assim, não é possível escolher o modelo a utilizar, embora seja possível editar certos parâmetros, quer relacionados com o modelo quer com o pedido em si.

A Apeer [6] é também uma plataforma de análise de imagens, com funcionalidades de segmentação garantidas a partir do uso de técnicas de aprendizagem profunda. Permite a criação e uso de *pipelines* de aprendizagem, com capacidade para usar estruturas pré-programadas de forma a formar um *workflow* funcional. Estas estruturas foram concebidas por especialistas diferentes, o que sugere que a plataforma permita a colaboração de vários programadores, centrados em áreas específicas. Cada módulo individual tem uma documentação que permite ao utilizador compreender as funcionalidades e requerimentos de cada um, tendo informação para decidir se este se adequa ao *pipeline* que pretende construir.

3.3.2 Comparação e Pontos de Melhoria

O estudo das plataformas mostrou que nenhuma delas é cumpre todos os objetivos a que esta dissertação se propunha, havendo pontos de melhoria em qualquer uma. Isto inclui funcionalidades complementares entre os sistemas, mas também áreas em que

nenhum é particularmente desenvolvido, nomeadamente no que toca à modularidade e transparência de parâmetros e tecnologias utilizadas.

Por um lado, embora exista documentação acerca da forma como são usadas as funcionalidades, acaba por haver pouca informação técnica e teórica acerca da forma como os modelos são desenvolvidos, dos parâmetros que usam ou das métricas de avaliação que os mesmos geram durante a fase de treino ou teste. Sendo o projeto subjacente a esta dissertação do ramo de engenharia, e esperando intervenção direta de técnicos informáticos e especialistas/investigadores na área de inteligência artificial, existe uma documentação mais exaustiva do ponto de vista técnico, incluindo detalhes dos modelos ou *pipelines* de aprendizagem. Esta informação é essencial para qualquer técnico que procure melhorar a qualidade dos modelos já presentes e saber os seus fundamentos na prática.

Por outro lado, nenhuma das plataformas é diretamente relacionada com a Agricultura, não havendo modelos especialmente aperfeiçoados para o tipo de problemas que estejam com ela relacionados. O trabalho desenvolvido vem colmatar estas falhas, permitindo incluir *pipelines* focados na resolução de problemas diretamente relacionados com esta área, parametrizáveis de acordo com as necessidades do utilizador.

Não obstante, apesar de haver suporte a algumas funcionalidades colaborativas, nomeadamente com partilha de repositórios de imagens, nenhum dos sistemas está realmente desenvolvido neste aspeto. Por um lado, todos tratam qualquer utilizador de forma indiferenciada, não distinguindo entre alguém que queira meramente adicionar fotografias, de outros que possam etiquetá-las ou alterar parâmetros de forma a afinar os modelos de aprendizagem. Por outro, não é permitida a intervenção simultânea de vários utilizadores num repositório, com etiquetagem das mesmas imagens ou correção de catalogações erradas.

Por último, apenas um dos sistemas tem alguns aspetos de modularidade, com capacidade para alterar parâmetros diretamente relacionados com o treino. Ainda assim, é impossível escolher realmente um *pipeline* em específico, alterar os parâmetros, ver métricas de avaliação ou adicionar novos modelos. Este é um aspeto que se vem resolver com o sistema apresentado na dissertação, garantindo mais modularidade para técnicos informáticos ou especialistas em IA que queiram adicionar e experimentar os seus modelos, permitindo depois que estes possam ser usados por outro tipo de utilizadores.

3.4 Conclusões

Este capítulo concentra uma parte importante do estudo prévio de algumas das tecnologias mais importantes para o desenvolvimento da plataforma abordada nesta dissertação. Além de introduzir as tecnologias e a sua aplicabilidade geral, oferece também uma visão de como foram utilizadas na fase de desenvolvimento.

Por outro lado, fez-se ainda o levantamento das características mais importantes de alguns dos sitemas estudados durante a investigação do estado da arte. A análise

destas plataformas e aplicações permitiu não só encontrar aspetos a adaptar, mas também identificar outros que poderiam ser melhorados. Desta forma, a secção anterior terminou precisamente com uma comparação dos sistemas e de possíveis pontos de melhoria.

Do exposto, é possível perceber que ainda existem alguns aspetos de inovação e melhoramento que o sistema desenvolvido vem endereçar. Este permite a intervenção de várias entidades, fornecendo uma base para futuras funcionalidades específicas relacionadas com cada entidade. Por outro lado, permite a inserção e anotação automática de conteúdo multimédia, que pode ser acedido por vários utilizadores, mas alterável apenas por moderadores da plataforma. Estes conteúdos podem ser usados para tarefas de aprendizagem automática relevantes.

Neste âmbito, especificações de *pipelines* de aprendizagem podem ser adicionadas e melhoradas por técnicos ou equipas de especialistas em inteligência artificial. A sua execução permite criar modelos parametrizados e as suas métricas de avaliação relevantes podem ser visualizadas através do sistema. Os detalhes de alta complexidade são delegados para a infraestrutura que serve de base à plataforma, baseada na execução de código e alojada na Cloud. Por último, o sistema desenvolvido serve ainda de base a funcionalidades futuras, bem como à adição de novos modelos e *templates* de código, permitindo que entidades interessadas possam intervir no sistema para o melhorar.

4

Desenvolvimento Frontend e Tecnologias Relevantes

No seguimento dos capítulos anteriores acerca de aprendizagem automática e tecnologias Cloud, que inclui também a análise de trabalhos relacionados com aquele que foi desenvolvido, importa abordar o estado da arte de tecnologias relacionadas com o desenvolvimento na Web. Estas permitiram desenvolver uma *interface* capaz de apresentar as funcionalidades da plataforma de uma forma mais amigável, servindo de complemento à API REST resultante da integração do *backend* com serviços Cloud.

Embora não tenha a mesma importância relativa que os conceitos abordados nesses capítulos, a interface (também denominada por *frontend*) faz parte do sistema de um ponto de vista geral, tendo implicado estudo e aprendizagem. Este capítulo faz então o levantamento de pressupostos teóricos relacionados com o desenvolvimento na Web, em particular das tecnologias mais relevantes usadas no projeto. Apresentam-se ainda casos de uso e imagens exemplificativas do *frontend* construído para o sistema, além de outras tecnologias relevantes para o desenvolvimento e testagem.

4.1 Desenvolvimento de Frontend

4.1.1 Introdução ao Desenvolvimento na Web

O desenvolvimento na Web é uma área da computação que abrange a construção e manutenção de *websites* e aplicações Web. Esta pode ser dividida em duas áreas principais, o *backend*, abordado no capítulo anterior, e o *frontend*, focado na construção de interfaces que garantam a interação do utilizador com a aplicação através do *browser* (navegador). Assim, o *frontend* disponibiliza as funcionalidades do sistema de uma forma que ajude o utilizador a interagir com o mesmo.

De um ponto de vista histórico, esta área começou com páginas estáticas simples, com pouca interatividade, sem conteúdo dinâmico e onde a estética não tinha um papel fulcral. Hoje em dia, com a massificação do uso da Internet e da utilização de *websites* para os mais variados ramos de atividade, a experiência do utilizador passou a ser cada vez mais

importante. Os *websites* passaram a ser cada vez mais complexos, interativos e atraentes, de forma a chamar a atenção dos utilizadores e promover o sucesso dos produtos digitais.

Esta evolução nos requerimentos de *websites* levou ao aparecimento de múltiplas tecnologias para incrementar e melhorar o desenvolvimento destes sistemas. Hoje em dia, tecnologias como HTML5, CSS3 e frameworks baseadas em Javascript oferecem as melhores ferramentas para o fazer, permitindo construir páginas web mais responsivas e adaptáveis a vários tipos de ecrã e dispositivo. Por outro lado, a integração de princípios de design e construção, bem como a necessidade de ter em conta questões de otimização e eficiência, tem vindo a aumentar a complexidade geral dos sistemas e a requerer uma maior aposta na formação de quem trabalha nesta área [14].

4.1.2 Fundamentos do Desenvolvimento Frontend

A evolução do desenvolvimento *frontend* tem vindo a ser impulsionada pelo aumento da capacidade dos computadores pessoais e, sobretudo, pelo desenvolvimento de *browsers* cada vez mais sofisticados. Estes são os sistemas mais usados para visualizar páginas web, permitindo usar um navegador para contactar e aceder a conteúdos alojados em cada parte do mundo através de URLs que apontam para destinos específicos na web. Estes conteúdos resultam da receção de código (na maioria dos casos *HyperText Markup Language* (HTML), *Cascading Style Sheets* (CSS) e Javascript) por parte do *browser*, que os interpreta de forma a gerar as páginas apresentadas, definir estruturas de controlo e autenticação, e garantir outros pedidos ao servidor com base nas ações do utilizador [19].

Atualmente, o desenvolvimento de *frontend* é baseado num conceito central, as *Single Page Application* (SPA), cuja tradução significa Aplicações de Página Única. Esta tecnologia está assente no carregamento de uma única página HTML que é atualizada dinamicamente consoante o estado do sistema e as ações do utilizador. Assim, é possível prevenir o carregamento de várias páginas inteiras do servidor, minimizando os tempos de espera e oferecendo uma experiência mais rápida e fluida [50].

Por outro lado, existem ainda vários princípios de design importantes no desenvolvimento destas páginas, tais como:

- **Simplicidade**: Minimizar elementos que não contribuam para a compreensão ou funcionalidade da aplicação;
- **Responsividade**: Assegurar o correto e rápido funcionamento em diferentes dispositivos e ecrãs;
- **Consistência**: Mantém um design uniforme em toda a aplicação, de forma a promover uma navegação mais simplificada;
- **Hierarquia Visual**: Fazer uso de elementos de design que permitam agrupar elementos visuais em hierarquias, indicando a importância relativa dos conteúdos.

Todos estes princípios, associados ao uso das SPA, compõem a espinha dorsal do desenvolvimento de *frontend* moderno, com o objetivo de promover a experiência do utilizador e maximizar o seu tempo de utilização dos sistemas.

4.1.3 Estado da Arte no Desenvolvimento Frontend

Como exposto nas subsecções anteriores, o desenvolvimento de *frontend* tem evoluído significativamente nos últimos anos de forma a poder dar resposta à crescente competição pela atenção dos utilizadores. Para contextualizar as tecnologias mais recentes e introduzir as linguagens de programação mais usadas na área, importa dar uma perspetiva histórica do desenvolvimento de *websites*.

Nos primórdios da web, o HTML permitiu a criação de páginas estáticas simples com texto e hiperligações. Por requerer que os estilos da página fossem aplicados diretamente no código-fonte, tornando-o confuso e difícil de manter, surgiu o CSS, que permite tornar o design mais flexível ao separar o conteúdo da apresentação. A necessidade de interatividade e dinamismo nas páginas levou depois ao aparecimento de uma nova linguagem, a Javascript, que se tornou rapidamente numa das linguagens mais usadas do mundo. Associada ao conceito de AJAX (*Asynchronous JavaScript and XML*), esta linguagem permitiu comunicações assíncronas com o servidor, tornando as aplicações web mais rápidas e responsivas [19].

Estas novas linguagens e métodos levaram ao surgimento das já mencionadas SPA que, em associação com melhores navegadores, abriram a oportunidade de criação de novas ferramentas de desenvolvimento na área de *frontend*. A necessidade de grandes conjuntos de código levou os profissionais da área a promover a criação de bibliotecas Javascripts que contém código pré-fabricado e pronto a usar para as mais diversas funcionalidades. Da mesma forma, grandes empresas criaram também as suas próprias *frameworks*, capazes de tirar partido não só de código, mas também de conceitos como *bidirectional data binding* (ligação de dados bidirecional), componentes e *virtual DOM*, com o objetivo de facilitar a construção das SPAs e o manuseamento dos dados e pedidos subjacentes [76].

Têm ainda vindo a ser desenvolvidos outros conceitos relevantes nesta área, tais como: *Progressive Web Applications* (PWA), que garantem uma base funcional que pode progredir à medida da capacidade dos dispositivos; Arquiteturas sem Servidor, em que se faz entrega de conteúdo estático pré-renderizado e interação com APIs JavaScript para colmatar a ausência de servidores web tradicionais e delegando a gestão do *backend* a serviços Cloud; CSS Moderno, com especificações recentes e mais poderosas como *CSS Grid* ou *Flexbox*.

Por último, o futuro do desenvolvimento *frontend* passa por mecanismos que sejam capazes de gerar código capaz de usar as tecnologias mais recentes e construir aplicações de forma automática. Com os recentes avanços na Inteligência Artificial, nomeadamente com o aumento das capacidades da IA Generativa, têm vindo a ser desenvolvidos modelos e aplicações capazes de gerar e alterar código automaticamente com base nos pedidos do programador/utilizador [46].

4.1.4 Angular, React e Vue.js

Com base na secção anterior, em que foram introduzidas tecnologias que compõem o estado da arte no desenvolvimento de aplicações Web, importa agora analisar as *frameworks* mais populares com maior detalhe. Estas fazem parte do motor da evolução recente do desenvolvimento de *frontend*, tendo sido desenvolvidas e utilizadas por grandes empresas na construção dos seus próprios *websites*.

A *framework Angular* começou a ser produzida em 2010 pela *Google*, sendo mantida pela mesma empresa até hoje [5]. Sendo focada no desenvolvimento de aplicações Web, usa o *TypeScript* como linguagem base, uma extensão do *Javascript* que adiciona tipagem estática e outras funcionalidades de mais alto nível. As principais características desta *framework* são a capacidade de criar SPAs de forma eficiente, um sistema de injeção de dependências e um ecossistema rico que permite a criação de componentes e serviços que promovem a separação e reutilização de código.

Já a *React* é essencialmente uma biblioteca de Javascript *open-source* que permite a construção de interfaces de utilizador, desenvolvida e mantida pela *Meta* [63]. É focada na criação e utilização de componentes reutilizáveis que garantem uma abordagem de programação mais simples e declarativa. Isto permite que o programador não tenha de se preocupar com a forma como os componentes são renderizados, mas sim com a forma como funcionam. Introduziu ainda o conceito de *Virtual DOM*, de forma a facilitar a atualização do navegador, e é conhecida pela sua flexibilidade, garantida pelo vasto ecossistema de bibliotecas complementares, como é o caso da *Redux*, focada no manuseamento de dados.

Por fim, a *framework Vue.js* [75] foi desenvolvida e projetada para construir interfaces de utilizador de forma progressiva, ou seja, procurando a conciliação de pequenos projetos com aplicações completas. Tal como as restantes, é capaz de usar ferramentas e bibliotecas de apoio, tentando procurar a simplicidade, flexibilidade e reatividade, de forma a tornar a aprendizagem mais fácil e as atualizações de interface mais rápidas de acordo com a mudança dos dados subjacentes.

Embora haja outras *frameworks*, como *Svelte*, *SolidJS*, *LitElement* e *Stencil*, as introduzidas nos parágrafos anteriores são as mais populares e utilizadas. Isso torna-as mais relevantes para o desenvolvimento do *frontend* deste projeto, uma vez que todas elas são suportadas por vasta documentação e por uma grande comunidade de desenvolvedores. A subsecção seguinte fará uma comparação técnica entre estas e justificará a escolha final tomada.

4.1.5 Comparação e Escolha

Para poder desenvolver o *frontend* deste projeto, foi necessário estudar as tecnologias disponíveis e decidir quais aquelas que melhor se adequam aos requerimentos. Na subsecção anterior foram introduzidas três das *frameworks* mais populares, que têm as suas qualidades e fraquezas.

Antes de passar à escolha de uma em particular, importa enumerar os aspetos mais

importantes para o desenvolvimento desta parte do sistema. Assim, por ordem de relevância, a *framework* deve apresentar as seguintes características: permitir o desenvolvimento de uma solução robusta e fiável, capaz de lidar com níveis de complexidade elevados; promover a padronização e reutilização de código, de forma a facilitar a manutenção e extensibilidade futuras; providenciar um ecossistema com variedade de ferramentas e bibliotecas, capazes de acelerar o desenvolvimento das funcionalidades.

Com base nesta enumeração, foram consultados artigos que comparam as características das *frameworks* e as suas respetivas vantagens e desvantagens [76] [38]. Tendo tudo isto em conta, e ainda que todas possibilitem as características mencionadas anteriormente, identificou-se a *framework Angular* como a que melhor se adequa às necessidades do projeto.

Esta é aquela que está mais desenvolvida em termos de robustez e capacidade de lidar com complexidade, sendo uma *framework* com bastante maturidade, mas que apresenta garantias de suporte a longo prazo. Oferece uma arquitetura completa para o desenvolvimento de aplicações Web como é o caso da *Angular CLI*, que permite automatizar a execução de comandos e instalação de componentes. Além do mais, a adoção do Typescript assegura a segurança do tipo de dados e a prevenção de erros em tempo de compilação.

4.2 Outras Tecnologias Relevantes

Para concluir a temática das tecnologias usadas na fase de desenvolvimento do projeto, é relevante abordar algumas tecnologias que não se encaixam diretamente nas categorias apresentadas nos capítulos anteriores. Desta forma, esta secção faz o levantamento de outras tecnologias utilizadas, introduzindo-as e explicando não só algumas das suas características, mas também a forma como se enquadram no desenvolvimento de aprendizagem automática, do *backend* e do *frontend*.

4.2.1 Linguagens de Programação

Na secção anterior foram apresentadas as linguagens Javascript, Typescript, HTML e CSS, muito relevantes no desenvolvimento do frontend da plataforma. Contudo, foram necessárias outras linguagens de programação para as tarefas relacionadas com o *backend* e com tarefas de aprendizagem automática. Estas são a Java e o Python, duas das linguagens mais populares e usadas atualmente, como mostra a figura abaixo 4.1.

As subsecções seguintes fazem o levantamento das suas características e das razões pelas quais foram escolhidas para as áreas mais importantes e relevantes do projeto.

4.2.1.1 Java

A Java é uma linguagem de programação orientada a objetos criada no início dos anos 90 e amplamente usada atualmente. O seu paradigma e filosofia de programação permitem tornar as aplicações mais flexíveis e transparentes entre dispositivos e sistemas operativos.

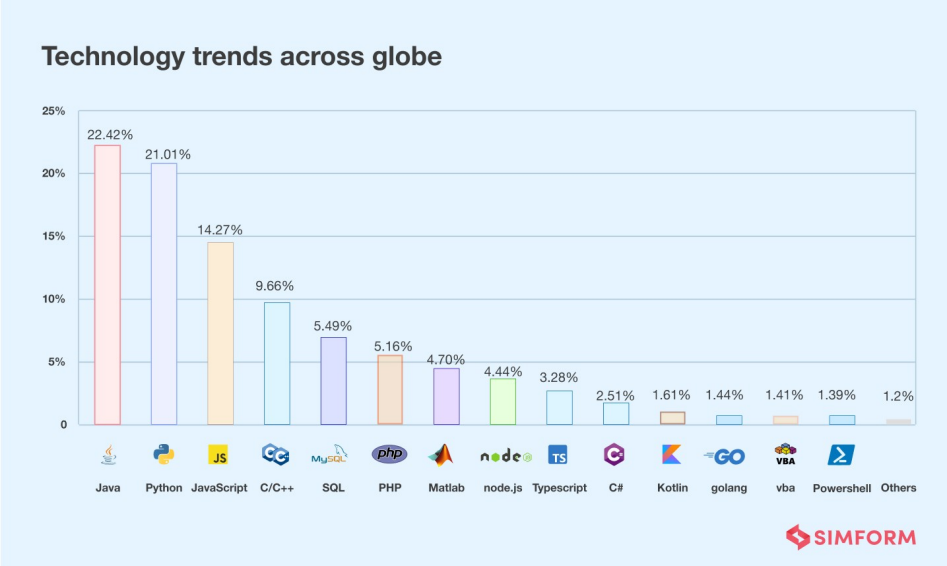


Figura 4.1: Tendências de Popularidade de linguagens de programação com base no número de procuras no ano de 2023 (fonte: [2])

A introdução do ambiente de execução *Java Runtime Environment* (JRE) e de uma máquina virtual própria *Java Virtual Machine* (JVM) facilitaram ainda o desenvolvimento de software robusto e seguro de uma forma eficiente e escalável [84].

No contexto do projeto, a Java foi a linguagem e ambiente de programação escolhido para desenvolver o *backend* da plataforma de acordo com o modelo Paas da App Engine. Esta escolha foi feita com base nas capacidades de robustez, segurança e flexibilidade apresentadas no capítulo anterior, mas também pela variedade de bibliotecas que a Google Cloud oferece para esta linguagem. A popularidade da linguagem, associada ao suporte oferecido pela Google, permite que haja várias amostras de código que puderam ser adaptadas para o projeto, bem como mais mecanismos de diagnóstico de problemas.

4.2.1.2 Python

A linguagem Python [59] é uma linguagem de mais alto nível e com uma filosofia que pretende assegurar a legibilidade do código e a flexibilidade de utilização em vários domínios. As suas extensas capacidades são garantidas pela sintaxe simples e flexível, bem como pelo suporte de vários paradigmas de programação (incluindo o orientado a objetos da Java). Destaca-se sobretudo na área da inteligência artificial pelo suporte de um grande número de bibliotecas e *frameworks* especializadas na área, que permitem o desenvolvimento de modelos de aprendizagem automática complexos com maior facilidade.

Relativamente ao projeto desenvolvido, esta linguagem foi usada para assegurar todas as funcionalidades relacionadas com o estudo, desenvolvimento e implementação de modelos e *pipelines* de aprendizagem automática. A sua utilização foi necessária para realizar as seguintes tarefas relacionadas com esta temática: experimentação de *pipelines* utilizando a biblioteca *ScikitLearn* e mecanismos de *Transfer Learning*; desenvolvimento de

código capaz de criar *pipelines* de aprendizagem na Google Vertex AI utilizando a *framework Kubeflow*; estudo de código desenvolvido por outros alunos do projeto *SmartFarm* para implantação dos seus modelos para a plataforma.

Noutro âmbito, a linguagem Python foi ainda usada para desenvolver um servidor Flask que permite a execução automática de código guardado na plataforma. Esta linguagem permitiu o uso das bibliotecas de inteligência artificial necessárias para que o código pudesse executar, incluindo aquelas que permitem criar *pipelines* na Google e interagir com os seus serviços.

4.2.2 Postman

O *Postman* [58] é uma das plataformas mais populares na área do desenvolvimento na Cloud e na Web. Permite facilitar a criação, partilha, testagem e documentação de APIs ao apresentar uma interface que permite visualizar e gerir requisitos de pedidos e respostas de forma intuitiva. Esta ferramenta apresenta várias funcionalidades para as tarefas mencionadas, como a criação de testes automáticos, simulação de *endpoints* ou definição detalhada variáveis de ambiente.

Em particular no âmbito deste projeto, o *Postman* permitiu agilizar o processo de desenvolvimento e especificação da API REST do *backend*, facilitando a substituição de *endpoints* e variáveis no corpo e no cabeçalho dos pedidos. Permitiu ainda simular os pedidos para essa mesma API, facilitando a validação das suas funcionalidades e a verificação de erros devolvidos. De um outro ponto de vista, esta ferramenta permitiu a organização de pedidos consoante as temáticas com os quais estavam mais relacionados (por exemplo, utilizadores, aprendizagem automática ou conjuntos de dados etc.) e a documentação da API, incluindo a definição do tipo de pedido e dos seus parâmetros de entrada e saída.

4.2.3 Flask

A *Flask* [21] é uma *framework* da linguagem Python para desenvolvimento web reconhecida pela simplicidade e flexibilidade. Estas caracetrísticas resultam da liberdade de implementação garantida aos desenvolvedores, permitindo-lhes escolher como implementar as funcionalidades das aplicações a desenvolver. Além do mais, a arquitetura tem um número mínimo de componentes para garantir o desenvolvimento de aplicações web, que podem ser complementados com extensões e bibliotecas externas. Esta *framework* é especialmente utilizada para desenvolver APIs REST que sustentem micro-serviços ou aplicações web num formato simples, mas eficiente.

No contexto do projeto, a *Flask* foi utilizada para desenvolver um servidor simples que permite o envio e execução automática de código Python. Desta feita, o servidor tem apenas dois *endpoints*, um para enviar e executar código e outro para receber os resultados da execução em tempo real, através da técnica *Server Sent Events* (SSE). O grande benefício no uso desta ferramenta passou pela simplicidade de implementação,

requerendo configuração mínima e poucas variáveis de ambiente para conseguir um servidor funcional. Outras bibliotecas foram apenas instaladas no ambiente de execução e importadas no código para satisfazer funcionalidades adicionais.

Importa ainda mencionar que o servidor desenvolvido foi colocado num container *Docker* (tecnologia apresentada na secção 3.2.5 do capítulo anterior) que permite a sua execução em qualquer máquina com o sistema *Docker* instalado, nomeadamente numa que esteja a correr na Cloud, tal como foi implementado no projeto. Esta abordagem permite ao servidor correr indefinidamente, oferecendo os *endpoints* mencionados e garantindo os serviços de execução de código. Da mesma forma, esta "containerização" permite que o servidor possa ser colocado a correr noutro tipo de ambiente ou máquina, bastando mínima configuração para garantir que tudo funcione de igual forma. Esta flexibilidade garante que as funcionalidades de execução de código podem continuar a ser utilizadas de forma independente da Cloud.

4.2.4 Google Colaboratory

O *Google Colaboratory* ou *Google Colab* é essencialmente uma plataforma de desenvolvimento na Cloud que permite a execução de código Python no navegador. A ferramenta é em grande medida gratuita e procura facilitar a investigação e colaboração na área da aprendizagem automática. Estes objetivos são assegurados pela disponibilização de recursos de computação poderosos através de um ambiente virtual que permite a importação de várias bibliotecas e *frameworks* populares na área. Associados à possibilidade de uso de *notebooks*, que permitem maior interatividade, experimentação rápida de algoritmos e partilha de código entre colaboradores, esta ferramenta permite a criação de modelos de aprendizagem automática com menor configuração e a baixo custo financeiro.

No âmbito do projeto, o *Colab* foi utilizado para desenvolver e experimentar código relacionado com *pipelines* da biblioteca *ScickitLearn*. Estas experiências passaram pela utilização de técnicas de extração de características baseadas em redes neuronais convolucionais combinadas com mecanismos de *Transfer Learning*, associadas posteriormente a classificadores baseados em redes neuronais densas ou algoritmos como o *Support Vector Machine* (SVM). Por outro lado, esta ferramenta permitiu a criação de *notebooks* para desenvolvimento de código capaz de criar e treinar *pipelines* de aprendizagem automática na *Vertex AI* da Google Cloud. Por último, foi usada para testar código desenvolvido por outros alunos do projeto *SmartFarm* com o objetivo de adaptar os modelos resultantes para a plataforma.

4.2.5 Testes Unitários

A testagem é uma componente chave para assegurar a fiabilidade do código e diagnosticar erros imprevistos em todas as fases de desenvolvimento do software. Nesta área, os testes unitários são uma das técnicas de testagem de software mais populares e utilizadas, cuja metodologia consiste em testar componentes isolados e independentes dos sistemas,

como funções, métodos ou *endpoints*. Além desta capacidade, este tipo de testes é ainda marcado pela simplicidade e rapidez de execução, facilitando o seu desenvolvimento e a interpretação de resultados. Tudo isto permite assegurar a qualidade do código em termos de performance e funcionalidades, promovendo um desenvolvimento contínuo e eficiente do mesmo.

No contexto do projeto, os testes unitários foram usados para testar vários pedidos da API REST disponibilizada pelo *backend*. Desta forma, foi possível verificar, de uma forma imparcial e eficiente, se as funcionalidades estavam a ser bem disponibilizadas e executadas pela lógica do código subjacente. Por outro lado, foi possível verificar se os *endpoints* devolviam respostas bem enquadradas com as expectativas, com tempos de resposta adequados a cada tipo de pedido.

5

Modelação da Plataforma

Como tem vindo a ser referido em capítulos anteriores, o principal objetivo desta dissertação foi a modelação e implementação de um Sistema de Informação (SI) para auxiliar o treino e a utilização de modelos de aprendizagem automática. A denominação SI é usada para definir um conjunto de componentes organizados que permite obter, guardar e processar informação que sirva de suporte ao processo de decisão. Ainda que o objetivo primário da plataforma desenvolvida não seja este, a denominação é suficientemente abrangente para compreender tudo o que inclua gestão e armazenamento de dados, envolvendo conceitos como transações, gestão de bases de dados e disponibilização de conteúdo específico.

Esta plataforma envolve vários componentes que têm de ser organizados entre si, de forma a obter um sistema capaz de suportar as funcionalidades requeridas. Tudo isto implica planeamento e escolha de soluções, num processo chamado de modelação do sistema. Este capítulo apresenta os passos deste processo, incluindo identificação de atores, requisitos funcionais e informacionais, definição da arquitetura, modelação de dados e integração/comunicação entre os vários componentes do sistema.

5.1 Atores do Sistema

A plataforma desenvolvida tem de lidar com o acesso e a participação de várias entidades, que devem ser conhecidas à partida e tratadas de forma diferente de acordo com os papéis que possam desempenhar. Podem ser chamadas de atores do sistema, e são apresentadas a seguir, incluindo a sua definição e características no âmbito deste projeto:

Agentes: Quem interage com o sistema. Estes devem ter papéis e permissões diferentes de acordo com o seu tipo, pelo que se podem dividir nos seguintes grupos:

- **Utilizadores Normais**: utilizadores que podem aceder à plataforma e usá-la. Estes são o grupo que tem menos permissões, podendo usar o sistema com limites e sem poderes administrativos;

- **Administradores**: utilizadores com mais poder na plataforma. Podem executar todas as operações disponibilizadas, tendo capacidades para tomar qualquer decisão de gestão disponível na plataforma, ou seja, executar qualquer pedido disponível na API;

- **Moderadores**: utilizadores que foram promovidos pelo administrador. Este grupo tem alguns poderes na plataforma, podendo monitorizar as ações de outros utilizadores e tomar algumas decisões administrativas;

- **Técnicos Agrícolas**: utilizadores credenciados pela plataforma como técnicos agrícolas. A sua introdução na plataforma foi feita para garantir a extensibilidade futura do sistema, havendo à partida uma categoria que possa ter permissões e papéis específicos relacionados com atividades agrícolas;

- **Especialistas Informáticos**: utilizadores credenciados pela plataforma como técnicos informáticos especializados na área de inteligência artificial. Por terem conhecimentos técnicos na área, este grupo tem capacidades exclusivas no que toca à criação, manutenção e verificação dos resultados dos *pipelines* da plataforma, sendo os únicos que podem tomar decisões de criação e treino que possam implicar custos financeiros;

Ainda no que toca aos agentes, o sistema está desenvolvido de forma a que as responsabilidades sejam cumulativas do ponto de vista administrativo, formando uma estrutura hierárquica de poder nos casos de administradores, moderadores e utilizadores. Por outro lado, técnicos agrícolas e especialistas informáticos têm, por natureza, as mesmas capacidades que os utilizadores normais, estando apenas credenciados para realizar tarefas específicas relacionadas com as suas áreas e não para decidir questões administrativas adicionais.

Dispositivos: correspondem aos diversos tipos de dispositivos que podem interagir com o sistema, nomeadamente: computadores, telemóveis ou tablets. Pelo facto de o *frontend* ter sido desenvolvido com base em *Angular* e *TypeScript*, utilizando a tecnologia SPA, o *website* pode ser usado num navegador moderno em qualquer um dos tipos de dispositivos mencionados.

Contas: trata-se das contas dos agentes descritos anteriormente, incluindo informações relevantes acerca dos mesmos e tokens de acesso que permitem a sua autenticação na plataforma.

Permissões: Surgem associadas às diferentes ações que cada tipo de agente pode executar dentro da plataforma, mencionadas anteriormente na secção dos agentes. Estas permissões são definidas de forma a garantir que os utilizadores possam ter acesso apenas aos dados e recursos corretos para os seus papéis e situações específicas.

5.2 Requisitos Funcionais e Informacionais

A plataforma desenvolvida satisfaz vários requisitos para cumprir os objetivos propostos na secção da Introdução, tendo sido feito um levantamento exaustivo dos mesmos sob a forma de requisitos funcionais que se pode encontrar no apêndice A. Estes apresentam de uma forma simples e intuitiva as funcionalidades do sistema, organizadas por categorias ou temáticas específicas.

De forma semelhante, também foi feito o levantamento dos *User Stories* (apêndice B) que podem ser definidos como uma explicação geral e informal das funcionalidades de uma peça de software descrita do ponto de vista do utilizador final, com o propósito de articular o valor que o software trará a quem com ele interaja [74], e permitir uma descrição mais próxima dos utilizadores finais, que podem ter papéis e funções diferentes.

5.3 Arquitetura

A arquitetura da plataforma está dividida em cinco camadas: a camada de servidor, a camada de armazenamento, a camada de inteligência artificial, a camada de apresentação (*frontend*) e a camada de servidor auxiliar. Excluem-se as camadas de rede e de segurança, que são geridas automaticamente pelos serviços da *Google Cloud Platform* (GCP). Um esboço da arquitetura, desenvolvido usando a ferramenta *Developer Cheat* [47], pode ser encontrado na figura 5.1, pretendendo ilustrar a forma como as diversas camadas interagem entre si. As setas representam o sentido dos pedidos para cada serviço, ou seja, de quem inicia a comunicação para quem a recebe. Por exemplo, é sempre a camada de apresentação que faz os pedidos à API REST, sendo que o inverso não acontece.

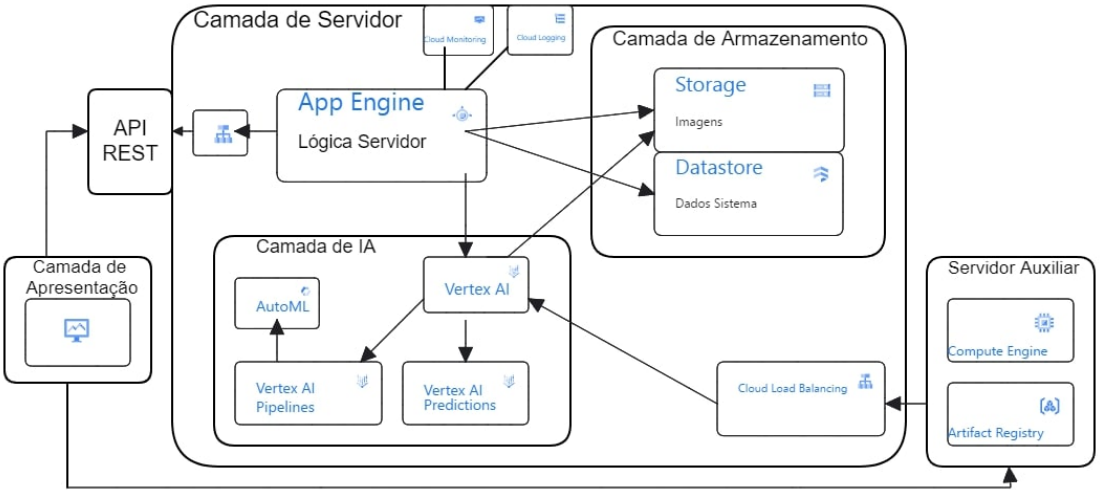


Figura 5.1: Arquitetura da Plataforma desenvolvida no âmbito da dissertação

As subsecções seguintes apresentam mais detalhes sobre cada uma das camadas da arquitetura, bem como a forma como interagem e comunicam entre si.

5.3.1 Camada de Servidor

Começando por aquela que tem uma maior importância relativa, a camada do servidor foi desenvolvida através dos serviços de computação disponibilizados pela Google. Para a parte da aplicação, usou-se a *Google App Engine* no modo *Standard*, segundo o modelo PaaS, resultando numa API REST pública que disponibiliza os *endpoints* da plataforma sob a forma de *URLs*. Estes podem ser chamados pelo cliente, permitindo acionar a lógica da camada de servidor capaz de digerir os dados de entrada e gerar respostas apropriadas.

Foram também usados serviços de monitorização disponibilizados pela Google, como o *Cloud Monitoring/Logging*, para ver informações relativas ao uso da plataforma. Estes permitiram consultar *logs* (registos) relacionados com os diversos *endpoints* disponibilizados, ajudando no diagnóstico de problemas e na avaliação da performance de resposta aos pedidos.

Ainda relacionado com a camada de servidor, as questões de autenticação e autorização fazem-se usando *tokens* gerados pela lógica subjacente, os quais guardam informações relativas a utilizadores e sessões, sendo requeridos em todos os pedidos da API, de forma a garantir comunicações seguras entre o cliente e o servidor. Nesta arquitetura, o *token* é gerado dinamicamente pelo servidor aquando de uma autenticação bem sucedida, sendo guardado pelo servidor e passado ao cliente para futuros pedidos.

5.3.2 Camada de Armazenamento

A camada de armazenamento também foi desenvolvida com recurso a serviços da Google Cloud. Para guardar os dados relativos a utilizadores, contas, meta-informação relacionada com dados e *templates*/parâmetros de código, o sistema usa a *Google Cloud Datastore* (GCD). Esta permite menor latência por ser uma base de dados NoSQL baseada em documentos, o que a torna ideal para tipos de dados de pequenas dimensões e que sejam acedidos várias vezes. No que toca a imagens, foi escolhida a *Google Cloud Storage* (GCS), especialmente otimizada para guardar tipos de dados não estruturados na forma de objetos. Este sistema permite ainda a organização do conteúdo de forma hierárquica através da criação de *buckets* e pastas.

A comunicação entre a camada de servidor e a de armazenamento é feita usando as bibliotecas para Java disponibilizadas pela Google. Estas permitem que o servidor contacte os serviços *Datastore* e *Storage* de forma eficiente e segura, garantindo a integridade dos dados. Por outro lado, o facto de a comunicação começar pela camada de servidor, permite integrar a lógica necessária para definir corretamente o tipo de ações a tomar em cada caso. A título de exemplo está a possibilidade de criação de pastas e subpastas na *Storage* dependendo do pedido do cliente, havendo mecanismos internos que asseguram que o pedido para o serviço inclui os parâmetros que façam a operação produzir os resultados esperados.

5.3.3 Camada de Inteligência Artificial

A camada de inteligência artificial é composta por todas as funcionalidades relacionadas com aprendizagem automática, que são asseguradas pelo serviço *Vertex AI*. Esta camada é responsável por gerir modelos de aprendizagem, permitir a sua invocação e apresentar as suas métricas. Além disso, apresenta *pipelines* de aprendizagem existentes na plataforma e ativa o treino de novos *pipelines* executando código parametrizado pelo utilizador.

De forma semelhante aos serviços de armazenamento, a camada de servidor contacta a camada de inteligência artificial através de bibliotecas para Java disponibilizadas pela Google. Mais uma vez, é a camada de servidor e a sua lógica subjacente que iniciam a comunicação de acordo com os pedidos do cliente. Além do mais, a camada de servidor serve também como ponto de ligação entre as camadas de inteligência artificial e de armazenamento. Isto acontece, por exemplo, nos casos em que conjuntos de dados guardados e catalogados na plataforma são usados para treinar *pipelines* de aprendizagem, tendo o servidor que obter as informações armazenadas e chamar os serviços de inteligência artificial subsequentes.

5.3.4 Camada de Servidor Auxiliar

Para satisfazer todas as funcionalidades da plataforma, foi necessário desenvolver um servidor auxiliar capaz de executar código Python guardado na plataforma. Este foi implantado na *Compute Engine* a partir de um *Container Docker* e fica a correr indefinidamente na Cloud, estando disponível para receber código parametrizado e executá-lo para criar pipelines de aprendizagem automática no serviço *Vertex AI Pipelines*.

Esta metodologia implica que a camada de servidor auxiliar esteja disponível para receber pedidos a partir das camadas de servidor ou de apresentação através de *endpoints* específicos. No entanto, o contrário não pode acontecer, não havendo a possibilidade de este servidor contactar ativamente as camadas referidas anteriormente. Em sentido inverso, aquando de determinados pedidos, esta camada comunica com a camada de inteligência artificial, ativando os mecanismos necessários para criar um *pipeline* de aprendizagem automática.

5.3.5 Camada de Apresentação

A camada de apresentação (*frontend*), como o próprio nome indica, apresenta uma interface dinâmica capaz de suportar os serviços da plataforma. Desta forma, contém lógica própria capaz de consumir e digerir conteúdo, para poder depois contactar a API REST (camada de servidor) de acordo com as ações do utilizador.

Esta camada é então responsável por interagir dinamicamente com o servidor de forma a disponibilizar, de uma forma amigável, informação relativa a contas, imagens, inteligência artificial, etc. Além do mais, facilita todo o processo de contacto com a API REST, permitindo lidar com detalhes relacionados com o tipos de pedidos, parâmetros,

autenticação e erros. Adicionalmente, comunica com a camada de servidor auxiliar, servindo de ponte entre esta e a camada de servidor, nomeadamente no que toca à obtenção de meta-informação acerca dos conjuntos de dados e aos *templates* de código;

5.4 Modelação de Dados

A modelação de dados é o processo de criação de um modelo que represente claramente a forma como os dados serão guardados numa ou em várias bases de dados. A modelação pretende obter uma estrutura para dados que permita compreender as relações que estes estabelecem entre si e que facilite o seu armazenamento, manipulação e obtenção para cada caso de uso.

As subsecções seguintes apresentam as opções de modelação de dados escolhidas para o sistema desenvolvido, havendo uma explicação introdutória de todo o modelo e seguindo depois para casos de uso específicos.

5.4.1 Tipo de Modelo de Dados

No caso habitual do sistema relacional, o processo de modelação de dados envolve a identificação das entidades participantes no sistema e das relações de ordem que estabelecem entre si. No entanto, embora o modelo relacional seja o mais comum, este pressupõe maiores desafios de construção e gestão, impedindo-o de ser a solução ideal para casos em que não se estabeleça um número significativo de relações entre as entidades. Os dados associados ao sistema desenvolvido enquadram-se nesta categoria, havendo entidades com atributos próprios relevantes, mas sem relações entre si, levando à escolha de um modelo híbrido para os guardar.

Desta forma, o modelo escolhido procura explorar as forças da tipologia NoSQL para as entidades e de outra tecnologia complementar para armazenar objetos. No que toca à primeira, foi escolhida uma base de dados orientada para documentos chamada *Google Datastore*, capaz de acomodar mudanças de estrutura e especialmente desenhada para menor latência e escalabilidade. Este sistema tira partido da flexibilidade da tipologia NoSQL, mas permitindo consultas (*queries*) suficientemente ricas para suportar as necessidades do sistema. Para lidar com o armazenamento de imagens em formato de objeto, foi escolhida a *Cloud Storage*, que assegura a gestão e organização de ficheiros e imagens de forma simples, segura e durável.

Dito isto, importa notar que a plataforma não se foca diretamente na gestão de dados, mas sim na disponibilização de serviços de aprendizagem automática. Os dados guardados na *Datastore* são relevantes para a disponibilização das funcionalidades gerais, mas não precisam de ser exaustivos ao ponto de guardar toda e qualquer informação. Ao invés disso, o sistema deve ser capaz de obter e guardar informações essenciais, sem esquecer a simplicidade do design. Por outro lado, a *Storage* garante não só o armazenamento e

organização de imagens, mas também a sua disponibilização para outros serviços, como a *Vertex AI*.

Pelo exposto, este modelo de tipo híbrido garante a flexibilidade, escalabilidade e performance requeridas para o sistema desenvolvido, sem obstacularizar outras operações necessárias e sem perder de vista a simplicidade de design, implementação e manutenção. As subsecções seguintes apresentam mais pormenores de cada um dos modelos de dados, abordando a forma como são implementados e quais as suas características.

5.4.2 Modelo de Imagens e Repositórios

Como referido anteriormente, o armazenamento de imagens de forma organizada é centrado no sistema de armazenamento da *Cloud Storage*. Para introduzir mais detalhes sobre o mesmo, é necessário mencionar o conceito de *bucket*, uma estrutura de armazenamento capaz de agrupar pastas e ficheiros segundo um padrão de regras pré-definido. Na fase de modelação do sistema decidiu-se usar quatro *buckets* seguintes:

- **mlplatform-public**: armazena imagens de utilizadores de forma pública, a fim de que qualquer interveniente no sistema possa aceder-lhes. A organização das imagens é feita usando pastas, reconhecidas pelo sistema como repositórios, que as organizam tipologicamente;
- **mlplatform-private**: armazena imagens de utilizadores de forma privada, pelo que apenas o utilizador que a criou possa aceder-lhes. A organização das imagens é feita da mesma forma que a do *bucket* anterior;
- **mlplatform-datasets**: guardar anotações de conjuntos de dados presentes nos dois *buckets* referidos anteriormente. Esta anotação é feita automaticamente pelo sistema, seguindo um padrão pré-definido segundo o qual um ficheiro *CSV* associa cada imagem a uma etiqueta específica;
- **mlplatform-pipelines**: guardar imagens de *pipelines* treinados na plataforma numa pasta específica. O *bucket* tem ainda outras pastas com dados referentes a *pipelines* treinados usando o serviço *Vertex AI*.

A criação de *buckets* e consequente organização dos ficheiros é feita de forma a distinguir diferentes tipos de dados e a permitir uma fácil navegação pelos mesmos. A existência de pastas/repositórios dentro dos *buckets* de imagens permite que os utilizadores possam definir a forma como estas são guardadas e obtê-las de forma coerente usando a API. Esta organização em pastas permite atribuir *URIs* específicos a cada imagem, guardadas sob a forma de um objeto, que os identificam de forma única.

Dentro de cada um dos *buckets* destinados a guardar apenas imagens (*mlplatform-public* e *mlplatform-private*), a organização é definida essencialmente pelo utilizador. Tirando o caso do *bucket* privado, em que uma pasta com o nome do utilizador é criada automaticamente pelo sistema, são apenas os utilizadores que podem criar ou remover repositórios.

As imagens são guardadas de acordo com parâmetros fornecidos nos pedidos da API, sob a forma de um objeto, que pode ser interpretado segundo diferentes formatos de acordo com a extensão contida no seu nome.

Todos os *buckets* e os respetivos conteúdos estão protegidos por mecanismos como o IAM (*Identity & Access Management*), sendo apenas acessíveis a partir da API. É também a API que gere o controlo de acessos através dos papéis concedidos pelo próprio sistema, possibilitando ou impedindo o acesso de determinados utilizadores a determinados recursos. Todos os dados guardados na *Storage* são encriptados, incluindo quando estão em trânsito entre a Cloud, o servidor e o cliente da API.

5.4.3 Modelo de Entidades

No seguimento da análise do modelo para guardar imagens de forma organizada, importa agora abordar o restante armazenamento fornecido pelo sistema. Este é composto por dados de menor dimensão, nomeadamente atributos de diferentes tipos associados a diferentes entidades. Esta informação é necessária para o correto funcionamento do sistema, possibilitando várias funcionalidades, controlo de acessos, autenticação e listagem de conteúdo.

Como dito anteriormente em 5.4.1, as entidades que compõem esta parte do modelo são sobretudo entidades independentes entre si, responsáveis por agrupar atributos relevantes. Muitas destas entidades estão relacionadas com os próprios recursos da API REST, referidos numa subsecção subsequente, 5.5.2. Além do mais, a maior parte dos atributos associados são do tipo *String*, *Long* ou *Float*, havendo exceções no que toca ao armazenamento de código e de parâmetros. Assim, é feita uma enumeração que contempla uma descrição inicial da entidade, a enunciação e explicação dos seus atributos, que inclui o seu tipo de dados e os valores que podem tomar.

Entidade: User

Descrição: entidade que representa um utilizador, com os seus atributos de perfil, papéis dentro do sistema e estado da conta.

Atributos:

- ***ID*** (String) identificador (email)
- ***user_email*** (String) - email
- ***user_username*** (String) - nome de utilizador (entre 5 e 30 caracteres)
- ***user_password*** (String) - password do utilizador (entre 8 e 128 caracteres)
- ***user_name*** (String) - nome do utilizador (entre 1 e 120 caracteres)
- ***user_account_state*** (String) - estado da conta (*ENABLED* ou *DISABLED*)
- ***user_profile*** (String) - estado do perfil (*PUBLIC* ou *PRIVATE*)
- ***user_role*** (String) - papel no sistema (*USER*, *MODERATOR*, *AGRIC*, *TECH* ou *ADMIN*)
- ***user_state*** (String) - estado (*ACTIVE* ou *REMOVED*)
- ***user_address*** (String) - endereço (entre 5 e 120 caracteres)
- ***user_pc*** (String) - código postal (formato nacional)
- ***user_region*** (String) - região (entre 3 e 120 caracteres)
- ***user_description*** (String) - descrição do utilizador (entre 1 e 120 caracteres)
- ***user_mobile*** (String) - número de telemóvel (formato nacional)
- ***user_landline*** (String) - número de telefone fixo (formato nacional)
- ***user_facebook*** (String) - link para *Facebook*
- ***user_instagram*** (String) - link para *Instagram*

Entidade: Session

Descrição: entidade que representa a sessão de um utilizador, tendo dois *tokens* gerados automaticamente pelo sistema para poder validar a autenticação no sistema.

Atributos:

- ***ID*** (String) - (*access_token*)
- ***access_token*** (String) - *token* de acesso
- ***token_creation_date*** (Long) - *timestamp* de criação
- ***token_expiration_date*** (Long) - *timestamp* de expiração
- ***refresh_token*** (String) - *token* de refrescamento
- ***token_user_email*** (String) - email do "dono" da sessão

Entidade: RefreshSession

Descrição: entidade que representa a sessão de refrescamento de um utilizador. A sua principal função é prolongar a duração do *token* normal e correspondente sessão,

permitindo que o utilizador não tenha de fazer outro login.

Atributos:

- ***ID*** (String) - (*refresh_token*)
- ***refresh_token*** (String) - *token* de refrescamento
- ***refresh_creation_date*** (Long) - *timestamp* de criação
- ***refresh_expiration_date*** (Long) - *timestamp* de expiração
- ***access_token*** (String) - *token* de acesso
- ***refresh_user_email*** (String) - email do "dono" da sessão

Entidade: Dataset

Descrição: entidade que representa um conjunto de dados anotado, com propriedades relevantes como a pasta anotada e a localização do ficheiro que contém a anotação na GCS.

Atributos:

- ***ID*** (String) - identificador *dataset_id*
- ***dataset_id*** (String) - identificador
- ***dataset_name*** (String) - nome
- ***dataset_storage_location*** (String) - localização do ficheiro de especificação na *Cloud Storage*
- ***dataset_creation_date*** (Long) - *timestamp* de criação do conjunto de dados
- ***dataset_owner_email*** (String) - email do criador do conjunto de dados
- ***dataset_labels_count*** (Map <String, Integer>) - mapa que associa o nome da etiqueta ao seu número de imagens

Entidade: Template

Descrição: entidade que representa templates de código guardados na plataforma, sob a forma de um objeto encriptado que pode ser instanciado com parâmetros pela lógica do servidor.

Atributos:

- ***ID*** (String) - (template_id)
- ***template_id*** (String) - identificador
- ***template_owner_email*** (String) - nome do criador do *template*
- ***template_creation_date*** (Long) - *timestamp* de criação
- ***template_notebook_content*** (Blob) - conteúdo do *notebook* (opcional)
- ***template_script_content*** (Blob) - conteúdo do *script* (obrigatório)
- ***template_parameters*** (Map <String, String>) - mapa que associa o nome do atributo ao seu valor no *template*

Entidade: ParameterSet

Descrição: entidade que representa conjuntos de parâmetros para templates de código guardados na plataforma, permitindo a sua instanciação com lógica adicional.

Atributos:

- ***ID*** (String) - (parameters_id)
- ***parameters_owner_email*** (String) - email do criador do conjunto de parâmetros
- ***parameters_content*** (Map <String, String>) - mapa que associa o nome do parâmetro ao seu conteúdo
- ***parameters_id*** (String) - identificador
- ***parameters_creation_date*** (Long) - *timestamp* de criação
- ***parameters_template_id*** (String) - identificador do *template* com o qual o conjunto de parâmetros está relacionado

Todas as entidades enumeradas têm um ID atribuído aquando da criação da entidade na *Datastore*. Entre parênteses encontra-se o atributo correspondente ao ID, estando sempre garantida a sua exclusividade no sistema. O formato de *Blob* é usado para guardar ficheiros que, neste caso, contém código que deve ser analisado e usado pelo sistema. Por outro lado, a menção de "Map" nalguns atributos representa a existência de um mapa, guardado sob a forma de String, que faz corresponder valores entre si.

Com estas entidades e respetivos atributos, o sistema consegue suportar as funcionalidades requeridas, dado que estas não implicam consultas de complexidade elevada, mas antes um acesso uniformizado a entidades independentes a partir do seu identificador único. A camada de servidor é responsável pela criação das entidades na base de dados, bem como pela sua gestão e utilização, havendo pedidos específicos para despoletar a lógica necessária para assegurar tais operações. Estes pedidos são abordados na secção seguinte (API REST), passando pelos princípios de construção, recursos e casos de uso relevantes.

5.5 API REST

Para garantir as funcionalidades enunciadas anteriormente, é necessário que o *backend* receba e responda a determinados pedidos. Assim, o sistema é suportado por uma API REST pública que permite a comunicação entre o cliente e o servidor (*backend*), podendo ser usada de forma independente do *frontend*. Esta secção aborda as características e regras de desenho desta forma de comunicação, enumerando o tipo de pedidos e os casos de uso suportados.

5.5.1 Princípios de Desenho

A sigla REST (*Representation State Transfer*), traduzida à letra, significa "transferência de estado de representação". Este foi um conceito introduzido pela primeira vez no ano de 2000, almejando uma forma mais simples e estruturada de transferência de dados na rede que não precise de estado [20]. O uso de métodos *HyperText Transfer Protocol* (HTTP) e o retorno de dados em formato JSON ajudam a assegurar essas características e a garantir a simplicidade e flexibilidade desta forma de comunicação.

Para poder desenvolver um sistema sobre a metodologia REST importa conhecer os seus princípios de desenho e construção, os quais são apresentados de seguida, incluindo referências à forma como são aplicados no sistema desenvolvido:

- **Modelo de Maturidade *Richardson***: Este modelo procura explicar a maturidade do desenho da API, medida pela aproximação a um sistema *RestFull*, ou seja, que satisfaça todas as características recomendadas pela metodologia REST. Este modelo tem 4 níveis que medem graus de complexidade do sistema progressivamente maiores: o nível 0 é o ponto de partida e apenas pressupõe o uso de HTTP para pedidos a um mesmo serviço; o nível 1 introduz o conceito de recursos, uma estrutura que reúne funcionalidades relacionadas entre si e acessíveis através de *endpoints* com o mesmo nome; o nível 2 pressupõe o uso correto de verbos HTTP para cada tipo de operação e, finalmente, o nível 3 associa endereços representativos de recursos ou funcionalidades às respostas dos pedidos [22].

  A API desenvolvida encontra-se no nível de maturidade 2, de forma a que as funcionalidades estejam agrupadas por recursos e que todos os *endpoints* façam uso dos códigos HTTP apropriados para cada tipo de operação. Nos pedidos, o servidor espera um método HTTP específico, como GET, POST, PUT, DELETE ou PATCH e, na resposta, o servidor envia códigos de acordo com o estado da operação, como OK (200), BAD REQUEST (400), NOT FOUND (404), etc.;

- **Operações CRUD**: o modelo de operações CRUD (*Create Read Update Delete*) é um modelo de arquitetura que estabelece as propriedades que um recurso deve suportar numa determinada API. A título de exemplo, pode imaginar-se o recurso de utilizadores que, segundo este modelo, deve suportar as operações de criação, leitura, atualização e remoção. Para tal, basta implementar os métodos POST, GET, PUT e DELETE, respetivamente. A maioria dos recursos da API desenvolvida apresenta as quatro operações, exceptuando os casos em que tal não é possível, seja por inadequação a algum dos métodos, seja por não fazer parte dos requisitos do sistema;

- **Autenticação**: o acesso à API é controlado por um sistema de autorização e autenticação baseado no uso de um *token* de autenticação gerado dinamicamente pelo servidor usando o método UUID e armazenado na *Datastore* do Google Cloud, juntamente com outras propriedades relevantes. Este *token* é devolvido ao utilizador

sempre que este execute uma operação de *login*, devendo ser usado em todos os pedidos subsequentes de forma a garantir a segurança e a autenticidade em cada pedido;

- **Autorização**: para além do *token* de acesso, cada utilizador tem um conjunto de permissões associado ao papel que lhe é atribuído pelo sistema. Estas permissões são utilizadas para controlar o acesso a determinadas funcionalidades e recursos, garantindo a segurança de determinadas operações dentro do sistema. Sempre que um utilizador não tem a permissão necessária para executar um pedido, o servidor devolve o erro de acesso negado (403 FORBIDDEN);

- **Sistema HTTPS**: o *HyperText Transfer Protocol Secure* (HTTPS) é uma extensão do protocolo HTTP que oferece uma melhoria na segurança das suas comunicações, protegendo credenciais, dados pessoais e todo o tipo de informações. Este processo é garantido pela encriptação dos dados a circular na rede, conseguida com auxílio de outros protocolos como o TLS (*Transport Layer Security*) ou o SSL (*Secure Sockets Layer*). Pelas razões apresentadas, a API suporta o uso do protocolo HTTPS para todos os pedidos, estando protegido de ataques como o *man-in-the-middle (MITM)*.

5.5.2 Recursos da API

Como referido anteriormente em 5.5.1, a API desenvolvida segue uma estrutura de recursos e *endpoints* que procura satisfazer determinados princípios de desenho. Neste contexto, os recursos pretendem representar entidades no sistema e podem ser manipulados através dos métodos HTTP oferecidos pela API, sob forma de *endpoints*. Assim, para a API desenvolvida, os recursos são os seguintes:

- **Utilizadores**: representam os utilizadores do sistema, incluindo o seu perfil e respetivos atributos. Este recurso é acessível através da extensão *'/users'* e inclui as operações de registo, remoção, atualização e obtenção de perfil;

- **Login**: representa as credenciais de acesso do utilizador, é acessível pela extensão *'login'* e engloba as operações de *login*, *logout*, validação e refrescamento de *token*;

- **Estado de Conta**: representa o estado de uma conta de utilizador, é acessível através da extensão *'/update'* e permite operações de alteração do estado, papel e visibilidade de perfil do utilizador;

- **Repositórios de Imagens**: representa a organização de pastas de imagens, é acessível através da extensão *'/repositories'* e inclui as operações de criação, obtenção, remoção e edição de repositórios;

- **Imagens**: representa o conceito de imagem, é acessível através da extensão *'/images'* e inclui as operações de criação, obtenção e remoção de imagens. Este recurso está ligado ao recurso anterior, uma vez que os objetos são criados dentro de repositórios;

- **Conjuntos de Dados**: representa o conceito de conjunto de dados, ou seja, anotação de conjuntos de imagens. É acessível através da extensão *'/datasets'* e inclui as operações de criação, obtenção, remoção e edição;

- **Modelos e Pipelines de Aprendizagem**: representa as capacidades de aprendizagem automática do sistema, é acessível através da extensão *'/aimodels'* e integra operações de obtenção dos modelos e *pipelines* presentes na plataforma, além de possibilidade de invocação de modelos;

- **Parâmetros de Código**: representa os parâmetros de código, é acessível através da extensão *'/parameters'* e inclui operações de criação, obtenção, edição e remoção de parâmetros, cujo objetivo é a instanciação num *template* de código;

- **Templates de Código**: representa os *templates* (modelos) de código, é acessível através da extensão *'/templates'* e engloba operações de upload, obtenção e remoção. A operação de obtenção está ligada ao recurso anterior, uma vez que a instanciação de código é feita a partir dos parâmetros de código apropriados.

5.5.3 Estrutura da API

A maior parte dos recursos identificados em 5.5.2 estão em conformidade tanto com o segundo nível de maturidade do modelo de maturidade de *Richardson*, como com as operações do modelo CRUD. Para tal, apresentam a estrutura enumerada abaixo, em que a palavra 'recurso' pode ser substituída pela extensão apresentada anteriormente para cada recurso:

- **POST** `https://<API_BASE_URL>/`*recurso* – Cria um objeto para o recurso correspondente, com base nas informações enviadas no corpo do pedido;

- **GET** `https://<API_BASE_URL>/`*recurso*/*id_recurso* – Devolve o objeto com o identificador *id_recurso* que esteja associado ao recurso correspondente;

- **GET** `https://<API_BASE_URL>/`*recurso*/ – Devolve todos os objetos associados ao recurso correspondente;

- **PATCH** `https://<API_BASE_URL>/`*recurso*/*id_recurso* – Edita ou atualiza o objeto com o identificador *id_recurso* que esteja associado ao recurso correspondente;

- **DELETE** `https://<API_BASE_URL>/`*recurso*/*id_recurso* – Remove o objeto com o identificador *id_recurso* que esteja associado ao recurso correspondente.

A enumeração anterior pressupõe o uso de *Path Parameters* (parâmetros de caminho), representados por '/*id_recurso*' no exemplo e usados para identificar o objeto a ser manipulado. Para além do uso deste tipo de parâmetros, a API pressupõe ainda o uso de *Header*

Parameters (parâmetros de cabeçalho) que contêm as informações relativas a credenciais de autenticação.

Por outro lado, aquando de um pedido do tipo POST, são usados *Body Parameters* (parâmetros de corpo), que permitem incluir dados binários de maior dimensão. O carregamento de imagens é um dos exemplos em que a grande dimensão do objeto pressupõe o uso de parâmetros de corpo, ao invés de outro tipo de parâmetros.

5.5.4 Casos de Utilização

No seguimento das subsecções anteriores, em que foram introduzidos os princípios de construção da API e as suas principais características, importa agora apresentar exemplos da sua utilização. Nesta vertente, deve compreender-se que a API é destinada tanto ao uso através do *frontend*, como ao uso de forma independente desta camada de apresentação.

O primeiro caso será abordado na próxima secção 5.6 e é aquele em que o utilizador não tem de lidar com o formato dos pedidos ou como processamento das respostas. No entanto, importa apresentar também dois exemplos da forma como a API pode ser acedida através de pedidos independentes, mostrando os formatos de entrada e de saída de cada um. Durante o desenvolvimento foi utilizada a ferramenta *Postman* para o contacto com a API, embora possa ser usado qualquer outro serviço que tenha capacidades semelhantes, desde que cada pedido esteja em conformidade com os requisitos da API.

Criação de um *Template* de Código:

Descrição: Criação de um *template* de código por *upload* de um script python

Endpoint: `https://mlplatform-310324.appspot.com/mlplatform/templates`

Tipo de Método: POST

Body Parameters (form-data):

email – Email do utilizador que faz o pedido
token – Token do utilizador que faz o pedido
script – Ficheiro de código com extensão '.py'
notebook (opcional) – Ficheiro de código com extensão '.ipynb'
templateName – Nome que deve ser atribuído ao *template*

Código de Resposta: 204 *NO_CONTENT*

Conteúdo da Resposta: Sem conteúdo

Obtenção das meta-informações de todos os *Templates* de Código:

Descrição: Obtenção das meta-informações de todos os *Templates* de Código (o pedido envia os cinco primeiros, sendo necessário usar o cursor para obter os seguintes)

Endpoint: `https://mlplatform-310324.appspot.com/mlplatform/templates/all`

Tipo de Método: GET

Header Parameters:

email – Email do utilizador que faz o pedido

token – Token do utilizador que faz o pedido

cursor – Cursor para pedir os próximos *templates*

Código de Resposta: 200 OK

Conteúdo da Resposta:

```
[
    {
        templateId: "exec_test",
        ownerEmail: "j.palmeiro@campus.fct.unl.pt",
        creationDate: 1711375010319,
        parameters: {
            span_time: "20",
            total_time: "60"
        },
        notebookExistance: true
    },
    {
        templateId: "gcp-pipeline-automl",
        ownerEmail: "j.palmeiro@campus.fct.unl.pt",
        creationDate: 1711375059786,
        parameters: {
            model_name: "default-model-name",
            training_set: "0.7",
            validation_set: "0.15",
            model_type: "MOBILE_TF_LOW_LATENCY_1",
            endpoint_display_name: "Default Endpoint Display Name",
            gcs_source: "gs://mlplatform-datasets/Seedlings/seedlings-3
    ↪ ba6d00d.csv",
            model_display_name: "Default Model Display Name",
            dataset_name: "Default Dataset Name",
            budget_time: "1000",
            pipeline_name: "default-pipeline-name",
            test_set: "0.15"
        },
        notebookExistance: false
    }
]
```

Listagem 5.1: Exemplo da resposta ao pedido de obtenção das meta-informações de vários *templates*, em formato JSON

5.6 Integração com o Frontend

Como foi aflorado na secção da arquitetura 5.3, a camada de apresentação do sistema tem o objetivo de facilitar o contacto com a API e apresentar os conteúdos de forma a melhorar a funcionalidade geral. Naturalmente, não é esperado que os clientes contactem a API diretamente, uma vez que não estão familiarizados com a formulação dos pedidos ou das respostas. O *frontend* procura então colmatar estas dificuldades e oferecer uma base para futuras melhorias do sistema como um todo.

Assim, para completar o exposto nas secções anteriores acerca da modelação da plataforma, importa detalhar a forma como o *frontend* se integra com as restantes componentes do sistema. Esta análise introduzirá os conceitos principais da escolha gráfica da interface, da interação com a API e dos detalhes da implantação na *App Engine* para disponibilização na web pública.

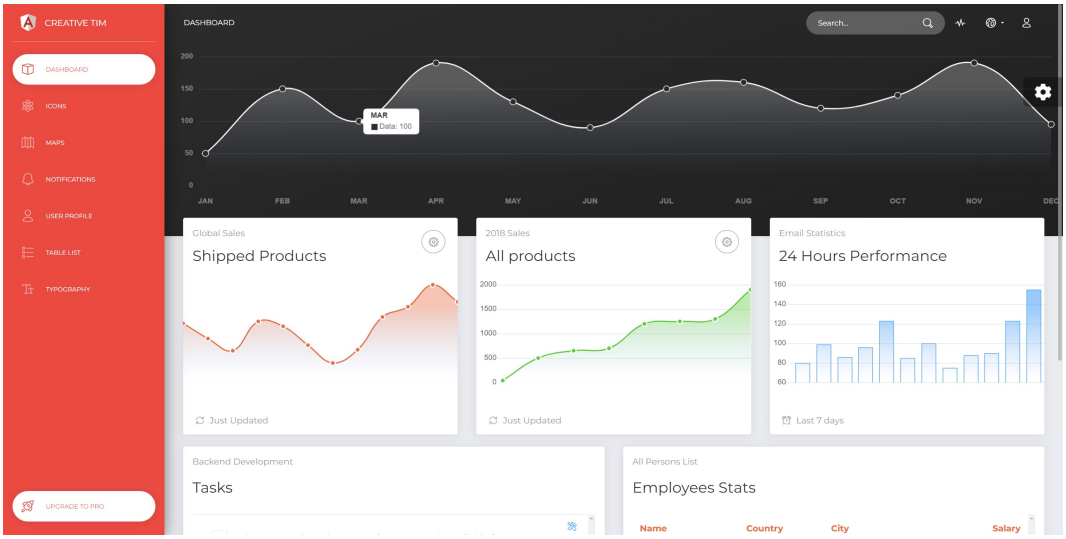
5.6.1 Design da Interface

Quando se aborda o desenvolvimento de uma UI (*User Interface*), é fundamental equilibrar aspetos como a estética, a simplicidade e o esforço para colocar tudo a funcionar. Geralmente, um incremento na estética implica colocar mais elementos e acabamentos nas páginas, de forma a torná-las mais apelativas. Este processo implica um grande esforço de planeamento, desenvolvimento e testagem, uma vez que envolve vários componentes de design da página usando HTML e da aplicação de estilos usando CSS.

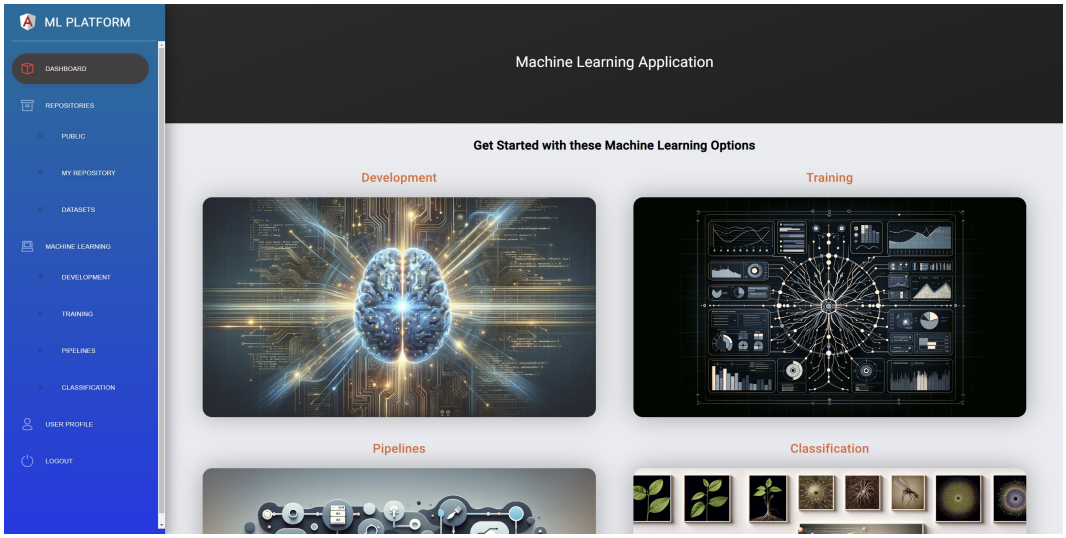
Para diminuir o esforço de planeamento e conceção de uma UI inicial, foi usado um *template* gratuito da "*Creative Tim*" [18]. Este permitiu obter uma interface inicial mais apelativa, mas que fosse customizável ao ponto de permitir adicionar e remover livremente componentes e estilos. A figura 5.2 mostra a comparação entre a página inicial do *template* e a página inicial da plataforma desenvolvida, que comprova a existência de uma base comum. A estrutura de ficheiros de código usada na fase de desenvolvimento foi também baseada no *template*, mas personalizada com outros conceitos, permitindo a adaptação a funcionalidades específicas do sistema.

Por outro lado, importa notar que o *template* apenas serviu como ponto de partida para a interface gráfica, tendo sido desenvolvidas todas as restantes páginas e demais funcionalidades. Este processo de desenvolvimento teve de se adaptar aos recursos que o sistema deveria disponibilizar. Assim, foi preciso criar páginas específicas relacionadas com os recursos da API (utilizadores, templates de código, repositórios...), capazes de apresentar as informações relevantes que lhes estão associadas.

Durante o processo de desenvolvimento, houve ainda um foco na simplicidade e usabilidade das páginas, diminuindo o número de elementos, alterando os estilos para facilitar a interação e colocando mensagens explicativas sempre que possível. Os aspetos de navegabilidade e responsividade também foram tidos em conta, disponibilizando uma barra de ligações para outras páginas e colocando elementos representativos do



(a) Dashboard Template



(b) Dashboard Plataforma

Figura 5.2: Comparação entre as páginas iniciais do *Template* e da Plataforma

carregamento dinâmico dos conteúdos.

5.6.2 Contacto com as APIs

A subsecção anterior abordou a forma como o *template* influenciou o design e organização do *frontend*, referindo a necessidade da sua adaptação ao que é disponibilizados pelo sistema. Estes recursos sustentam toda a camada de apresentação, sendo essencial o contacto com a API para que tudo funcione como pretendido, uma vez que esta é o ponto central do sistema como um todo, ao estabelecer a conexão com os serviços Cloud que sustentam a maior parte das operações.

A camada de apresentação estabelece o contacto com a API através de bibliotecas para *Angular* que suportam a execução de pedidos HTTP. Estes só podem ser executados

com lógica que permita a escolha dos parâmetros certos e respetivos formatos para cada situação. Da mesma forma, são usadas funções em *Angular* que permitem conexões assíncronas e a lógica necessária para lidar com as respostas obtidas de forma correta.

Estando intimamente ligada à API, a camada de apresentação executa pedidos para praticamente todas as ações que o utilizador possa tomar, desde a entrada nas diversas páginas ao uso dos botões nelas presentes. Dependendo das respostas obtidas, o sistema apresenta notificações que guiam o utilizador, confirmando ou negando a execução bem sucedida das operações.

Por outro lado, o *frontend* é ainda responsável por executar os pedidos para o servidor auxiliar, capaz de correr código guardado na plataforma. Estes pedidos são feitos de forma semelhante ao contacto com a API principal, embora tenham a particularidade de suportar o uso da técnica SSE. Esta tecnologia permite manter uma conexão ativa com outro servidor, assegurando o fluxo assíncrono de respostas através de uma única chamada ao *endpoint*. Desta forma, o utilizador é capaz de treinar *pipelines* de aprendizagem usando a plataforma, ao mesmo tempo que recebe informações relevantes acerca da sua execução em tempo real.

5.7 Visão Geral da Plataforma

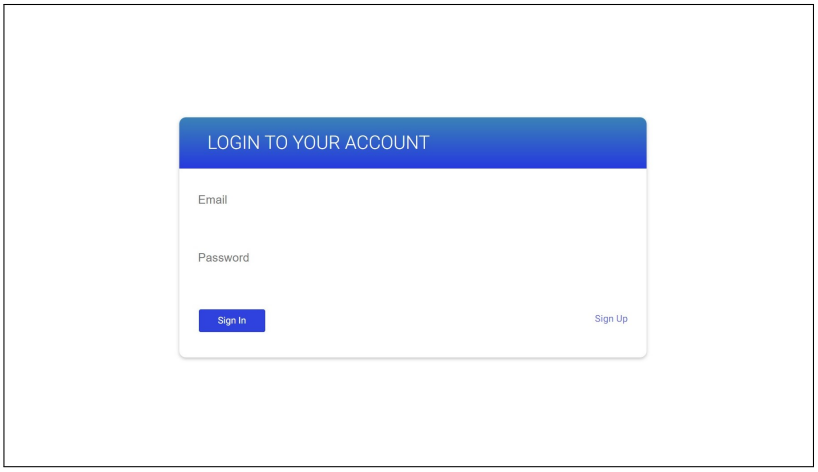
Até este ponto do documento, foram apresentadas tecnologias e conceitos de modelação e desenho do sistema. Antes de passar aos detalhes de implementação, importa agora apresentar uma visão do sistema do ponto de vista da usabilidade geral. A API oferece uma série de pedidos, cuja combinação e complementaridade permite executar uma série de funcionalidades no sistema, tendo sido desenvolvida uma interface gráfica para as disponibilizar.

Esta secção apresenta e explica as funcionalidades concretas e forma como se integram para formar o sistema global, exemplificando-as com imagens tiradas da interface.

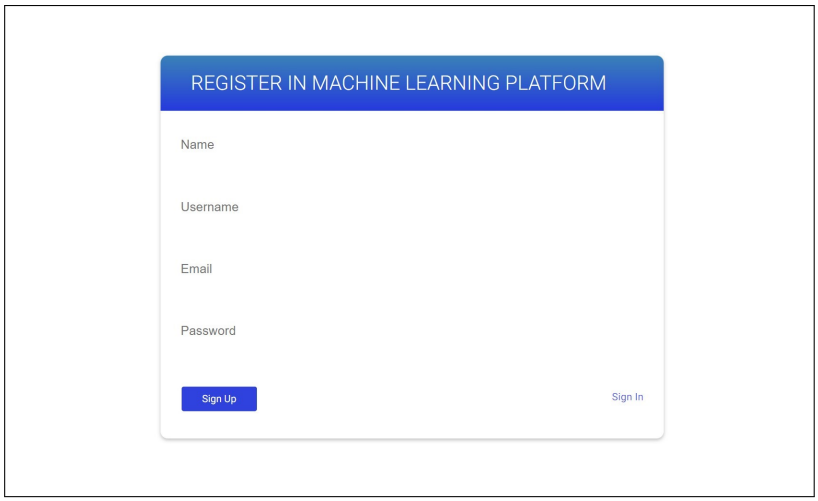
5.7.1 Autenticação, *Dashboard* e Perfil de Utilizador

O processo de autenticação é relativamente direto e simples de executar. Logo que o utilizador tenta aceder ao sistema, é automaticamente redirecionado para a página de *Login*, apresentada na figura 5.3, em cima. Esta apresenta simplesmente os campos de email e password, mas também um botão (”*Sign Up*”) que o utilizador deve pressionar para se poder registar no sistema, sendo redirecionado para a página apresentada na figura 5.3, em baixo.

Depois de completado o processo de autenticação, o utilizador é redirecionado para a página inicial (”*Dashboard*”) e o sistema guarda o email, o token de acesso e o papel do utilizador no sistema para poder executar pedidos futuros. Além do mais, caso o utilizador feche a janela do website, uma nova reentrada na plataforma permite entrar



(a) Página de Login



(b) Página de Registo

Figura 5.3: Páginas de Registo e Login da Plataforma

automaticamente na *Dashboard*, a menos que ambos os tokens de acesso já tenham expirado, o que implica a execução de uma nova operação de *login*.

A página inicial do sistema é apresentada na figura 5.4, importando mencionar dois aspetos relevantes. Em primeiro lugar, as imagens apresentadas pretendem melhorar a estética da página e motivar o utilizador a experimentar as funcionalidades. Ao passar sobre as imagens com o rato, estas tornam-se mais claras, mostrando que podem ser pressionadas para aceder à página que representam. Para além deste acesso rápido a outras páginas específicas, a barra lateral, presente em todas as páginas, oferece a possibilidade de acesso a todas as páginas disponibilizadas.

O utilizador pode alterar as suas informações através da página do perfil de utilizador, exemplificada na figura 5.5, que apresenta diversos campos de interesse. Por outro lado, também possibilita a verificação do papel na plataforma, como é apresentado no índice inferior direito da figura. Este é um aspeto importante, uma vez que apenas utilizadores com papéis privilegiados podem aceder a todas as funcionalidades do sistema. Ainda neste

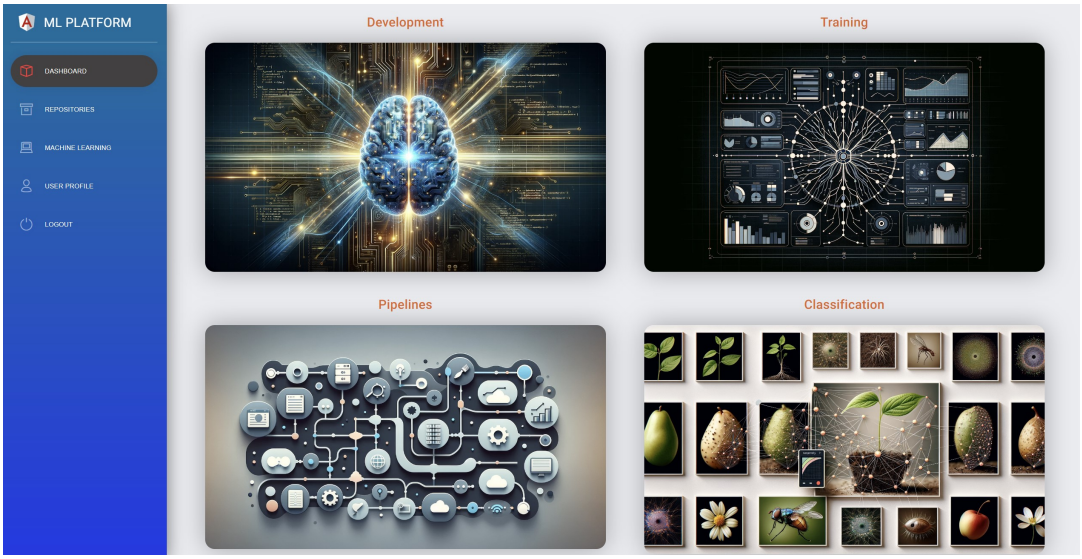


Figura 5.4: Página Inicial da Plataforma (*Dashboard*)

tópico, a API oferece aos papéis de moderador e administrador a possibilidade de alterar os papéis de outros utilizadores ou expulsá-los da plataforma caso o seu comportamento não seja adequado.

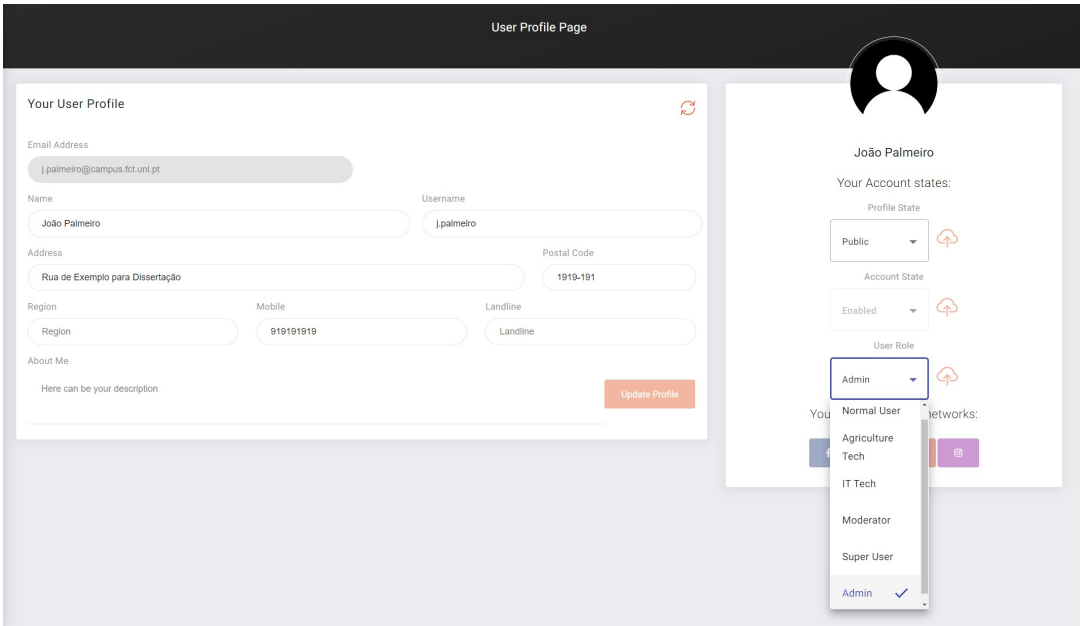


Figura 5.5: Página do Perfil de Utilizador

5.7.2 Conjuntos de Dados e Anotação

Após as páginas e funcionalidades gerais exemplificadas na subsecção anterior, abordar-se-á de seguida os conjuntos de dados e a forma como são estruturados e apresentados pelo sistema. Como foi referido noutras zonas do documento, existem dois tipos de repositórios. Ambos são semelhantes em termos de funcionalidades gerais, pelo que a

figura 5.6 permite exemplificar tanto o repositório público como o pessoal.

Este exemplo em específico é relativo ao repositório pessoal, na pasta "Plântulas", como se pode ver pelo URL presente na barra de navegação do browser: `https://frontend-dot-mlplatform-310324.oa.r.appspot.com/repositories/private/Plantulas`. Isto mostra a forma como o *frontend* associa o URL do website à estrutura de pastas dos repositórios. Assim, um utilizador pode aceder a um URL relacionado com pastas específicas dentro do sistema de forma direta.

Dentro desta pasta existem três imagens e uma subpasta chamada "*Misopates Orotium*". É possível ver um botão para adicionar uma nova pasta, do lado direito do ecrã, e um ícone que permite adicionar uma nova imagem. Álém disso, a plataforma está preparada com um mecanismo de *cache* de curta duração que permite guardar as imagens temporariamente. Isto permite que, caso o utilizador saia para outra página, possa voltar novamente a uma página concreta dos repositórios e visualizar imediatamente as imagens, sem que estas tenham de ser carregadas novamente.

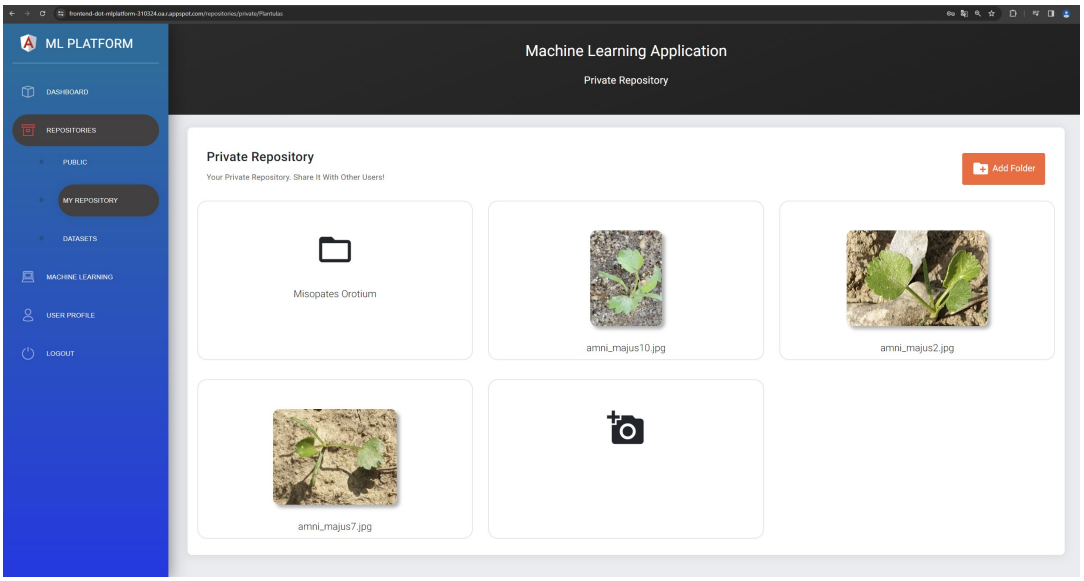


Figura 5.6: Página do Repositório Pessoal

As principais diferenças entre repositórios residem nas permissões de acesso e capacidades de gestão de cada um. O repositório público permite a todos os utilizadores o acesso e a possibilidade para adicionar imagens. Contudo, a criação de pastas e a remoção ou movimentação de imagens entre pastas apenas é permitida aos moderadores ou administradores do sistema. Em associação com as capacidades de expulsão de utilizadores por parte daqueles papéis, é possível garantir a correta utilização do sistema e a organização apropriada das imagens. Por outro lado, o repositório pessoal apenas permite o acesso do utilizador ao qual pertence, ou seja, ao utilizador que tem o *login* feito naquele navegador.

Todas estas ferramentas e possibilidades de gestão das imagens visam garantir um dos principais objetivos do sistema, a capacidade de integração destes dados anotados no treino de *pipelines* de aprendizagem automática. A principal funcionalidade de IA

presente na plataforma resulta precisamente da conjugação de *pipelines* com o mecanismo de AutoML, possibilitada pelo uso dos serviços da Google Cloud. Para tal, é necessário que os dados estejam anotados de determinada forma, ou seja, de acordo com um formato que permita a sua integração no treino de IA.

Os serviços escolhidos neste âmbito usam um ficheiro de especificação *CSV* que relaciona as localizações de cada uma das imagens na *Cloud Storage* às suas etiquetas classificativas. A solução encontrada para garantir que este processo de anotação seja simples, mas muito eficiente, tem por base a organização das imagens nos repositórios. A plataforma consegue gerar o ficheiro de especificação automaticamente com base na localização das imagens nas subpastas de uma determinada pasta.

A figura 5.7 apresenta a página de conjuntos de anotações da plataforma (excluindo a barra lateral). Na parte de baixo da página é possível ver um conjunto de anotações que foi criado para a pasta "*Seedlings*", que inclui 10 imagens para cada etiqueta. Em cima, foi escolhida a pasta "*Trees*" no selecionador, para a qual se pode criar um conjunto de anotações ao carregar no botão *Label Dataset*.

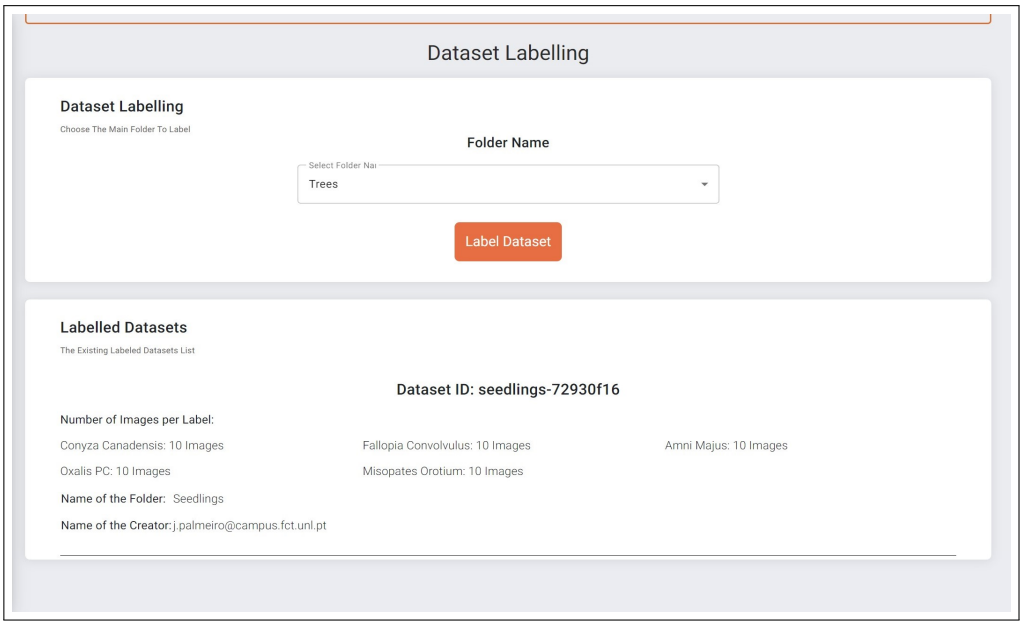


Figura 5.7: Página do Conjunto de Anotações

É possível criar mais anotações para uma mesma pasta, uma vez que é possível adicionar novas imagens a determinadas subpastas ou movê-las de umas pastas para outras. Ainda para mais, é precisamente este mecanismo que permite garantir a atualização e organização dos conjuntos de anotação que serão usados para treinar os *pipelines* de aprendizagem.

5.7.3 Parametrização e Execução de Código

No seguimento da secção sobre conjuntos de dados, importa agora abordar uma das funcionalidades mais importantes do sistema, ou seja, a criação e treino de modelos de aprendizagem automática. A abordagem adotada pelo sistema visa garantir três

grandes objetivos: uma utilização eficaz e segura dos mecanismos de treino de modelos de aprendizagem; a possibilidade de parametrização do treino desses modelos, sob o formato de *pipeline* com dados presentes no sistema e parâmetros relevantes; e a capacidade para adicionar novas técnicas de treino e desenvolvimento de modelos ao longo do tempo.

Para garantir tudo isto, a API suporta o carregamento de *templates* de código testado e funcional, apenas por parte de alguém que tenha papéis de administrador ou técnico informático. Os *templates* precisam de ter um formato específico, através do qual a lógica do servidor consegue fazer automaticamente a extração e armazenamento de parâmetros. Desta forma, é possível perceber que parâmetros podem ser editados em cada template, e disponibilizar ao utilizador os mecanismos para o fazer. Este processo pode ser executado através da página "Desenvolvimento" da plataforma, cujo exemplo é apresentado na figura 5.8.

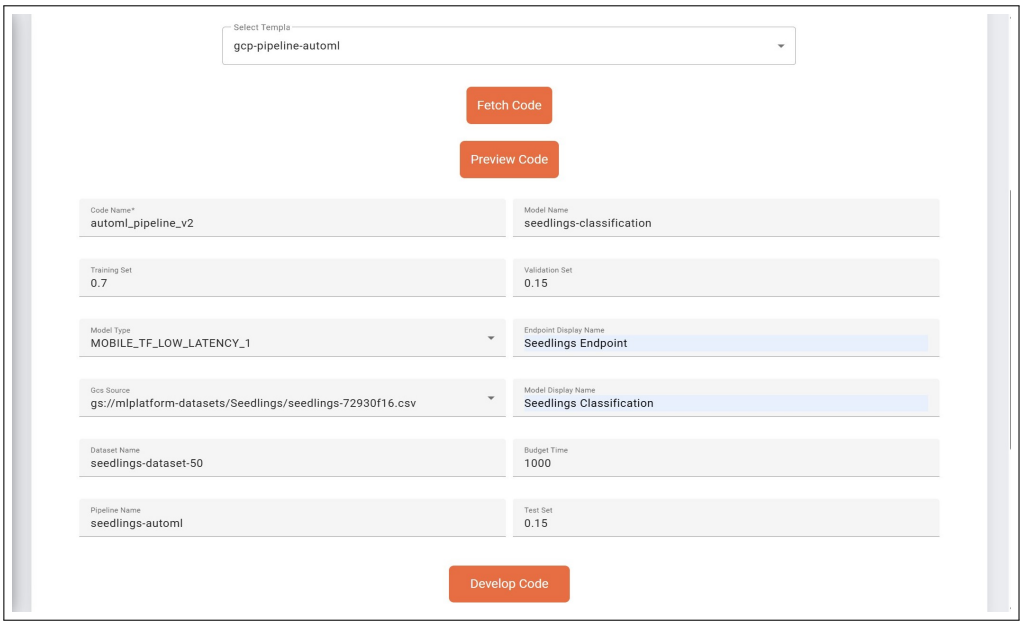


Figura 5.8: Página do Desenvolvimento, apresentando a parametrização de código

O exemplo da figura mostra a parametrização do *template* com nome *gcp-pipeline-automl*. Este tem código para criar toda a especificação de um *pipeline* capaz de assegurar todos os passos de treino de um modelo de *AutoML* na Cloud. É possível atribuir nomes às várias componentes do *pipeline* ou decidir acerca de parâmetros relevantes como os conjuntos de treino, validação e teste, segundo validadores que apenas permitem submeter uma soma destes valores igual a 1. Por outro lado, a plataforma limita automaticamente os valores "*Model Type*" para este *template* e permite escolher o conjunto de dados com o qual o modelo será treinado (dentro da lista de anotações apresentada na página apropriada, mostrada anteriormente na figura 5.7).

Pressionar o botão *Develop Code* cria um conjunto de parâmetros no sistema, segundo o nome *automl_pipeline_v2*. Este conjunto de parâmetros permite instanciar o *template* com nome *gcp-pipeline-automl* para o qual foi desenvolvido. Esta é uma das operações

disponibilizadas pela página de "Treino", apresentada na figura 5.9. Neste exemplo, foi obtido o *template* instanciado com o conjunto de parâmetros criado aquando da operação de parametrização de código, tendo sido posteriormente desencadeada a sua execução no servidor auxiliar.

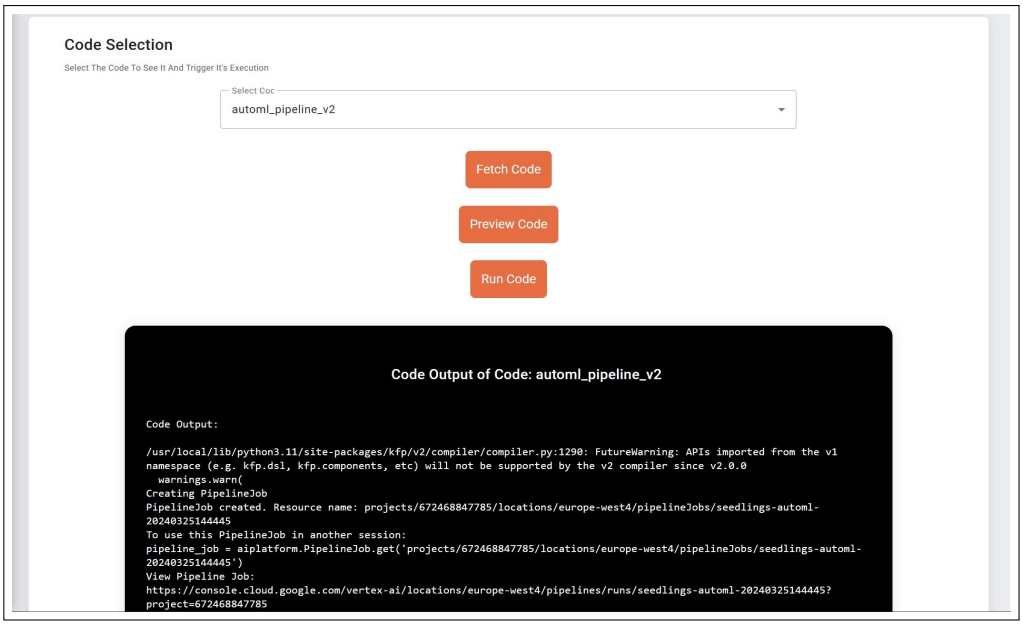


Figura 5.9: Página do Treino, apresentando a obtenção e execução de código

Ainda neste exemplo, é possível visualizar os registos da execução do código em tempo real, que continuarão a ser apresentados até que o processo esteja finalizado, sendo apresentada a mensagem "*Execution Complete*". Como se pode perceber no exemplo, foi criada a especificação do *pipeline* e o processo de treino está a decorrer. Importa notar que a operação de execução de código é exclusiva dos utilizadores com papel de administrador ou técnico informático, pelo risco potencial que comporta quando disponibilizada para outro tipo de utilizadores.

5.7.4 *Pipelines* e Modelos de Aprendizagem Automática

A última subsecção tratou da forma como são criados *pipelines* no serviço *Vertex AI* da Google Cloud, que podem depois ser verificados na plataforma. Existe uma página simples para visualizar o identificador do pipeline e uma imagem que o representa, exemplificada na figura 5.10. Esta mostra precisamente um *pipeline* que foi criado a partir do código parametrizado na subsecção anterior (5.7.3), com os dados presentes na pasta "*Seedlings*" do repositório público.

O *pipeline* que pode ser pré-visualizado na página apresentada na figura permitiu criar, treinar e implantar um modelo de aprendizagem automática usando a ferramenta de *AutoML*. Os modelos criados a partir da plataforma, são disponibilizados ao utilizador para que possam ser invocados para previsões futuras. Na página específica para o efeito,

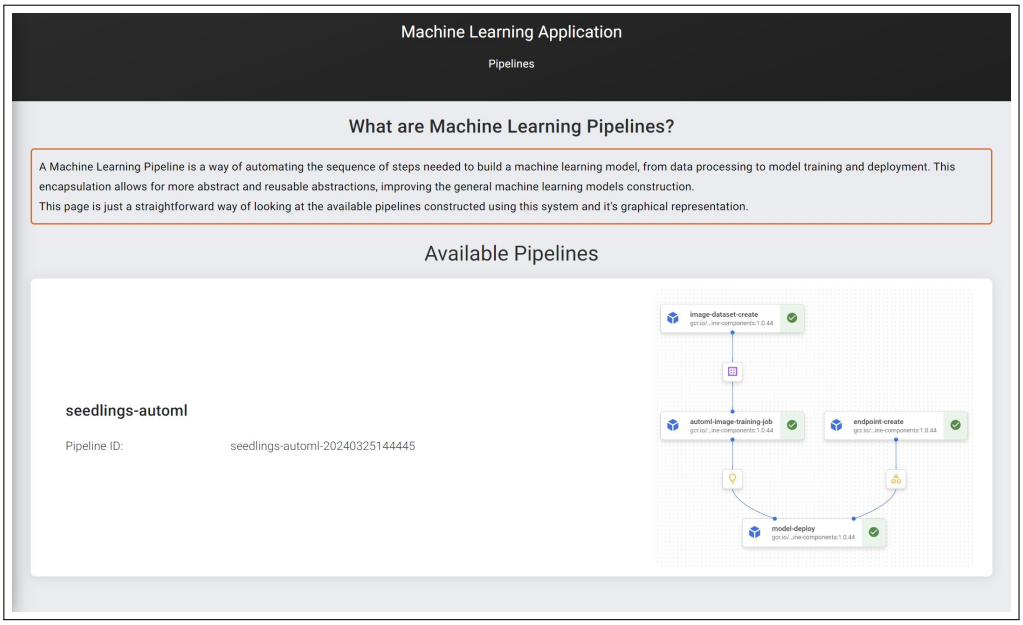


Figura 5.10: Página que apresenta os *Pipelines*

apresentada na figura 5.11, é possível ver o exemplo de utilização do modelo "*Seedlings Classification* para previsão da etiqueta classificativa da imagem carregada pelo utilizador.

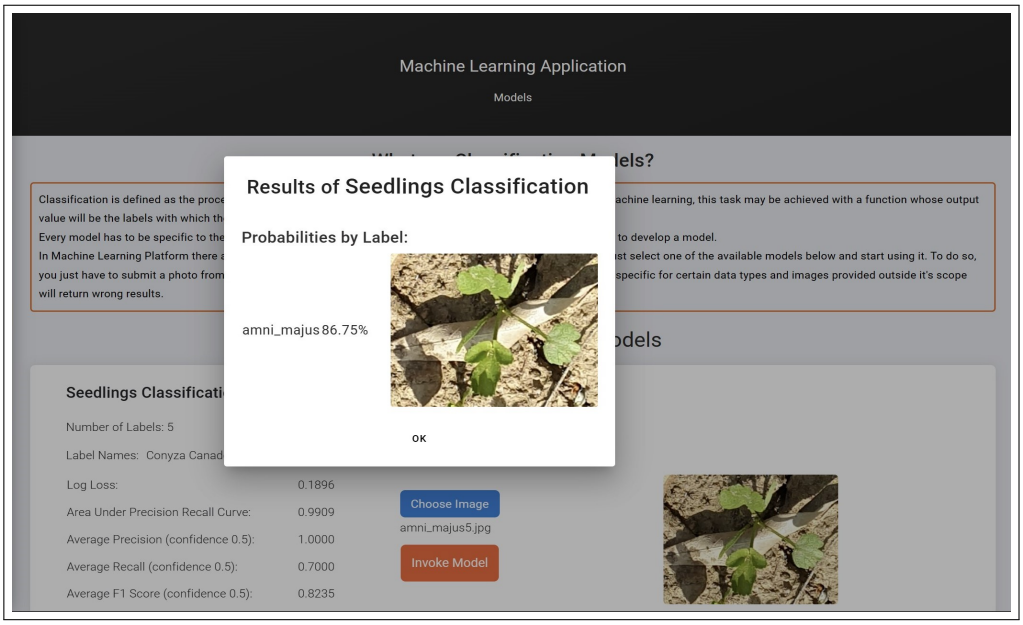


Figura 5.11: Resulto da previsão de um modelo de classificação

No fundo da imagem, é ainda possível ver algumas características do modelo utilizado, nomeadamente o número e nome das etiquetas que é capaz de prever. Conseguem também visualizar-se algumas das suas métricas mais relevantes, como é o caso da *Log Loss*, Previsão e Recall médios. Estas são geradas na fase de avaliação do modelo, que também é garantida de forma automática pelo uso do sistema *pipeline*.

5.8 Conclusões

Por percorrer os vários passos que levaram ao desenho do sistema, este capítulo é de grande importância no que toca à explicação do sistema desenvolvido, já que os conteúdos nele apresentados são resultados de planeamento da arquitetura, modelo de dados, estrutura da API e integração da mesma com o *frontend*. No entanto, importa mencionar que o resultado final não corresponde exatamente ao planeamento inicial.

Durante a fase desenvolvimento houve um processo importante de testagem de alternativas, alterando detalhes de implementação e o próprio desenho do sistema, de forma a que este pudesse satisfazer os requisitos propostos. Apesar de a maior parte das decisões de modelação terem sido tomadas à partida para o projeto, e as mudanças referidas terem sido mínimas, foi necessário adaptar alguns conceitos para que se pudesse chegar à melhor solução. Assim, a matéria exposta neste capítulo apresenta apenas o resultado final, ao invés de passos e decisões intermédias que não refletem o estado real do sistema.

Ainda assim, a fase de modelação foi essencial para se poder iniciar o desenvolvimento do sistema. Esta implicou um estudo prévio dos conceitos relacionados com as tecnologias usadas e da forma como todos se poderiam integrar para formar um sistema robusto. Houve ainda a preocupação de manter o sistema simples, mas eficaz, conseguindo cumprir as funcionalidades esperadas com o mínimo de pontos de falha.

A própria forma como o sistema está desenhado garante também a sua extensibilidade, havendo a possibilidade para adição de novas funcionalidades que sigam o código e organização existentes. O modelo de dados simples permite que se adicionem outro tipo de entidades, a API central e a camada de apresentação podem ser complementadas com novas funcionalidades e o servidor adicional pode ser reutilizado para correr outro tipo de código capaz de interagir com serviços na Cloud.

6

Detalhes de Implementação

Finalizada a descrição do processo de modelação do sistema, interessa agora abordar os detalhes de implementação de vários dos componentes que lhe estão associados. De referir que os conceitos abordados neste capítulo correspondem à versão do sistema tal como está aquando da entrega deste documento, deixando espaço para que possa sofrer melhorias.

Como abordado no capítulo anterior, o sistema foi desenvolvido com base em serviços Cloud, usando a *Google Cloud Platform*, e cuja API resultante comunica com a restante arquitetura do sistema de forma a disponibilizar as diversas funcionalidades. Desta forma, são apresentados detalhes relevantes relacionados com todos estes conceitos, desde o contacto com a base de dados, à estrutura do código e à forma como se garante a qualidade e segurança do sistema. O capítulo termina com uma visão global do sistema, das suas capacidades e da forma como se enquadra nos objetivos estabelecidos no primeiro capítulo.

6.1 Base de Dados

Como foi referido em capítulos anteriores, a camada de armazenamento foi desenvolvida com recurso aos serviços *Cloud Storage* e *Cloud Datastore*. O modo como estes são usados para guardar os vários tipos de dados foi discutida no capítulo anterior, mas ficaram de fora alguns detalhes de implementação.

Esta secção faz o apanhado dos detalhes relevantes relacionados com o armazenamento do sistema, falando da configuração das bases de dados, da forma como se estabelecem as comunicações entre estas e o servidor e dos mecanismos de segurança que envolvem acessos e comunicações.

6.1.1 Configuração das Bases de Dados

Antes de poder interagir com as opções de armazenamento de dados, quer na *Storage*, quer na *Datastore*, foi necessário configurar ambos os sistemas de acordo com determinadas preferências. Aquelas que acabam por ter mais impacto são a localização dos bancos de

dados, os modos de armazenamento e os mecanismos de segurança e encriptação. Por isso, vai-se explicar e justificar as opções tomadas para cada um destes serviços, bem como o impacto que têm no sistema.

6.1.1.1 Configuração da *Cloud Storage*

O primeiro passo para se poder começar a armazenar objetos na *Cloud Storage* é a criação de um ou mais *buckets*, que permitem organizar dados de tipos diferentes e tomar diferentes opções de configuração de acordo com as necessidades. Como mencionado no capítulo anterior, mais especificamente na subsecção 5.4.2, foram criados quatro *buckets* com propósitos específicos. No entanto, todos eles têm as seguintes configurações:

- **Localização**: *europe-west4* (localizada nos Países Baixos), escolhida por ser uma das poucas regiões da Europa que suporta o uso da técnica *AutoML* e pelo facto de que os *pipelines* que usam esta técnica precisam que os dados de treino estejam guardados na mesma região;
- **Classe de Armazenamento**: *Standard*, por ser a classe que oferece as melhores capacidades para armazenar dados que sejam acedidos com frequência. As outras opções, *Nearline*, *Coldline* e *Archive*, são destinadas a dados que sejam acedidos apenas uma vez por mês, trimestre ou ano, respetivamente;
- **Controlo de Acessos**: uniforme, garantindo que são usadas permissões ao nível do *bucket* para acesso generalizado. Todos os *buckets* têm prevenção de acesso público, pelo que os seus dados apenas são acessíveis através da plataforma ou da API;
- **Proteção de Dados**: política de exclusão reversível ativada por sete dias, garantindo que, a partir do momento em que os dados são eliminados, existe um espaço temporal de sete dias para os recuperar, período após o qual são excluídos de forma definitiva. Todos os dados são criptografados, havendo uma chave de criptografia gerida pela Google e usada automaticamente nos acessos através dos seus serviços (como é o caso da *App Engine*).

6.1.1.2 Configuração da *Cloud Datastore*

Por outro lado, a utilização da GCD é mais direta e as suas opções de configuração são menos exaustivas. Como já foi abordado, a *Datastore* é um modo de um serviço mais abrangente chamado *Firestore*. Para usar este serviço, foi criada apenas uma base de dados, no modo *Datastore* (a outra alternativa seria o modo Nativo), para armazenar todas as entidades do sistema.

Aquando da criação da base de dados, foi definido o modo *Standard* (em oposição ao modo flexível), uma opção mais equilibrada e que permite armazenar e obter os dados de forma rápida e eficaz. Para diminuir a latência das consultas à base de dados, a localização

dos servidores de armazenamento foi estabelecida para a Suíça (região denominada por *"europe-west6"* pela Google).

6.1.2 Ligação às Bases de Dados

Como referido na secção da arquitetura, é a camada de servidor que interage diretamente com a camada de armazenamento. Para tal, o servidor desenvolvido através do serviço *App Engine* utiliza as bibliotecas disponibilizadas pela Google para contactar ambos os serviços de armazenamento. Para usar estas bibliotecas, o sistema tem de declarar as dependências *Maven* necessárias, utilizando o ficheiro "pom.xml" presente no projeto, como é exemplificado no exemplo seguinte 6.1.

```
<!-- Cloud Datastore Dependency -->
<dependency>
    <groupId>com.google.cloud</groupId>
    <artifactId>google-cloud-datastore</artifactId>
</dependency>

<!-- Cloud Storage Dependency -->
<dependency>
    <groupId>com.google.cloud</groupId>
    <artifactId>google-cloud-storage</artifactId>
</dependency>
```

Listagem 6.1: Exemplo das dependências XML para os serviços *Cloud Datastore* e *Cloud Storage*

Depois de feita a declaração de dependências, a utilização das bibliotecas passa pela sua importação nas classes em que serão usadas, pela inicialização do serviço e pela utilização de lógica adicional para manusear os dados e validar as operações.

No caso da *Cloud Datastore*, o excerto de código apresentado em 6.2, retirado da classe correspondente ao recurso de utilizadores, é exemplificativo disso mesmo. Este apresenta a importação das bibliotecas, a inicialização dos serviços e a obtenção da entidade pretendida através de chaves criadas a partir do identificados único do utilizador.

```
/*Importacao das bibliotecas necessarias para o exemplo*/
import com.google.cloud.datastore.Datastore;
import com.google.cloud.datastore.DatastoreOptions;
import com.google.cloud.datastore.Entity;
import com.google.cloud.datastore.Key;
import com.google.cloud.datastore.KeyFactory;

@Path("/users")
@Produces(MediaType.APPLICATION_JSON + ";charset=utf-8")
public class UsersResource {

```

```
12 /*Obtencao e inicializacao do servico*/
13   private static Datastore datastore = DatastoreOptions.getDefaultInstance().
         ↪ getService();
14
15 /*Criaçao de uma "fabrica" de chaves para a entidade "User"*/
16   private static KeyFactory usersFactory = datastore.newKeyFactory().setKind("
         ↪ User");
17
18 @GET
19   @Produces(MediaType.APPLICATION_JSON)
20   public Response getUser(@HeaderParam("email") String email, @HeaderParam("
         ↪ token") String token) {
21
22     try{
23         /*...Restante logica de validacao do pedido...*/
24
25         /*Criacao de uma chave, baseada no email do utilizador*/
26        Key userKey = usersFactory.newKey(email);
27
28         /*Obtencao da entidade a partir da chave criada anteriormente*/
29        Entity user = datastore.get(userKey);
30
31         return Response.ok(json.toJson(new ProfileReturn(user, true))).build()
       ↪ ;
32     } catch (...){
33         /*Obtencao de erros no codigo anterior*/
34     }
35     }
36 }
```

Listagem 6.2: Exemplo de utilização da *Cloud Datastore* para obtenção de uma entidade

Já no caso do acesso à GCS, a importação e inicialização do serviço é feita de forma semelhante, embora a sua utilização seja diferente. O exemplo apresentado em 6.3 mostra a forma como se pode fazer o *download* de conteúdos armazenados neste serviço, havendo blocos de comentários para explicar os passos mais importantes.

Para este caso em concreto, a obtenção do objeto tem de ser feita de acordo com o caminho de ficheiros e subficheiros onde está armazenado. Para tal, é necessário não só aceder ao bucket certo, mas também à pasta certa, sendo precisa lógica capaz de incluir o caminho até ao objeto no seu próprio nome recebido em parâmetro. Por ser pouco relevante e não estar relacionado diretamente com a ligação à base de dados, esse código foi omitido do exemplo que se segue.

```
1 /*Importacao das bibliotecas necessarias para o exemplo*/
2 import com.google.cloud.storage.Blob;
3 import com.google.cloud.storage.BlobId;
```

```
4  import com.google.cloud.storage.Storage;
5  import com.google.cloud.storage.StorageOptions;
6
7  public StreamingOutput downloadObject(String objectName, boolean bucketType)
       ↪ throws FileNotFoundException {
8
9      /*Inicializacao do servico com credenciais definidas no sistema*/
10     Storage storage = StorageOptions.newBuilder().setProjectId(Constants.
       ↪ PROJECT_ID).build().getService();
11     String bucketName = bucketType ? Constants.PUBLIC_BUCKET_ID : Constants.
       ↪ PRIVATE_BUCKET_ID;
12
13 try {
14
15     /*Obtencao do objeto (Blob), de acordo com os parametros*/
16     Blob blob = storage.get(bucketName, objectName);
17
18     if (blob == null || !blob.exists()) {
19         throw new FileNotFoundException("The object with the given ID does
       ↪ not exist: " + objectName);
20     }
21
22     /*Subscricao do metodo "write", responsavel por passar o conteudo do Blob
       ↪  para uma variavel de Streaming*/
23     StreamingOutput stream = new StreamingOutput() {
24         @Override
25         public void write(OutputStream output) throws IOException,
       ↪ WebApplicationException {
26             /*Utilizacao do metodo "getContent" para obter o conteudo*/
27             output.write(blob.getContent());
28             output.flush();
29         }
30     };
31
32     return stream;
33
34 } catch (Exception e) {
35     throw new FileNotFoundException(e.getMessage());
36 }
37 }
```

Listagem 6.3: Exemplo de utilização da *Cloud Storage* para *download* de um objeto

Os excertos de código apresentados pretendem exemplificar o contacto com as bases de dados que compõem a camada de apresentação. Tentou-se mostrar um caso genérico de obtenção de dados para cada um dos serviços, com o objetivo de estabelecer um paralelismo

entre a forma de utilização dos serviços para uma funcionalidade semelhante. Para disponibilizar todas as funcionalidades do sistema, foram necessárias outras operações de consulta mais complexas, bem como outras de inserção, atualização ou remoção de dados.

6.1.3 Mecanismos de Segurança e Consistência de Dados

Abordadas as configurações e a forma como a camada de armazenamento é contactada, passa-se de seguida aos mecanismos de segurança e consistência de dados que são aplicados. Quando são usadas as bibliotecas da Google para Java, a segurança é assegurada em camadas, sobretudo em duas frentes.

Por um lado, a segurança dos dados em repouso está sempre garantida através de mecanismos de encriptação e de controlo de acessos, sendo que apenas entidades credenciadas, como o servidor que corre na *App Engine*, podem aceder aos dados. Ainda nesta matéria, a própria camada de servidor faz a verificação de acesso com base nos papéis dos utilizadores, garantindo que os dados são disponibilizados apenas a quem verdadeiramente lhes possa aceder. Por outro lado, quando em trânsito, a segurança é garantida através da encriptação de todos os conteúdos, sendo que apenas entidades credenciadas pelos serviços podem ler os dados corretamente.

Relativamente a outro tópico, o sistema de bases de dados usado pela *Cloud Datastore* começou por apresentar consistência eventual. Este conceito reflete a possibilidade de as atualizações poderem demorar a ser propagadas, resultando em leituras parcialmente desatualizadas num curto período de tempo. A introdução do *Firestore* no modo *Datastore* evoluiu as suas capacidades, de modo a assegurar consistência forte a garantir que todas as leituras refletem a versão mais recente dos dados após uma atualização.

Ainda assim, o uso de transações para assegurar a consistência dos dados continua a ser comum, especialmente em operações mais complexas. Este mecanismo permite que a execução de várias operações seja feita em bloco, assegurando que todas sejam executadas ou que, em caso de falha, nenhuma seja aplicada. Isto é sobretudo importante nos casos em que é preciso inserir dados ou fazer atualizações que dependam de leituras prévias. Para tal, o sistema usa sempre transações para implementar operações que envolvam escritas, como é o caso do exemplo apresentado em 6.4.

```
1  /*Importacao das bibliotecas necessarias para o exemplo*/
2  import com.google.cloud.datastore.Entity;
3  import com.google.cloud.datastore.Key;
4
5  /*... Criacao das variaveis de classe (datastore e usersFactory) ...*/
6
7  @PATCH
8  @Consumes(MediaType.APPLICATION_JSON)
9  public Response doUpdateProfile(UpdateProfileData data) {
10
```

```
11    /*... Codigo de Validacao ...*/
12
13    /*Criacao de uma transacao*/
14    Transaction txn = datastore.newTransaction();
15
16    try {
17
18      /*Validacao do utilizador e obtencao da entidade*/
19      TokensUtil.checkIsValidAccess(data.getToken(), data.getEmail());
20      Key userKey = usersFactory.newKey(data.getEmail());
21      Entity user = txn.get(userKey);
22
23      /*Alteracao das informacoes do utilizador com base nos parametros
            ↪ recebidos no pedido*/
24      user = Users_DB.changeProperty(data, userKey, user);
25
26      /*Atualizacao da base de dados e confirmacao da operacao*/
27      txn.put(user);
28      txn.commit();
29
30      /*... Restante Código ...*/
31  }
```

Listagem 6.4: Exemplo da utilização de uma transação para atualização de dados na *Cloud Datastore*

Este exemplo de código mostra a forma como um utilizador é atualizado na base de dados usando uma transação. Existe lógica escondida no que toca à atualização das informações do utilizador, que passa pela criação de uma nova entidade, mas que não tem relevância para o exemplo de utilização da transação. Assim, este mecanismo permite não só que o servidor consulte a versão mais atualizada do utilizador na base de dados, mas também que a sua substituição seja feita de maneira coerente, ainda que haja pedidos que manipulem os mesmos dados e sejam executados de forma concorrente.

6.2 Servidores

Na secção anterior foram apresentados os detalhes de implementação da camada de armazenamento do sistema, incluindo referências ao ponto central do sistema, a camada de servidor. Tendo isso em conta, esta secção revelará os detalhes desta camada, passando pela forma como a *App Engine* foi configurada e utilizada, bem como pelos seus mecanismos de segurança. Por outro lado, falar-se-á também do servidor adicional responsável pela disponibilização da funcionalidade de execução de código e da forma como foi desenvolvido, testado e implantado num servidor da Cloud.

6.2.1 Servidor da *App Engine*

Como foi dito em capítulos anteriores, nomeadamente na secção 3.2.1, a *Google App Engine* (GAE) é um serviço flexível que facilita o desenvolvimento e implantação de aplicações na Web. No caso deste projeto em particular, o objetivo deste serviço é o desenvolvimento de um servidor capaz de disponibilizar uma API REST capaz de satisfazer vários pedidos. Para tal, é usada a infraestrutura da Google, responsável por vários passos intermédios que possibilitam o uso da lógica de computação pretendida.

Em primeiro lugar existe um balanceador de carga, responsável por receber os pedidos e redirecioná-los para o ambiente de execução correto, com base na configuração, carga de trabalho e disponibilidade dos recursos. Este balanceador assegura a disponibilidade e escalabilidade horizontal do sistema, alocando recursos conforme as necessidades do sistema. Seguidamente, o ambiente de execução processa o pedido de acordo com a lógica especificada no código, podendo recorrer a outros serviços da Google Cloud, como os de armazenamento ou de aprendizagem automática. Por último é gerada a resposta, que é enviada para o utilizador através do mesmo caminho de rede, completando o ciclo de execução.

Como os restantes serviços da Google, a *App Engine* também tem opções de configuração, embora sejam reduzidas ao máximo para assegurar os próprios objetivos de simplicidade do serviço. A configuração mais importante passa pela escolha entre o ambiente padrão e o ambiente flexível, sendo o primeiro mais equilibrado e automático, e o segundo mais flexível e granular. A escolha acabou por recair sobre o primeiro, delegando mais aspetos de configuração para a infraestrutura. Ainda assim, a aplicação inclui duas configurações adicionais, especificadas no ficheiro "appengine-web.xml" e apresentadas em 6.5.

```
1 <!-- Definir numero minimo de instancias e tempo de espera maximo -->
2 <automatic-scaling>
3     <min-idle-instances>1</min-idle-instances>
4     <max-pending-latency>8s</max-pending-latency>
5 </automatic-scaling>
```

Listagem 6.5: Opções de configuração da *App Engine*

A primeira é responsável por assegurar que há sempre uma instância disponível em cada momento e a segunda por detalhar o tempo máximo que os pedidos podem demorar até que novas instâncias sejam criadas para os suprir. Ambas são incluídas no sistema, por um lado evitando que pedidos tenham de esperar pela inicialização de novas instâncias e, por outro, limitando o tempo de espera.

6.2.2 Autenticação e Autorização

Uma vez explicados alguns pormenores de execução e configuração da camada de servidor, implantada na *App Engine*, faz sentido aprofundar também os mecanismos de segurança

inerentes quer ao serviço em si, quer à lógica do servidor. Utilizando este tipo de serviço, as comunicações internas entre serviços da Google, contemplando a fase de balanceamento de carga e criação de instâncias, estão protegidas automaticamente pelo uso de uma rede privada. Além disso, a encriptação dos dados nos pedidos e nas respostas é sempre assegurada pelo uso do protocolo HTTPS.

Garantida a segurança de comunicações e pedidos externos, foi necessário implementar medidas capazes de autenticar os utilizadores do sistema. Uma das opções tomadas foi a encriptação e verificação de *passwords* através do algoritmo *Argon2*, oferecendo uma implementação segura contra vários tipos de ataques, como os de força bruta ou os baseados em GPUs. O algoritmo usa diversos parâmetros para assegurar a robustez da encriptação, como se pode exemplificar em 6.6, num excerto retirado de uma das classes usadas pela lógica do servidor. (Todas as ocorrências de termos em *Caps Lock*, como "SALT_LEN", representa o uso de constantes definidas anteriormente).

```
1  public static Argon2 argon2 = Argon2Factory.create(Argon2Types.ARGON2id, SALT_
       ↪ LEN, HASH_LEN);
2
3  public static void setDefaultParameters() {
4    argon2 = Argon2Factory.create(Argon2Types.ARGON2id, SALT_LEN, HASH_LEN);
5  }
6
7  public static String hashPassword(String password) {
8    return argon2.hash(ITERATIONS, MEMORY_USAGE, THREADS, password);
9  }
10
11 public static boolean verify(String hash, String password) {
12   return argon2.verify(hash, password);
13 }
```

Listagem 6.6: Exemplo de utilização do algoritmo *Argon2*

Outra das técnicas de segurança implementadas é o uso de *tokens* de acesso, para haver uma forma de autenticação que não requeira um uso repetido da *password*. Assim, o utilizador apenas tem de utilizar a *password* pessoal no *login*, operação que gera automaticamente um *token* de acesso a partir da classe UUID (*Universally Unique Identifier*) para Java. Este é guardado numa entidade da base de dados (apresentada em 5.4.3) e é requerido para contactar a API, sendo verificada a sua validade e autenticidade em todos os pedidos.

No que toca à autorização, é usado um mecanismo de papéis baseado em níveis, que garante acesso a determinadas funcionalidades apenas a utilizadores credenciados. Para essas operações, é feita a verificação do papel do utilizador, sendo negado o acesso caso não tenha o conjunto de permissões necessárias para executar o pedido. Da mesma forma, é sempre feita a análise do utilizador quanto à sua validade no sistema, impedindo o

acesso caso este tenha sido removido ou banido. O exemplo 6.7 apresenta alguns dos métodos usados nos pedidos para fazer esta validação.

```
1  /*Verifica se o utilizador foi removido ou esta banido*/
2  public static boolean isRemovedOrBannedUser(Entity user) {
3    return user.getString(Users_DB.ACCOUNT_STATE).equals(Account_State.REMOVED.
         ↪ toString())
4      || user.getString(Users_DB.USER_STATE).equals(User_State.BANNED.toString()
5  }
6
7  /*Verifica se o utilizador pode banir outro com base nos seus papeis*/
8  public static boolean hasStatePermission(Entity user, Entity target,
       ↪ Transaction txn) {
9
10   int user_level = getLevel(Action.MODIFY_STATE, Roles.valueOf(user.getString(
         ↪ Users_DB.ROLE)), txn);
11   int target_level = getLevel(Action.MODIFY_STATE, Roles.valueOf(target.
         ↪ getString(Users_DB.ROLE)), txn);
12
13   return target_level < user_level;
14 }
```

Listagem 6.7: Exemplo de métodos usados para validar utilizadores e as suas permissões

O uso desta e de outra lógica, associada aos diversos pedidos, permite garantir a autenticidade dos utilizadores e o controlo de acessos no que toca à utilização do sistema. Sempre que as condições de acesso não estejam reunidas, a API devolve respostas com códigos e mensagens adequadas, como o 404 NOT_FOUND ou o 403 FORBIDDEN.

6.2.3 Servidor Flask

Para além do servidor principal, que cobre praticamente todos os pedidos, existe um servidor adicional responsável unicamente por uma funcionalidade específica, mas de grande importância no contexto global do sistema. Este servidor auxiliar foi desenvolvido usando a *microframework Flask*, que permite criar uma API REST limitada de forma simples, mas eficaz.

Neste contexto, foi necessário criar um ambiente local *Python* com as dependências necessárias para desenvolver o servidor, mas também para executar os templates de código guardados no sistema (mais detalhes em 6.3.2). A execução do ficheiro principal "gcp-server.py", complementado com a definição e uso de variáveis de ambiente, coloca o servidor a correr no ambiente pretendido, ficando disponível para receber pedidos HTTP através dos seus *endpoints*.

Este servidor tem apenas dois *endpoints*, um para receber e executar código, e outro para enviar os resultados da execução de código em tempo real. No que toca ao primeiro pedido, algum do código utilizado é apresentado e comentado abaixo em 6.8. Este apresenta o

cabeçalho do método, e a forma como é criada uma *thread* para executar o código da função "run_code()", responsável pela execução e captura do resultado do código. São omitidos detalhes de implementação relacionados com parâmetros e variáveis, nomeadamente a validação de uma *password* obrigatória sem a qual o código não pode ser executado.

```
@app.route('/code', methods=['POST'])
def code_execution():
    #... Codigo de validacao de parametros e inicializacao de variaveis ...

    threading.Thread(target=run_code, args=(session_id,)).start()

    return jsonify({"message": "Execution started", "session_id": session_id})
    ↪ , 200

def run_code(session_id):
    #... Codigo de obtencao e inicializao de variaveis...
    python_executable = sys.executable
    try:
        with subprocess.Popen([python_executable, "-u", file_path], stdout=
    ↪ subprocess.PIPE, stderr=subprocess.STDOUT, bufsize=1, universal_
    ↪ newlines=True) as proc:
            for line in proc.stdout:
                received_code[session_id]['output'].append(line)

        proc.wait()
        if proc.returncode != 0:
            received_code[session_id]['output'].append(f"\nProcess terminated
    ↪  with return code {proc.returncode}")
    except Exception as e:
        error_trace = traceback.format_exc()
        received_code[session_id]['output'].append(f"Error: {str(e)}\n{error_
    ↪ trace}")
```

Listagem 6.8: Segmentos de código do *endpoint* que garante a receção e execução de código

Aquando de um pedido bem-sucedido, inicia-se a execução do código enviado, que se mantém até que termine naturalmente ou haja um erro. Os resultados desta execução, como registos relacionados a passos intermédios, são guardados numa variável global, motivando a existência de um segundo pedido complementar. Este é capaz de criar uma *stream* de dados, que se mantém ativa durante todo o período de execução do código, e que é responsável por devolver os seus resultados. Para tal, o pedido faz uso da técnica SSE, que permite que um navegador possa manter uma conexão com um endpoint em tempo real. O código deste *endpoint* é mostrado em 6.9 para apresentar os detalhes de implementação da funcionalidade.

```
@app.route('/stream/<session_id>')
```

```
def stream(session_id):
    def generate():
        last_message_time = time.time()

        while True:
            if session_id in received_code:
                if received_code[session_id]['finished']:
                    if not received_code[session_id]['output']:
                        yield "data: Execution complete\n\n"
                        break

                while received_code[session_id]['output']:
                    line = received_code[session_id]['output'].pop(0).rstrip(
    ↪ '\n')
                    yield f"data: {line}\n\n"
                    last_message_time = time.time()

            #Verificar tempo para garantir o formato SSE
            if time.time() - last_message_time >= 5:
                yield "data: keep-alive\n\n"
                last_message_time = time.time()

            time.sleep(1)  #Pausa para prevenir ciclos

    return Response(stream_with_context(generate()), content_type='text/event-
    ↪ stream')
```

Listagem 6.9: Código do endpoint que garante o envio de resultados através da técnica SSE

O código apresentado mostra a forma como se obtêm os dados gerados pela execução de código e guardados de forma global pelo servidor. Estes são guardados num dicionário que apenas pode ser acedido pelo identificador da sessão gerado e devolvido pelo pedido de execução de código. O exemplo também mostra o modo como os resultados são devolvidos, tendo de estar em conformidade com as regras da técnica SSE, nomeadamente segundo o formato de mensagens "data: {line}\n\n" e segundo o tipo de conteúdo definido para "text/event-stream".

6.2.4 Implantação do Servidor na Cloud

No seguimento da explicação mais detalhada do servidor auxiliar e das suas características, aborda-se de seguida a sua implantação na Cloud. Como já foi dito, o uso de um ambiente Python permite instalar apenas as dependências necessárias para cada projeto, sendo possível obter um ficheiro com a especificação deste conjunto de dependências. Este ficheiro é importante para a "containerização" do servidor, ou seja, a colocação de todo o

ambiente, dependências e código num ambiente isolado que possa ser utilizado noutros contextos.

Este conceito base sustenta o sistema *Docker*, que foi usado no servidor especificado para possibilitar a sua execução isolada em máquinas onde o *Docker* esteja instalado e configurado. Isto traz várias vantagens, como o manuseamento e implantação facilitada noutro ambiente, mas foi sobretudo concretizado para poder colocar o servidor a correr na Cloud, processo composto por dois passos principais: a utilização do serviço *Artifact Registry* para carregar o servidor em formato de *Container* e a integração com o serviço *Compute Engine* para alocar uma máquina onde o servidor possa correr. Existe uma variedade de tipos de máquinas disponíveis, como é apresentado na figura 6.1, tirada diretamente da consola da *Google Cloud*.

	Series	Descrição	vCPUs	Memory	Plataforma
○	C3	Desempenho consistente constante	4 - 176	8 a 1.408 GB	Intel Sapphire Rapids
○	C3D	Desempenho consistente e constante	4 - 360	8 a 2.880 GB	AMD Genoa
◉	E2	Computação diária de baixo custo	0.25 - 32	1 a 128 GB	Com base na disponib
○	N2	Equilíbrio entre preço e desempenho	2 - 128	2 a 864 GB	Intel Cascade e Ice Lal
○	N2D	Equilíbrio entre preço e desempenho	2 - 224	2 a 896 GB	AMD EPYC
○	T2A	Cargas de trabalho de escalonamento horizontal	1 - 48	4 a 192 GB	Ampere Altra Arm
○	T2D	Cargas de trabalho de escalonamento horizontal	1 - 60	4 a 240 GB	AMD EPYC Milan
○	N1	Equilíbrio entre preço e desempenho	0.25 - 96	0,6 a 624 GB	Intel Skylake

Figura 6.1: Escolha do tipo de máquina no serviço *Compute Engine*

A opção selecionada na figura representa a escolha tomada, tentando balancear o custo financeiro com o poder computacional, tendo em conta que as operações suportadas pelo servidor auxiliar são raras. A máquina concreta tem o nome de *e2-small*, 2 vCPU, 1 núcleo, 2GB de memória RAM e um disco permanente de 10GB de memória ROM. Esta máquina, reservada apenas para a execução destes dois endpoints, é capaz de suportar um número baixo de pedidos em simultâneo, mas tem capacidade computacional suficiente face às necessidades do sistema. Outra grande vantagem das técnicas abordadas é a possibilidade de alocar mais ou menos recursos computacionais de acordo com as necessidades, mas também de colocar o servidor a correr numa máquina, servidor ou *cluster* locais ou particulares.

6.3 Funcionalidades de Inteligência Artificial

Após as secções referentes às camadas de servidor e de armazenamento, segue-se o próximo conjunto de funcionalidades, que compõem a camada de inteligência artificial. Como é responsável pelas tarefas relacionadas com aprendizagem automática disponibilizadas pelo sistema, é importante tratá-la mais atentamente e introduzir os seus detalhes de implementação.

6.3.1 Ligação à Vertex AI

Como foi referido em capítulos anteriores, as tarefas de inteligência artificial são disponibilizadas com recurso ao serviço *Vertex AI* da Google Cloud. Da mesma forma, e de acordo com a secção referente à arquitetura 5.3, esta camada tem ligação às camadas de servidor, de armazenamento e de servidor auxiliar. Esta subsecção apresenta a forma como as comunicações são estabelecidas, estando dividida em duas subsubsecções que providenciam detalhes, relacionados primeiro com o servidor principal e depois com o auxiliar e o serviço de armazenamento.

6.3.1.1 Camada de Servidor

O contacto entre a camada de servidor e a *Vertex AI* é feito de forma semelhante à mencionada na subsecção 6.1.2. Assim, são usadas as bibliotecas Java disponibilizadas pela Google para que o servidor possa contactar este serviço. Mais uma vez, para que isso possa acontecer, o código do sistema tem de declarar as dependências *Maven* necessárias, utilizando o ficheiro "pom.xml", como é exemplificado em 6.10.

```
<!-- Vertex AI Dependencies -->
<dependency>
    <groupId>com.google.cloud</groupId>
    <artifactId>google-cloud-aiplatform</artifactId>
    <version>3.13.0</version>
</dependency>

<dependency>
    <groupId>com.google.cloud</groupId>
    <artifactId>google-cloud-automl</artifactId>
    <version>2.15.0</version>
</dependency>
```

Listagem 6.10: Dependências XML para os serviços da *Vertex AI*

Depois de feita a declaração de dependências, a utilização das bibliotecas passa pela sua importação nas classes em que serão usadas, pela inicialização do serviço e pela utilização de lógica adicional para manusear os dados e validar as operações. Um exemplo destes passos é apresentado em 6.11, incluindo a importação das bibliotecas e a forma como os serviços são inicializados e usados para realizar a operação de listagem de modelos de aprendizagem automática.

```
import com.google.cloud.aiplatform.v1.Model;
import com.google.cloud.aiplatform.v1.ModelServiceClient;
import com.google.cloud.aiplatform.v1.ModelServiceSettings;
import com.google.cloud.aiplatform.v1.ModelEvaluation;

```

```
6  public List<ModelMetricsReturn> listModels() throws Exception,
       ↪ VertexAIException {
7
8      /*Inicializacao do servico da Vertex AI*/
9    ModelServiceSettings modelServiceSettings = ModelServiceSettings.newBuilder
         ↪ ().setEndpoint(Constants.VERTEXAI_SERVICE_LOCATION + "-aiplatform.
         ↪ googleapis.com:443").build();
10
11     /*Acesso ao servico para iterar pelos modelos*/
12   try (ModelServiceClient modelServiceClient = ModelServiceClient.create(
         ↪ modelServiceSettings)) {
13     String serviceClient = "projects/" + Constants.PROJECT_ID + "/locations/"
           ↪ + Constants.VERTEXAI_SERVICE_LOCATION;
14     List<ModelMetricsReturn> models = new ArrayList<>();
15
16     for (Model model : modelServiceClient.listModels(serviceClient).iterateAll
           ↪ ()) {
17       ModelMetricsReturn modelReturn = new ModelMetricsReturn();
18       for (ModelEvaluation modelEvaluation : modelServiceClient.
             ↪ listModelEvaluations(model.getName()).iterateAll()) {
19         /*Codigo que digere os modelos e os adiciona a lista "models"*/
20       }
21     }
22
23     return models;
24   } catch (Exception e) {
25     throw new VertexAIException(e.getMessage());
26   }
27 }
```

Listagem 6.11: Utilização dos serviços da *Vertex AI* para listagem de modelos de aprendizagem

O exemplo anterior é chamado pelo método associado ao *endpoint* de listagem de modelos de aprendizagem, sendo excluída muita lógica que permite processar os resultados e devolvê-los da forma correta ao utilizador. Ainda assim, a forma como os serviços da *Vertex AI* são acedidos neste exemplo estabelece um padrão de utilização para outras funcionalidades importantes, como a invocação de modelos de aprendizagem ou a listagem de *pipelines*.

6.3.1.2 Camadas de Servidor Auxiliar e de Armazenamento

Para além das comunicações com a camada de servidor, a camada do servidor auxiliar também interage diretamente com a camada de inteligência artificial, aquando da execução de código a partir da plataforma. Como será abordado na próxima secção 6.3.2, os *templates* de código guardados na plataforma estão preparados para contactar a camada

de inteligência artificial. Para isso, é usado um mecanismo de autenticação que envolve o uso de variáveis de ambiente e a definição de políticas de segurança através do serviço IAM da Google Cloud. Um excerto de código exemplificativo é apresentado em 6.12.

```
def authenticate():
    global credentials

    service_account_path = os.getenv('GOOGLE_APPLICATION_CREDENTIALS')

    credentials = service_account.Credentials.from_service_account_file(
    ↪ service_account_path)
    scoped_credentials = credentials.with_scopes(['https://www.googleapis.com/
    ↪ auth/cloud-platform'])

    aip.init(credentials=scoped_credentials, project=PROJECT_ID, staging_
    ↪ bucket=BUCKET_URI, location=REGION)

    return scoped_credentials
}
```

Listagem 6.12: Mecanismo de autenticação com o serviço *Vertex AI*

Sempre que o servidor adicional é contactado para executar código validado pelo sistema, o serviço da *Vertex AI* é chamado a realizar as operações nele especificadas. Uma vez que essas operações geralmente conduzem à criação de *pipelines* de aprendizagem, é preciso aceder aos dados guardados na camada de armazenamento. O serviço *Vertex AI* é responsável por esta comunicação, estando configurado com uma conta de serviço no IAM para poder aceder a esses dados e usá-los no treino de modelos de aprendizagem. Este processo conduz a que as camadas troquem dados entre si, possibilitando as funcionalidades requeridas pelo sistema.

6.3.2 Templates de Código

Como foi abordado em secções e capítulos anteriores, a funcionalidade de execução de código garante a criação e treino de *pipelines* de aprendizagem automática. Este processo requer alguns passos e funcionalidades auxiliares que visam carregar código, verificá-lo, instanciá-lo com parâmetros e executá-lo, como vai ser explicado nesta subsecção.

Em primeiro lugar, a API oferece um *endpoint* para carregar código, que está limitado pelo controlo de acessos a administradores ou técnicos informáticos. Isto previne que utilizadores mal-intencionados possam carregar código malicioso e delega a responsabilidade do funcionamento correto a utilizadores credenciados e confiáveis. Além do mais, este código, enviado em formato de *script* ou de *notebook* na linguagem *Python*, é verificado pelo sistema e tem de corresponder a um padrão específico, nomeadamente com uma declaração de parâmetros pré-definida, exemplificada em 6.13.

```
#Pipeline Variables
PARAMETERS = {
    "training_set" : 0.7,
    "validation_set" : 0.15,
    "test_set" : 0.15,
    "model_type" : "MOBILE_TF_LOW_LATENCY_1",
    "gcs_source" : "gs://mlplatform-datasets/Seedlings/Seedlings.csv"
}
```

Listagem 6.13: Definição de parâmetros num template de código

O código é digerido pela lógica do sistema e os seus parâmetros são automaticamente inferidos, sendo guardados na base de dados. Este armazenamento de parâmetros, de forma estruturada, é importante para assegurar a funcionalidade de instanciação do código com parâmetros, permitindo criar conjuntos de parâmetros que podem depois ser substituídos dinamicamente no código. A própria estrutura do *template* implica a utilização destes parâmetros, como é apresentado em 6.14.

```
def run():
    filtered_parameters = {key: value for key, value in PARAMETERS.items() if
    ↪  key not in ['pipeline_name']}

    #Utilizacao de parameter_values para enviar parametros ao pipeline
    job = aip.PipelineJob(
        display_name=PIPELINE_NAME,
        parameter_values=filtered_parameters,
    )

#Pipeline com substituicao dos parametros enviados em parameter_values
@kfp.dsl.pipeline(name=PIPELINE_NAME)
def pipeline(training_set: float,
        validation_set: float,
        test_set: float,
        model_type: str,
        gcs_source: str):

    ds_op = gcc_aip.ImageDatasetCreateOp(gcs_source = gcs_source)

    training_job_run_op = gcc_aip.AutoMLImageTrainingJobRunOp(
        model_type = model_type,
        training_fraction_split = training_set,
        validation_fraction_split = validation_set,
        test_fraction_split = test_set)

    endpoint_op = EndpointCreateOp(...)

```

```
28    ModelDeployOp(...)
```

Listagem 6.14: Definição de um *Pipeline* que utiliza parâmetros pré-definidos

O exemplo mostra a declaração de um *pipeline* e a forma como este usa os parâmetros definidos anteriormente em 6.13, através da variável "filtered_parameters". O código de onde o exemplo foi retirado é bastante mais extenso e implica a utilização de outros parâmetros, omitidos aqui para simplificação.

Este processo garante a modularidade necessária para que os utilizadores decidam a forma como querem guiar o treino e a conceção de *pipelines*, bem como o conjunto de dados que deve ser usado, delegando a substituição de parâmetros para a lógica do servidor. Só um código verificado, com parâmetros adequados e corretamente instanciado pode ser obtido através do sistema e executado no servidor auxiliar. Isto garante a segurança do sistema, ao mesmo tempo que promove a modularidade e a extensibilidade, ao permitir carregar novos templates e aumentar progressivamente a capacidade do sistema ao longo do tempo.

6.3.3 Execução Automática de Código

A execução de código foi abordada brevemente nas subsecções anteriores, sendo agora altura de falar acerca dos seus detalhes de configuração e implementação. Esta funcionalidade é implementada pelo servidor auxiliar que corre na Cloud, sendo preciso contactar o *endpoint* por este disponibilizado para a ativar. Para contactar o *endpoint* introduziram-se duas limitações: o código tem de ser pré-obtido e o utilizador tem de ser um administrador ou técnico informático. Isto permite atuar em duas frentes distintas, por um lado obtendo o código necessário para o poder enviar ao servidor auxiliar, e por outro garantindo que apenas utilizadores confiáveis o podem executar.

Isto é importante porque os *templates* de código instanciados pelo sistema têm o objetivo de criar *pipelines* no serviço *Vertex AI*, uma operação que tem necessariamente custos financeiros associados. Estes podem ser maiores ou menores consoante o tipo de template a executar e a dimensão dos conjuntos de dados de treino, pelo que é essencial prevenir que utilizadores sem conhecimentos na área possam executar tais operações.

Aquando da chamada ao servidor e consequente execução de código, os registos correspondentes ao processo de treino são apresentados ao utilizador em tempo real, como é apresentado na figura 6.2, tirada diretamente da *interface* da plataforma. Se todo o processo decorrer sem problemas, é apresentada a mensagem "*Execution Complete* e é criado um modelo de aprendizagem automática para a tarefa e dados definidos nos parâmetros, que fica disponível na página apropriada e estará pronto para ser invocado em futuras previsões.

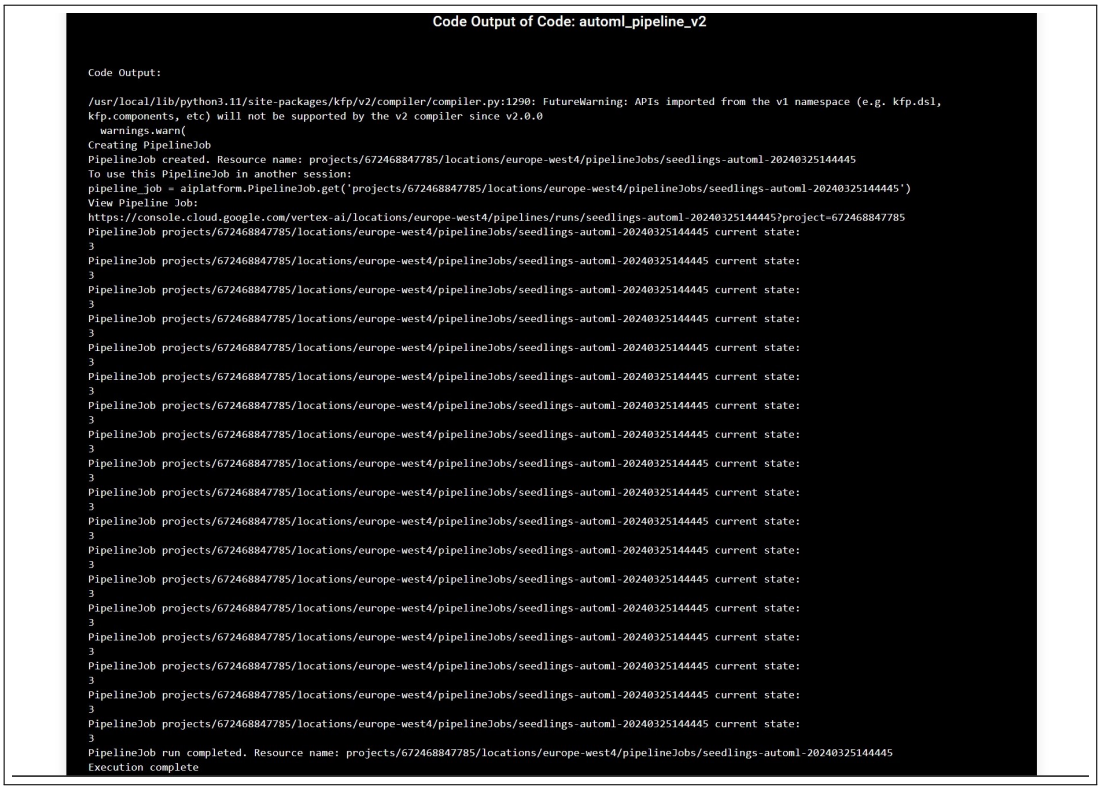


Figura 6.2: Resultado da execução de código para criação e treino de um *Pipeline*

6.4 Camada de Apresentação

Uma vez explicada a grande parte das camadas que compõem a arquitetura do sistema, passa-se agora a alguns detalhes de implementação e utilização da camada de apresentação, desde pormenores do uso da *framework Angular* aos processos de autenticação e gestão de permissões e à forma como as APIs são contactadas.

6.4.1 Estrutura da Aplicação *Frontend*

A *framework Angular* foi apresentada anteriormente neste documento, mais especificamente na subsecção 4.1.4. Contudo, a abordagem foi feita de um ponto de vista teórico, pelo que faz sentido introduzir conceitos e detalhes práticos da sua utilização no sistema desenvolvido.

Durante o desenvolvimento do *frontend* foram seguidas regras e boas práticas relacionadas com o uso da *framework*. Partindo do *template* mencionado em 5.6.1, a estrutura do projeto foi projetada de forma a incluir e organizar os ficheiros necessários para colocar a aplicação a funcionar. Isto inclui os ficheiros de configuração como "*tsconfig.**" ou "*package-lock.**", cuja função é guardar informação relacionada com as bibliotecas usadas e o modo como é feita a compilação da aplicação. Outros ficheiros como o "*main.ts*" ou o "*index.html*" são usados como ponto de entrada da aplicação e para definição do seu sistema de encaminhamento.

Os restantes ficheiros estão relacionados com o conceito central da *framework*, o uso de módulos e componentes. Os primeiros são mais alto-nível e os segundos estão geralmente associados a páginas ou funcionalidades específicas. A aplicação tem cinco módulos principais: o da aplicação, onde se importam bibliotecas e se declaram componentes; o dos elementos de página, que incluem o design da barra de navegação lateral, da barra de navegação e do rodapé; o de utilizadores, que assegura a sua página e respetivas funcionalidades; o de inteligência artificial, que contém os componentes relacionados com modelos e *pipelines*; e o administrativo, que centraliza o uso da SPA para a maior parte das funcionalidades do sistema.

Foi no contexto deste último módulo que se centrou o desenvolvimento da aplicação, dado que o componente deste módulo é responsável pelo formato da SPA, que inclui o módulo dos elementos de página e a responsabilidade de encaminhamento para outros componentes, todos eles representando uma página específica, normalmente associada a um dos recursos da API, organizado da seguinte forma:

- ***{nome_componente}*.component.html** – inclui os elementos de página e define a sua estrutura;
- ***{nome_componente}*.component.scss** – inclui os estilos a aplicar aos elementos de página;
- ***{nome_componente}*.component.spec.ts** – inclui especificações de uso do componente;
- ***{nome_componente}*.component.ts** – inclui a lógica necessária para o correto funcionamento do componente, incluindo normalmente o manuseamento de dados;
- ***{nome_componente}*.service.spec.ts** – inclui especificações de uso do serviço;
- ***{nome_componente}*.service.ts** – inclui a lógica necessária para fazer os pedidos à API e processar as respostas obtidas.

Desta forma, cada componente centraliza a estruturação, estilização e operacionabilidade de uma página, de acordo com o modelo SPA. Além do mais, estabelece o contacto com os serviços para poder fazer pedidos à API, obter os dados das respostas e decidir, ou não, mostrá-los na página.

A navegação entre páginas é garantida por mecanismos de encaminhamento especificados no módulo da aplicação, definindo a forma como cada módulo subsequente deve ser acedido. Neste âmbito, é apresentado o excerto de código abaixo 6.15, retirado do ficheiro "*app-routing.module.ts*" e que apresenta a forma como o módulo administrativo é invocado e outros dois aspetos relevantes: o uso de um *Resolver* para carregar automaticamente conteúdos de forma assíncrona e o uso de uma *Guard*, abordada na próxima subsecção 6.4.2.

```
const routes: Routes = [
    //Outras routes para outras paginas, como login, registo etc.
    {
        component: AdminLayoutComponent,
        canActivate : [LoginGuard],
        resolve: { userProfile: UserProfileResolver },
        children: [
            { path: '',
            loadChildren: () => import('./layouts/admin-layout/admin-layout.
    ↪ module').then(x=>x.AdminLayoutModule) }]
    }
]
```

Listagem 6.15: Definição do encaminhamento para o módulo admistrativo da aplicação

Complementarmente, o componente associado à barra lateral contém a declaração das rotas específicas dos outros componentes, garantindo a sua correta navegação de acordo com a ação do utilizador. Além do mais, são usadas injeções de dependências para que os componentes possam comunicar entre si. A título de exemplo está o componente das notificações, que permite uniformizar o aparecimento de mensagens de erro ou sucesso na *interface* e que é "injetado" e usado em praticamente todos os componentes. Por último, no que toca à gestão do estado e de dados relacionados com a aplicação, foi usada a API *Web Storage*, que permite o armazenamento direto no navegador. Neste tópico, foi ainda desenvolvido um mecanismo de cache de imagens para permitir sejam guardadas algumas imagens no navegador, prevenindo chamadas desnecessárias ao servidor aquando da entrada num repositório de imagens visitado anteriormente.

6.4.2 Autenticação e Autorização

No que toca ao desenvolvimento de aplicações, e esta não é exceção, os mecanismos de autenticação e autorização são essenciais para o bom funcionamento de todo o sistema, pelo que vão ser agora abordados em relação à camada de apresentação.

Neste caso, o processo de autenticação inicia-se sempre que o utilizador tenta executar um login, usando o seu email e a sua password pessoal. Estas credenciais são usadas para fazer um pedido de *login* à API, sendo devolvida uma mensagem de erro em caso de falha, ou *tokens* de acesso em caso de sucesso. Dependendo do caso, o *frontend* apresenta uma notificação de erro ou garante a entrada no sistema, direcionando o utilizador para a página principal e guardando os *tokens* devolvidos utilizando o sistema de *Local Storage* do navegador.

A persistência das credenciais (que não incluem a password, mas sim o email e os *tokens*) evita que o utilizador tenha de efetuar repetidamente a operação de login. Além do mais, o sistema vai verificando a validade do token principal de acordo com o seu prazo de expiração. Caso este tenha expirado, tenta-se validar o token de refrescamento e, caso

este também já tenha expirado, redireciona-se o utilizador para a página de *login*. Tudo isto é garantido com a ajuda do sistema de *Guards*, introduzido na subsecção anterior e apresentado no exemplo 6.15, que proíbe a entrada de utilizadores não autenticados nas páginas do sistema. Este sistema é essencial, uma vez que o uso dos *tokens* é requerido para a execução de todas as outras operações da API.

Por outro lado, o sistema de autorização procura refletir as políticas de controlo de acessos da API. À semelhança do que é implementado para os *tokens*, o sistema guarda o papel de cada utilizador na *Local Storage* e valida-o para as várias funcionalidades do sistema. Caso o utilizador não tenha permissões para realizar determinadas ações, a interface apresenta uma mensagem de aviso apropriada e a lógica subjacente não permite que se efetue o pedido à API.

6.4.3 Comunicação com as APIs

Como foi referido anteriormente, a interface gráfica garantida pela camada de apresentação tem como principal objetivo a facilitação do uso das APIs que garantem as funcionalidades do sistema. Sendo este um aspeto chave, importa detalhar a estrutura do projeto que permite esta comunicação, de que forma esta é feita e quais os seus impactos no sistema e para o utilizador.

Começando pela estrutura do projeto, e em sintonia com o exposto em 6.4.1, os serviços associados a cada componente são responsáveis diretamente pelos pedidos. Desta forma, cada serviço tem um ou mais métodos para cada chamada à API principal, de acordo com o formato apresentado em 6.16. Este exemplo mostra, de forma genérica e omitindo alguns detalhes, como se usam as credenciais no formato "*form-data*", funções das bibliotecas *Angular HTTP* ou *rxjs* e lógica para lidar com a resposta.

```
genericMethod(modelId: string){
    const formData = new FormData();
    formData.append('email', email);
    formData.append('token', token);

    const headers = new HttpHeaders();
    headers.append('Content-Type', 'multipart/form-data');
    const url = this.config.apiEndpointAIModels;

    return new Promise<{ status: boolean, data?: any }>((resolve) => {
      this.http.post(url, formData, { headers, observe: 'response' }).pipe(
        tap((response: HttpResponse<any>) => {
          if (response.status === 200)
            resolve({ status: true, data: response.body });
          else
            resolve({ status: false });
        }),
        catchError((error: any) => {
```

```
19            return of(error);
20          })
21        ).subscribe();
22      });
23  }
```

Listagem 6.16: Exemplo genérico de um pedido à API num serviço

Ainda relativamente ao exemplo, a resposta é processada dentro de uma *Promise*, facilitando o seu tratamento assíncrono. Uma vez que esta função terá de ser invocada pelo componente que contém o serviço, esta solução permite que o processamento da resposta seja feito de forma limpa e reativa. O componente pode lidar com a interface durante o tempo de espera assíncrono, apresentando elementos que mostrem ao utilizador que a interface está à espera da execução do pedido, nomeadamente com o uso de *spinners*. Quando finalizada, o componente avalia o estado da "promessa" e os dados enviados no corpo do pedido, usando-os em lógica subsequente ou apresentando-os ao utilizador.

Este padrão de desenvolvimento promove boas práticas de programação, ao separar claramente as responsabilidades de fazer o pedido e do processamento das respostas, mas também melhora a experiência do utilizador em duas vertentes, por um lado através de feedback imediato e, por outro, com o sistema de notificações, que mostra de forma clara se o pedido foi executado ou não com sucesso.

Por último, importa abordar o contacto com a API do servidor auxiliar. O pedido de execução de código é feito da mesma maneira que no exemplo anterior, incluindo a password guardada num ficheiro dedicado da aplicação e enviando o código para o servidor. Já o pedido para receber os *outputs* é feito a partir de um serviço dedicado, capaz de usar *Observables* para lidar com a tecnologia SSE. Neste caso, são usados objetos do tipo "*Event Source* para abrir conexões unidirecionais persistentes com o *endpoint*, de forma a poder receber o fluxo de dados contínuo que este retorna. Do lado do componente, é feita a análise das respostas obtidas e, caso tudo tenha corrido normalmente, os resultados são apresentados ao utilizador em tempo real.

6.5 Conclusões: Visão Geral do Sistema

Este capítulo e o anterior abordaram a forma como o sistema foi planeado e desenvolvido, passando pelas técnicas usadas e pela forma como se integram para disponibilizar as funcionalidades requeridas. Primeiramente, identificaram-se os requisitos funcionais e os intervenientes no sistema, delineando as suas competências e capacidades. De seguida, foi definida a arquitetura do sistema, composta por várias camadas que interagem entre si, passando depois para a escolha da melhor forma de guardar os dados e pelo desenho da API REST, segundo princípios específicos.

Depois deste planeamento, foi necessário passar à implementação do sistema, descrita e exemplificada neste capítulo, tendo sido descritos vários detalhes da forma como foram

desenvolvidas as diversas camadas e da forma como estas comunicam entre si. Esta análise passou pelas bases de dados, pela camada de servidor, pelas funcionalidades de inteligência artificial e pela integração com o *frontend*, culminando nesta visão geral de todo o processo e do sistema propriamente dito.

A forma como a modelação foi concebida tem dois objetivos principais: em primeiro lugar, garantir a robustez e fiabilidade do sistema e, depois, a extensibilidade e capacidade de suporte para adições e melhorias futuras. O uso de tecnologias *Cloud*, um modelo de dados simples e eficiente, o uso do sistema *Docker* para isolar funcionalidades e uma *interface* intuitiva são aspetos que contribuem para os objetivos traçados.

O sistema é então capaz de responder às diversas solicitações, mas também de suportar a introdução de funcionalidades futuras. O modelo de dados é facilmente estendido com novas entidades e a camada de servidor complementada com novos *endpoints*, estando os mecanismos de segurança já preparados para esta adição de capacidades ao sistema, quer no *backend*, quer no *frontend*. Por outro lado, a flexibilidade assegurada pelo uso de *templates* de código, associada ao isolamento garantido pelo *Docker*, permitem estender as funcionalidades de inteligência artificial de forma simples e segura.

7

Avaliação

Abordados todos os aspetos relevantes do projeto subjacente a esta dissertação, falta agora apresentar a avaliação do sistema resultante. Este capítulo faz essa análise, introduzindo a parte das testagens e as suas facetas, passando depois para testes concretos e conclusões deles retiradas.

7.1 Introdução e Metodologia

Como em qualquer sistema, a avaliação é sempre uma parte importante, quer para identificar erros desconsiderados durante a fase de desenvolvimento, quer para medir a qualidade geral do sistema. Para tal, o processo de avaliação deve reunir um conjunto de abordagens.

Primeiramente, é fundamental fazer uma análise crítica do protótipo final de acordo com os objetivos inicialmente definidos, analisando não só o seu grau de cumprimento, mas também a importância relativa das funcionalidades. Apesar de o sistema ser importante como um todo, há funcionalidades que são marcas distintivas e, portanto, devem ser analisadas mais cuidadosamente, incluindo até uma análise comparativa com outros sistemas e metodologias.

Por outro lado, devem testar-se as funcionalidades do sistema no que toca à sua robustez e usabilidade. Isto significa que, numa situação ideal, se fariam testes pragmáticos a todas elas de forma a avaliar o seu funcionamento e os seus efeitos no sistema. No entanto, o sistema desenvolvido acaba por ter uma dimensão alargada, compreendendo muitos *endpoints* no *backend* e vários casos de uso através do *frontend*. Tudo isto torna o sistema virtualmente impossível de testar para a totalidade de cenários de uso possíveis, pelo que se optou por abordagens alternativas.

Assim, o sistema foi dividido em duas componentes principais: a API, que compõe e disponibiliza as funcionalidades (*backend*), e a interface gráfica para utilizadores (*frontend*). Foram escolhidas metodologias de testagem diferentes para cada uma, dando maior ênfase ao *frontend*, cuja utilização implica invariavelmente também a utilização da API.

Relativamente ao *backend*, a testagem foi feita de três formas distintas. Em primeiro lugar, através de pedidos enviados à API através da ferramenta *Postman*, quer durante a

fase de desenvolvimento, quer em fase de produção. Esta metodologia permitiu testar vários casos de uso à medida que o *backend* estava a ser programado, tendo sido essencial para procurar erros e corrigi-los, repetindo-se este processo até atingir o funcionamento esperado da API. Em segundo lugar, através do desenvolvimento e utilização de alguns testes unitários, capazes de testar as funcionalidades do sistema de uma forma mais pragmática. Em terceiro e último lugar, de forma indireta, aquando da testagem do *frontend*. Os pedidos para o servidor adicional foram ainda testados repetidamente durante a fase de desenvolvimento, funcionando corretamente para os *templates* de código produzidos, mas podendo falhar para *templates* que precisem da instalação de outras dependências.

Isto leva-nos então à componente de avaliação relacionada com a usabilidade e capacidade do sistema do ponto de vista do utilizador, que usa a interface gráfica. Para tal, foram desenvolvidos questionários usando a metodologia de *System Usability Scale* (SUS), que permite fazer a avaliação de acordo com várias métricas que abrangem o seu funcionamento geral. Neste caso, os utilizadores seguiram um guião pré-programado e responderam a um questionário com perguntas pertinentes acerca da usabilidade do sistema.

Dito isto, o restante capítulo procura responder aos pontos levantados nesta secção introdutória, começando com um julgamento crítico da utilidade, usabilidade e extensibilidade da plataforma. Seguidamente, apresentam-se os testes a que o sistema foi sujeito, nomeadamente as consultas à base de dados, os testes unitários e os testes de utilizador, culminando depois nas conclusões que estes permitiram tirar.

7.2 Avaliação Crítica do Sistema

À entrada para o processo de avaliação crítica do sistema, importa voltar aos objetivos iniciais do projeto e compará-los com aquele que acabou por ser o resultado final. Essencialmente, perspetivava-se que os esforços de desenvolvimento se centrassem sobretudo nos seguintes cinco requisitos:

- Disponibilização de funcionalidades relacionadas com utilizadores, nomeadamente gestão de perfis e atribuição de papéis no sistema;
- Existência de um banco de dados capaz de armazenar imagens de forma organizada, no qual se pudessem efetuar anotações classificativas;
- Disponibilização de funcionalidades de aprendizagem automática sobre os dados armazenados e anotados, nomeadamente usando tecnologias estado-da-arte como *Pipelines*, *AutoML* e *Transfer Learning*;
- Disponibilização de uma API que centralizasse as funcionalidades requeridas, protegidas por mecanismos de autenticação e autorização apropriados;
- Disponibilização de uma interface gráfica que pudesse apresentar os conteúdos da API e guiar o utilizador na sua utilização;

Tendo por base os objetivos expostos, é justo dizer que o sistema apresenta todas as funcionalidades principais e responde às necessidades identificadas, trazendo melhorias à tecnologia existente. Contudo, existem ainda aspetos em que o sistema pode ser aperfeiçoado, nomeadamente com adição de novos *endpoints*, melhoria de detalhes na interface ou inclusão de outras funcionalidades de inteligência artificial.

Ainda assim, é possível fazer um balanço positivo do que foi desenvolvido, deixando espaço para melhorias futuras. Com isto em mente, as próximas subsecções fazem a análise crítica das funcionalidades, principalmente daquelas que estão relacionadas com inteligência artificial e acabam por ser marcas distintivas da plataforma. Analisa-se igualmente a aplicabilidade prática do sistema e a forma como este pode ser melhorado e complementado com outras valências.

7.2.1 Usabilidade Comparativa

Como tem sido falado ao longo do documento, e inclusivamente na introdução desta secção, o sistema focou-se nalguns objetivos principais. A complementaridade das funcionalidades relacionadas com bancos de dados, anotação automática e treino/utilização de modelos de aprendizagem automática define a utilidade do sistema.

À entrada do processo de desenvolvimento, foram estudadas algumas plataformas e sistemas com funcionalidades semelhantes às que se pretendiam implementar, as quais apresentavam características muito positivas, nomeadamente no que toca ao design, facilidade de utilização e qualidade dos modelos de aprendizagem. No entanto, o processo de escolha do modelo é uma caixa-negra que não permite perceber as tecnologias que são usadas, impossibilitando a parametrização com vista à melhoria dos resultados. Esta capacidade para saber exatamente o código e a tecnologia usados é um dos pontos diferenciadores do sistema desenvolvido.

No centro destas funcionalidades encontra-se a criação e treino de modelos de aprendizagem automática, utilizando uma combinação das tecnogias de *pipeline* e *AutoML*, que permitem automatizar todo o processo de desenvolvimento em Inteligência e Artificial e têm a possibilidade de parametrização capaz de melhorar a qualidade dos modelos em iterações futuras. Além do mais, o sistema de *AutoML* consegue equilibrar esta parametrização com uma utilização mais casual da Inteligência Artificial, gerando resultados altamente positivos numa quantidade reduzidas de dados.

Com isto em mente, e tendo em conta que esta funcionalidade é suportada por serviços da GCP, importa perceber o que é que o utilizador ganha ou perde com a utilização da plataforma desenvolvida face a serviços semelhantes oferecidos pela UI da Google, enumerando-se os passos necessários para criar um modelo de classificação usando este tipo de abordagem:

I. Criação de uma conta na Google Cloud;

II. Ativação da faturação da conta, acarretando a possibilidade de custos financeiros;

III. Ativação das APIs necessárias, nomeadamente *Vertex AI API* e *Cloud Storage API*;

IV. Criação de um *bucket* na *Cloud Storage* e definição das suas propriedades para visibilidade pública. Este passo é essencial para que a *Vertex AI* tenha acesso aos dados armazenados, mas implica que estes sejam expostos para a internet pública, algo que pode ser indesejável;

V. *Upload* de imagens para a *Cloud Storage*, criando pastas específicas ou de forma complemente desordenada;

VI. Anotação das imagens, uma a uma, usando um ficheiro "csv". Este é um dos passos mais demorados, uma vez é necessário associar a localização de cada imagem na *Storage* a uma etiqueta. Além de ser um processo penoso e sujeito a erros, é também tanto mais demorado quanto mais dados sejam usados. Isto implica que existe uma complexidade acrescida caso se usem várias dezenas ou centenas de imagens, sobretudo caso se pretendam usar pastas para organizar os dados na *Storage*;

VII. Início do processo de treino usando o *AutoML*, nomeadamente carregando o ficheiro da anotação e escolhendo o nome do modelo;

VII. Escolha de parâmetros, em número limitado face a outras abordagens como o uso direto da API, em lugar de usar a UI da *Google Cloud*;

IX. Início do processo de treino, seguido do tempo de espera para que o modelo seja treinado e esteja disponível;

X. Caso tudo corra normalmente, implantação do modelo resultante num *endpoint* para que este possa ser invocado para previsões.

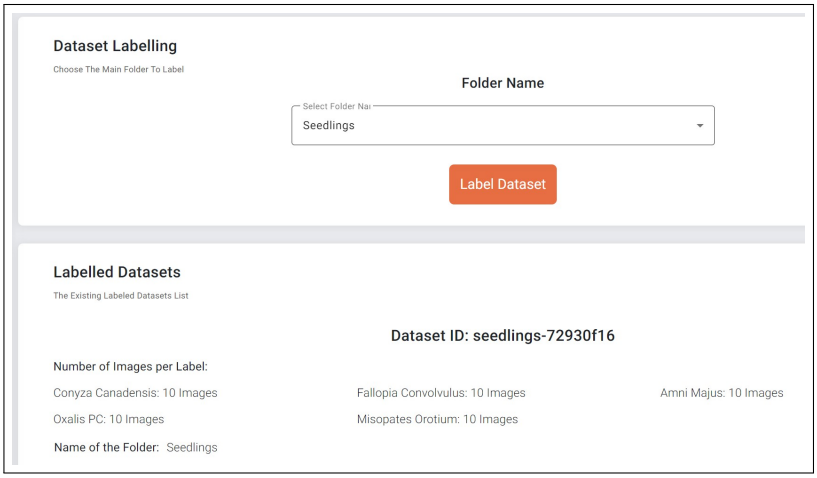
Por outro lado, faz-se a enumeração dos passos necessários para executar a mesma tarefa na plataforma desenvolvida, para um utilizador já inscrito e com permissões de acesso à funcionalidade. É apresentado o descritivo por extenso, seguido de um conjunto de três figuras 7.1 demonstrativas dos três últimos passos na interface da plataforma:

I. *Upload* de imagens para a plataforma, tendo em conta que o nome da pasta em que as imagens são colocadas se deve referir à classe da imagem. Isto implica que se criem pastas e se faça o carregamento das imagens de forma adequada;

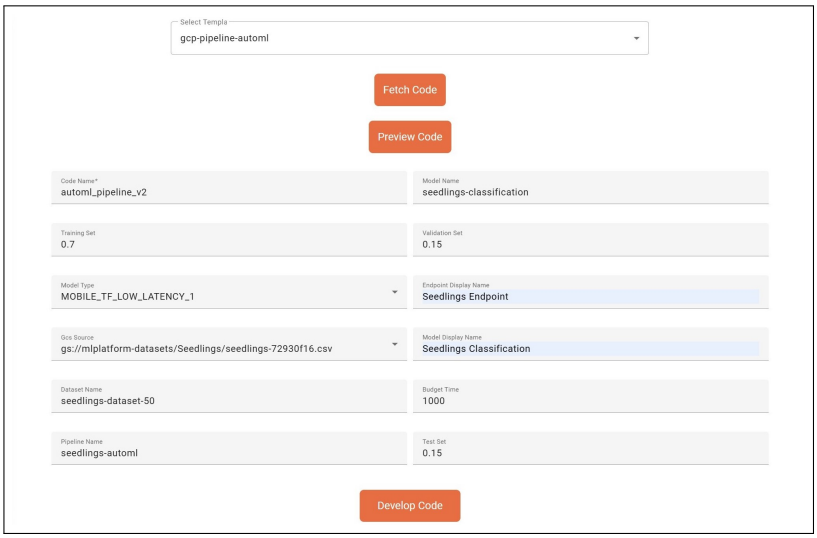
II. Anotação automática de um conjunto de imagens com base no nome das pastas e subpastas em que estas estão contidas. Para tal, basta selecionar o nome da pasta e carregar num botão, sendo gerados automaticamente os ficheiros de anotação necessários 7.2.1;

III. Criação de um conjunto de parâmetros associados ao *template* com nome "*gcp-pipeline-automl*", incluindo o conjunto de dados anotado no passo anterior (II) 7.2.1;

IV. Obtenção do código instanciado com o conjunto de parâmetros criado no passo anterior (III) e início da sua execução, observando os seus resultados de saída em tempo real 7.2.1;



(a) **II** - Anotação automática de imagens



(b) **III** - Parametrização de código

(c) **IV** - Execução de código parametrizado

Figura 7.1: Três passos necessários para criar um *pipeline* de aprendizagem automática na plataforma desenvolvida.

A simulação apresentada pretende mostrar a facilidade de todo o processo comparativamente a uma funcionalidade igual que usa diretamente os serviços da GCP. Além de permitir "saltar" vários passos do processo, a plataforma é capaz de executar automaticamente passos morosos, complicados e propensos ao erro. Além do mais, a utilização da API da *Vertex AI* permite incluir simultaneamente mais parâmetros de configuração do que a UI da Google. O facto de haver a configuração de uma conta de serviço para trabalhar na *Vertex AI* permite ainda que o *bucket* não tenha de ter visibilidade para a internet pública, podendo ser configurado para acesso exclusivo através do sistema. Por outro lado, a própria utilização de um *pipeline* garante que o modelo criado fica automaticamente implantado num *endpoint* e pronto a usar, bastando ir à página correspondente da interface gráfica.

Este exemplo concreto pretendeu mostrar a facilidade do uso do sistema para esta funcionalidade específica. No entanto, poder-se-ia ter usado outra funcionalidade, correspondente a outro *template*, levando à introdução do conceito de extensibilidade, que será abordado na próxima subsecção.

7.2.2 Reutilização e Extensibilidade

No seguimento do exposto anteriormente, esta subsecção apresenta a análise crítica da reutilização e extensibilidade do sistema. Como foi falado em capítulos anteriores, um dos objetivos da plataforma é a possibilidade de adição contínua de funcionalidades e modelos de aprendizagem automática pré-treinados. Da mesma forma, pretendia-se desenvolver algo que servisse de arquitetura de base para um sistema semelhante, mas usando outro tipo de implementação ou fornecedor de serviços na Cloud. Neste âmbito da reutilização e extensibilidade dos diversos componentes da arquitetura, podem abordar-se três camadas em pormenor:

- **Camada de Servidor**: estando desenvolvida sobre a arquitetura da *Google* através do serviço *App Engine*, a reutilização de código fica bastante dificultada. O código está organizado de forma intuitiva, com separação de classes de recursos, de implementação, auxiliares e de receção ou retorno de resultados. Isto facilitaria o processo de transição para um servidor local ou para outro prestador de serviços, mas implicaria, ainda assim, uma alteração geral de toda a implementação e uma nova testagem completa. No entanto, e ainda que haja uma dependência direta dos serviços da *Google*, a adição de novas funcionalidades é relativamente simples, bastando seguir a lógica usada para outros pedidos;

- **Camada de Apresentação**: apesar de estar desenhada para suportar especificamente as funcionalidades do sistema desenvolvido, basta substituir o URL do ambiente da aplicação para a poder usar com outra API. Além do mais, é possível reutilizar estilos, elementos e o design geral da interface. No que toca à extensibilidade, haveria

também facilidade em adicionar novas funcionalidades à aplicação, podendo criar-se novas páginas gerando outros componentes e reutilizando a lógica do código já desenvolvido;

- **Camada de Servidor Auxiliar**: esta é a camada mais fácil de reutilizar para outros serviços. A API está desenhada para devolver resultados em tempo real, mas ambos os pedidos, incluindo o pedido de execução de código, podem ser usados por qualquer biblioteca ou ferramenta que permita o envio de pedidos HTTP. Para tal, basta usar o URL do servidor e incluir a password encriptada que este requer para executar a funcionalidade. Seria também fácil estender as capacidades da API com outros pedidos, apesar de ser moroso e técnico todo o processo de testagem local, containerização e implantação na Cloud.

Esta descrição demonstra que o sistema está construído para poder incorporar novas funcionalidades. Esta extensibilidade torna fácil a adaptação de todo o sistema para outros casos de uso e a adição de novos pedidos. Nomeadamente no que toca a funcionalidades de inteligência artificial, o desenho da arquitetura permite que sejam criados novos *templates* de código capazes de incluir cada vez mais possibilidades de exploração da API da *Vertex AI*. Neste caso, todas as dependências necessárias para o fazer estão já incluídas no servidor auxiliar.

Por outro lado, a forma como este servidor adicional é construído deixa a possibilidade de o redesenhar e instalar novas dependências, a fim de incluir outras funcionalidades de inteligência artificial que usem APIs diferentes. Do mesmo modo, podem-se integrar outras bibliotecas e possibilitar a criação e treino de modelos/*pipelines* independentes da Cloud e das suas APIs. Por último, é possível utilizar todo este mecanismo para implantar modelos já treinados, em que haja um conjunto de parâmetros definido e uma estrutura de modelo que seja capaz de os usar.

7.3 Testes com Utilizadores

Com vista a testar o sistema quanto à sua usabilidade, foram realizados testes com utilizadores. Com este objetivo, foi desenvolvido um guião de utilização, seguido de dois questionários, o primeiro relativo ao SUS e o segundo com perguntas específicas acerca das funcionalidades testadas. Tudo isto foi incluído no terceiro apêndice deste documento C.

Esta secção explica a formulação do guião de utilização, apresenta e explica ambos os questionários e analisa os resultados finais obtidos.

7.3.1 *System Usability Scale*

O método de avaliação *System Usability Scale* tem por base um questionário de 10 perguntas genéricas que medem diferentes aspetos de usabilidade de um sistema. A resposta a cada uma das perguntas é dada numa escala de *Lickert*, com valores de um a cinco, variando de

discordância total a concordância total, respetivamente. Para verificar se o utilizador leu corretamente as questões, as perguntas com número ímpar representam aspetos positivos e aquelas com número par representam aspetos negativos.

A pontuação do SUS varia entre zero e 100, ajustada de acordo com a natureza ou tom de cada uma das perguntas. Desta forma, às afirmações ímpar subtrai-se o valor "um" à resposta obtida, e nas par subtrai-se a resposta obtida ao valor "cinco". Depois, o valor obtido para cada pergunta é somado e multiplicado por "dois e meio". Assim, caso a pontuação atribuída seja sempre "cinco" nas perguntas com tom positivo e "um" naquelas que têm um tom negativo, o resultado final será de 100. Por fim, um sistema é considerado acima da média caso a pontuação obtida no SUS seja superior a 68.

Por haver a necessidade de atribuir significados aos resultados, a maior parte dos estudos e artigos sugerem sobretudo dois métodos [1,36]. O primeiro interpreta o resultado global segundo o percentil da média de pontuações do SUS e o segundo pretende atribuir uma classificação qualificativa aos resultados. As imagens 7.3 e 7.2 apresentam essas duas alternativas de interpretação de resultados.

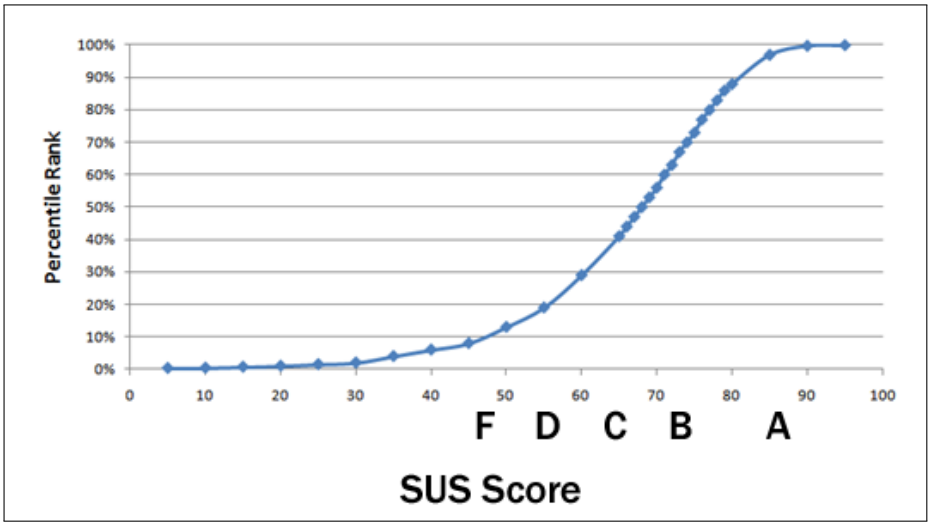


Figura 7.2: Classificação do SUS por percentis (Fonte: [48])

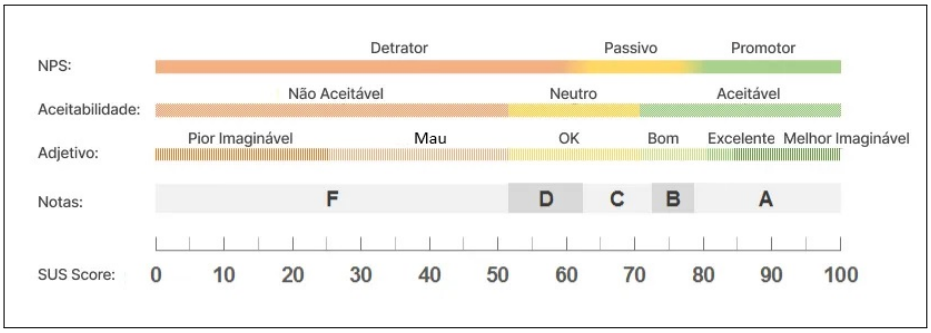


Figura 7.3: Classificação qualitativa do SUS (Fonte: [28])

Analisando a primeira figura 7.2, o valor médio de 68 encontra-se no percentil 50,

ou seja, esse valor permite, em média, estar acima de 50% dos resultados ao SUS. Por outro lado, o mesmo valor de 68, quando analisado segundo a metodologia da segunda figura 7.3, permite inferir uma nota de "C", tendo um resultado neutro nessa escala.

7.3.2 Formulação do Guião e dos Questionários

No seguimento da introdução feita ao sistema de avaliação SUS, fala-se agora da formulação do guião e dos questionários subsequentes.

O guião foi desenvolvido com o intuito de experimentar as principais funcionalidades do sistema de uma forma lógica e metódica. Assim, a forma como foi formulado procura seguir uma cadeia de ações específicas capazes de testar o sistema como um todo. O guião encontra-se no anexo C, mas importa fazer aqui um resumo das ações que pretendeu demonstrar. Para cada uma é atribuído um identificador que associa a letra "**A**" à ordem pela qual a ação é executada:

- **A_1.** Registo na plataforma;
- **A_2.** *Login* na plataforma;
- **A_3.** Acesso ao repositório público e pastas correspondentes;
- **A_4.** *Upload* de imagens para o repositório público;
- **A_5.** Visualização de anotações de conjuntos de dados;
- **A_6.** Anotação de uma pasta presente no repositório público;
- **A_7.** Parametrização de código;
- **A_8.** Obtenção e visualização de código parametrizado;
- **A_9.** Visualização de *pipelines*;
- **A_10.** Visualização de modelos e métricas de avaliação;
- **A_11.** Invocação de modelos de classificação;
- **A_12.** *Logout* da plataforma.

Do guião foram excluídas algumas funcionalidades menos importantes, com o objetivo de não confundir os utilizadores com ações que não fossem estritamente necessárias. A título de exemplo está a utilização do repositório privado, criação de pastas e o acesso e alteração de dados do perfil de utilizador.

O questionário SUS, respondido imediatamente após a realização do guião, é composto pelas 10 perguntas padrão que são utilizadas neste sistema. Como referido em 7.3.1, as perguntas visam testar a usabilidade geral do sistema do ponto de vista do utilizador. A cada uma das perguntas é atribuído um identificador que associa a letra "**SUS**" à ordem da pergunta:

- **SUS_1.** Acho que gostaria de utilizar este produto com mais frequência.

- **SUS_2.** Considerei o produto mais complexo que o necessário.
- **SUS_3.** Achei o produto fácil de utilizar.
- **SUS_4.** Acho que necessitaria de ajuda de um técnico para conseguir utilizar este produto.
- **SUS_5.** Considerei que várias funcionalidades deste produto estavam bem integradas.
- **SUS_6.** Achei que este produto tinha muitas inconsistências.
- **SUS_7.** Suponho que a maioria das pessoas aprenderia rapidamente a utilizar este produto.
- **SUS_8.** Considerei o produto muito complicado de utilizar.
- **SUS_9.** Senti-me muito confiante a utilizar este produto.
- **SUS_10.** Tive de aprender muito antes de lidar com este produto.

O último questionário, respondido depois do SUS, foi desenvolvido de forma a perceber a facilidade ou dificuldade que os utilizadores tiveram a executar as várias ações do guião. Este é inspirado no sistema de testagem *User Experience Questionnaire* (UEQ), que pretende medir a experiência do utilizador em relação à interatividade com o produto.

Assim, as perguntas correspondem quase de forma exata a estas ações, havendo duas perguntas suplementares para tentar perceber se o sistema é esteticamente agradável e se é rápido e responsivo. Tal como no SUS, fez-se variar o tom da pergunta entre positivo e negativo, para tentar evitar que os utilizadores respondessem ao questionário sem ler devidamente as perguntas. Para cada uma é atribuído um identificador que associa a letra "**ESP**" (de questão específica) à ordem pela qual a ação é executada:

- **ESP_1.** Considero que foi simples aceder às diversas páginas.
- **ESP_2.** Considero difícil aceder aos repositórios e visualizar imagens.
- **ESP_3.** Considero que foi fácil fazer o *upload* de imagens para o sistema.
- **ESP_4.** Considero que foi difícil visualizar as características de conjuntos de anotação.
- **ESP_5.** Considero que foi simples criar um conjunto anotado a partir do nome de uma pasta.
- **ESP_6.** Considero que foi fácil selecionar templates de código.
- **ESP_7.** Considero que foi fácil alterar parâmetros nos *templates* de código.
- **ESP_8.** Considero que foi difícil pré-visualizar o código parametrizado.
- **ESP_9.** Considero pouco intuitiva a visualização dos pipelines treinados na plataforma.
- **ESP_10.** Considero simples visualizar os modelos de classificação presentes na plataforma e as suas métricas.

- **ESP_11.** Considero que foi fácil invocar um modelo de classificação e visualizar os seus resultados.
- **ESP_12.** Considero que a aplicação é esteticamente agradável.
- **ESP_13.** Considero que a aplicação é rápida e responsiva.

Concluindo, o guião e os questionários procuraram explorar a maior parte das funcionalidades do sistema, tentando explicar ao utilizador a sua utilidade global. A série de ações escolhida procurou levar o utilizador por toda a cadeia de passos necessária para treinar um modelo de aprendizagem automática, através do desenvolvimento de um *pipeline*. Os utilizadores puderam enviar dados para o sistema, desencadear a sua anotação automática, parametrizar código e visualizá-lo. Foi-lhes ainda explicado, através do guião, que o desencadeamento do código criado está limitado a certos papéis na plataforma, uma vez que o processo de treino é demorado e pode acarretar custos financeiros elevados. Por último, puderam visualizar modelos treinados previamente e invocá-los para tarefas de classificação de imagens.

7.3.3 Resultados dos Questionários

Esta secção encontra-se dividida em duas subsecções, separando a análise dos resultados dos dois questionários (SUS e questões específicas). São ainda incluídos comentários feitos por alguns dos utilizadores que exploraram certas funcionalidades de forma voluntária, fora do escopo do guião.

Os questionários foram respondidos por um total de 12 participantes, dos quais três eram do género feminino e nove do género masculino. Todos os participantes tinham idades entre os 21 e os 24 anos (com média de 22,8 anos), sendo que praticamente todos referiu ter experiência elevada ou muito elevada na utilização de *websites*. Em contraste, nenhum considerou ter experiência muito elevada na área de inteligência artificial, sendo que sete referiu ter experiência moderada ou elevada, e os restantes cinco referiram ter pouca ou muito pouca experiência nesta área.

7.3.3.1 Resultados do questionário *System Usability Scale*

Através do cálculo mencionado na subsecção 7.3.1, verificou-se que a plataforma obteve uma pontuação média de 81,46 pontos no questionário *System Usability Scale*. Isto significa que se encontra aproximadamente no percentil 90% (segundo 7.2) e tem a classificação qualitativa máxima de "A" (segundo 7.3). Destes resultados, pode inferir-se que o sistema apresentou uma boa usabilidade.

Aprofundando em detalhe, foi possível criar o seguinte gráfico de colunas 7.4 representativo das pontuações médias de cada uma das questões do questionário. Importa ainda salientar que todas as perguntas têm uma pontuação mínima de 0 e uma pontuação máxima de 100.

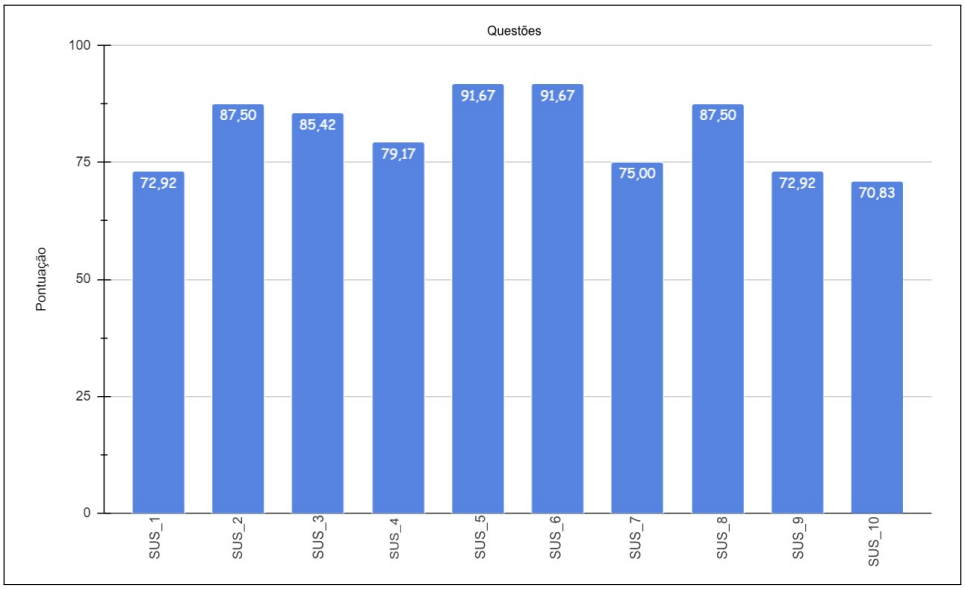


Figura 7.4: Resultados do questionário SUS por pergunta individual

Com base nas respostas a cada questão gerou-se um segundo gráfico de colunas 7.5. Neste apresenta-se a frequência de respostas para cada uma das perguntas caso se adaptassem as perguntas para que correspondessem sempre a tons positivos. Assim, para perguntas ímpares, os valores apresentados são os reais. Para perguntas pares foi considerada uma inversão do tom das perguntas negativas e, portanto, uma simetria das respostas correspondentes. Desta forma todas as respostas com valor "Discordo Totalmente" foram apresentadas como "Concordo Totalmente" e vice-versa. O gráfico apresenta ainda o número de respostas para cada uma dessas categorias, de modo a facilitar a leitura.

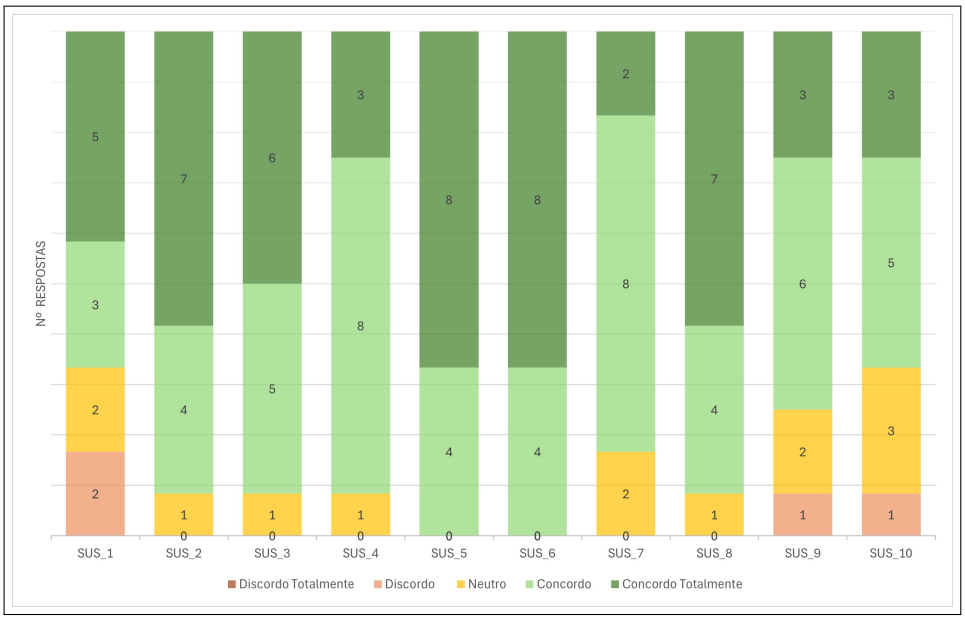


Figura 7.5: Frequência do tipo de respostas para cada pergunta do SUS (adaptando os tons das perguntas)

Pela análise dos gráficos 7.4 e 7.5, é percetível que há claramente questões com resultados mais positivos que outras. No extremo inferior estão as questões "**Tive de aprender muito antes de lidar com este produto.**" (SUS_10), "**Senti-me muito confiante a utilizar este produto.**" (SUS_9), "**Acho que gostaria de utilizar este produto com mais frequência.**" (SUS_1) e "**Suponho que a maioria das pessoas aprenderia rapidamente a utilizar este produto.**" (SUS_7), com pontuações de 70.83, 72.92, 72.92 e 75 respetivamente. Os resultados obtidos significam que os utilizadores penalizaram a quantidade de conhecimento necessária para conseguir experimentar o produto. Isto pode ser explicado pelo facto de, no guião, serem dadas explicações não triviais acerca do propósito do sistema e de alguns dos seus conceitos. Além do mais, este valor pode ser relacionado com o facto de quase metade dos utilizadores terem revelado pouco ou muito pouco conhecimento na área de Aprendizagem Automática, sendo mais difícil compreender a *interface* ou pretender usar o sistema no futuro.

No extremo oposto estão as questões "**Considerei que várias funcionalidades deste produto estavam bem integradas.**" (SUS_5) e "**Achei que este produto tinha muitas inconsistências.**" (SUS_6), ambas com pontuações de 91.67 pontos. Apesar de a questão "SUS_6" apresentar um tom negativo, a sua pontuação é igualmente elevada, uma vez que os utilizadores responderam maioritariamente com as opções "Discordo" e "Discordo Totalmente". Assim, as pontuações de ambas as questões são muito positivas, comprovando que todos os utilizadores conseguiram executar as tarefas que lhes foram sugeridas. Além do mais, compreenderam a forma como todas estas ações se podem integrar para garantir os objetivos do sistema, desde o *upload* de imagens, à anotação de conjuntos e parametrização de *templates*. Por outro lado, os resultados elevados também podem ser explicados precisamente pela utilização de um guião, que apresenta cuidadosamente os passos a ser tomados e explica a sua importância.

Na média estão também alguns resultados curiosos, nomeadamente nas questões "**Considerei o produto mais complexo que o necessário.**" (SUS_2) e "**Considerei o produto muito complicado de utilizar.**" (SUS_8), que obtiveram ambas a pontuação de 87.50 pontos. Estas duas questões têm um sentido similar e mostram regularidade nas respostas, atribuindo pouca complexidade ao sistema. Ainda assim, estes resultados podem ser explicados pelo facto de haver um guião específico que limita as ações tomadas durante a utilização do sistema. Não obstante, validam o facto de que o sistema pode ser usado de forma simples e intuitiva paras as funcionalidades testadas.

Uma análise ao gráfico 7.5 permite ainda compreender que, para a maior parte das perguntas, não houve uma variabilidade grande entre as respostas dadas. Isto significa que os valores se encontram sempre entre respostas neutras ou muito positivas. Não obstante, as perguntas SUS_1, SUS_9 e SUS_10 apresentaram uma variabilidade superior, havendo até alguns resultados negativos tendo em conta o tom da pergunta. Pode depreender-se também que nenhum dos utilizadores assinalou a opção de discórdia total em nenhuma das questões, mostrando que não foram detetadas categorias de usabilidade em que o sistema tivesse falhado totalmente.

7.3.3.2 Resultados do questionário de uso específico da aplicação

Analisados os resultados do questionário *System Usability Scale*, importa agora visualizar e interpretar também os resultados do questionário de uso específico da aplicação. Para tal, foi gerado um gráfico semelhante ao usado na secção anterior para mostrar a frequências das respostas a cada uma das perguntas. Mais um vez, usou-se a mesma estratégia, considerando uma inversão do tom das perguntas negativas e, portanto, uma simetria das respostas correspondentes. O gráfico resultante é então apresentado na figura 7.6.

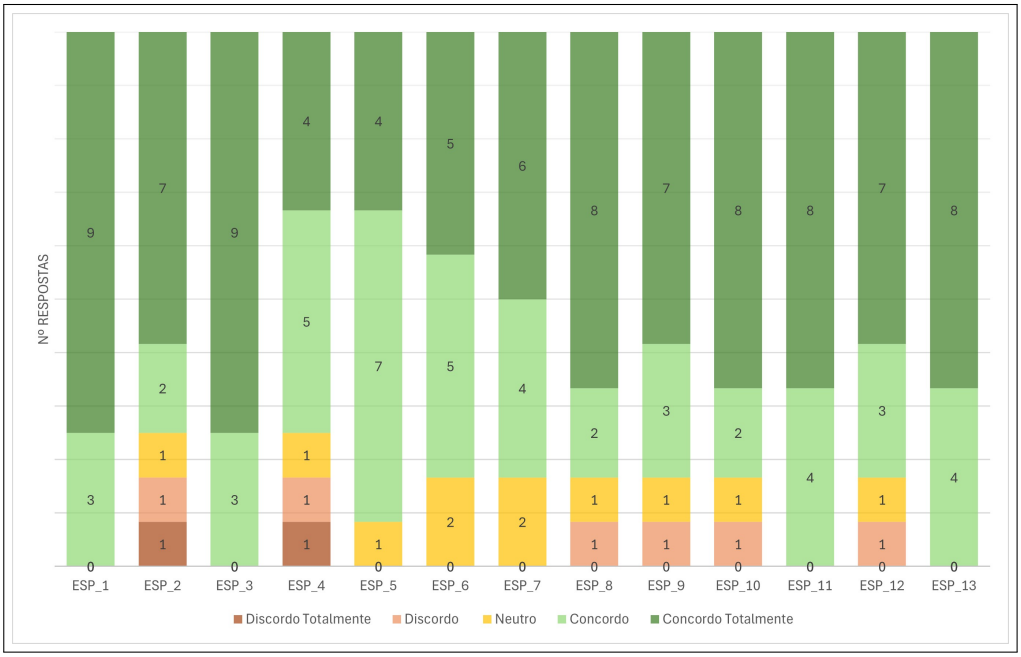


Figura 7.6: Frequência do tipo de respostas para cada pergunta do questionário específico (adaptando os tons das perguntas)

Tal como para o questionário SUS, houve algumas perguntas com respostas claramente melhores que outras. Diretamente a partir da visualização das colunas e cores do gráfico, é fácil notar que quatro das questões tiveram resultados amplamente positivos. Em primeiro lugar, a questão ''Considero que foi simples aceder às diversas páginas.'' (ESP_1) obteve os melhores resultados, transmitindo a ideia de que os utilizadores conseguiram aceder facilmente às diferentes páginas, quer através da ''Dashboard'', quer através da barra lateral. Relativamente a funcionalidades concretas, as questões ''**Considero que foi fácil fazer o *upload* de imagens para o sistema.**'' (ESP_3) e ''**Considero que foi fácil invocar um modelo de classificação e visualizar os seus resultados.**'' (ESP_11) apresentaram também resultados muito satisfatórios, como é apresentado com maior detalhe nos gráficos das figuras 7.7 e 7.8.

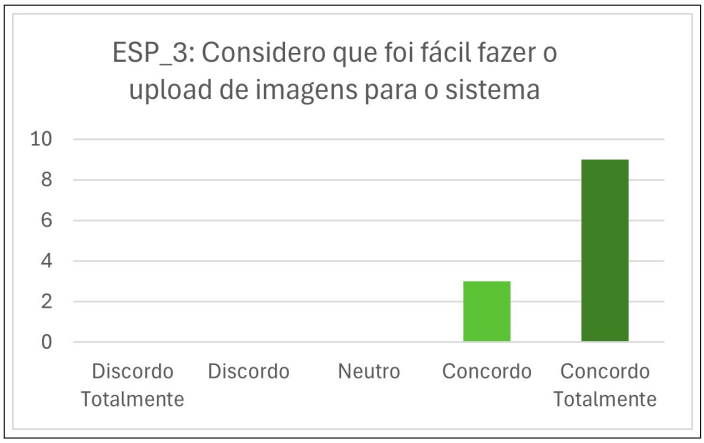


Figura 7.7: Resultados obtidos na questão "ESP_3"

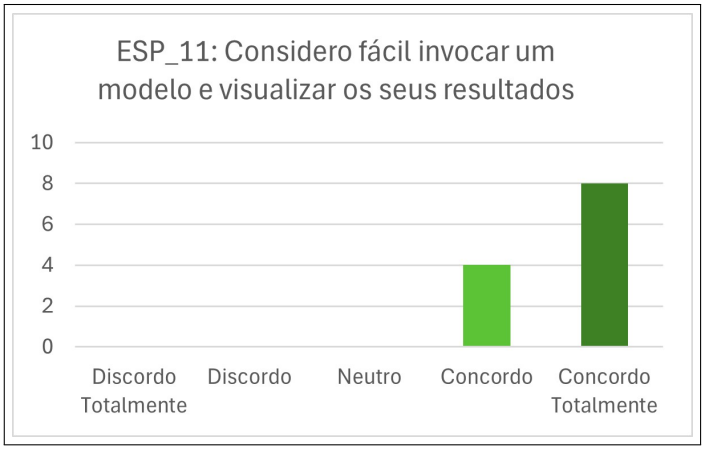


Figura 7.8: Resultados obtidos na questão "ESP_11"

A pontuação do gráfico apresentado na figura 7.7 sugere que os utilizadores tiveram facilidade em carregar imagens para o sistema, mais especificamente para uma das pastas do repositório público. Da mesma forma, o resultado apresentado no gráfico da figura 7.8 é muito positivo no contexto do sistema desenvolvido, ao mostrar que, independentemente da experiência na área de Inteligência Artificial, todos os utilizadores conseguiram invocar o modelo para previsões em tempo real e verificar os seus resultados. Durante este passo, alguns dos utilizadores invocaram voluntariamente os modelos com imagens que tinham nas suas máquinas locais ou imagens que descarregaram a partir da Web no momento. Isto permitiu personalizar a sua experiência de utilização do modelo, bem como testar a sua competência para classificar imagens diferentes das propostas no guião.

Outras questões, como "**Considero que foi simples criar um conjunto anotado a partir do nome de uma pasta.**" (ESP_5), "**Considero que foi fácil selecionar templates de código.**" (ESP_6) e "**Considero que foi fácil alterar parâmetros nos *templates* de código.**" (ESP_7) apresentaram resultados relativamente positivos, de forma consistente. Isto significa que a maior parte dos utilizadores não teve problemas em desempenhar as tarefas mencionadas, confirmando que o uso de selecionadores dinâmicos da biblioteca

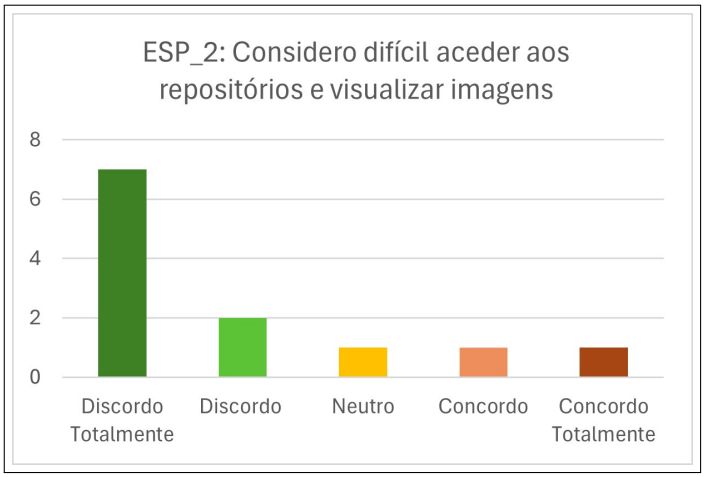


Figura 7.9: Resultados obtidos na questão "ESP_2"

Angular Material UI facilitou a experiência do utilizador.

As questões "**Considero que foi difícil pré-visualizar o código parametrizado.**" (ESP_8), "**Considero pouco intuitiva a visualização dos pipelines treinados na plataforma.**" (ESP_9) e "**Considero simples visualizar os modelos de classificação presentes na plataforma e as suas métricas.**" (ESP_10), referentes à visualização de código, de *pipelines* e de modelos de aprendizagem, apresentaram resultados ligeiramente inferiores e com uma variabilidade superior. Noutras palavras, houve uma quantidade substancial de resultados amplamente positivos e outros medianos ou negativos. Uma possível interpretação para isto é o facto de as perguntas "ESP_8" e "ESP_9" terem um tom negativo, possibilitando que algum dos utilizadores se tenham confundido. No entanto, faz mais sentido interpretar os resultados como uma crítica à forma como estas funcionalidades estão apresentadas. Efetivamente, poderia haver uma melhoria na *interface* das três páginas mencionadas nas perguntas. Um dos aspetos mencionados por um dos utilizadores aquando da passagem pela página dos *pipelines* foi precisamente uma grande quantidade de espaço vazio (ou em branco) entre o identificador do *pipeline* e a sua imagem representativa. Desta forma, a página deveria ser complementada com outras características, além do identificador e do nome.

Ainda que os resultados tenham sido genericamente positivos, as questões "**Considero difícil aceder aos repositórios e visualizar imagens.**" (ESP_2) e "**Considero que foi difícil visualizar as características de conjuntos de anotação.**" (ESP_4) acabaram por apresentar algumas respostas bastante insatisfatórias, como se pode comprovar através dos gráficos das figuras 7.9 e 7.10.

A análise de ambos os gráficos permite inferir que estas são as duas funcionalidades mais inconsistentes do sistema e respetiva *interface*. Apesar de a maior parte das respostas ser positiva, houve algumas respostas muito negativas. Ainda que se possam interpretar como meros enganos, uma análise crítica permite perceber alguns motivos que possam ter levado às respostas obtidas.

Relativamente à visualização de imagens nos repositórios, funcionalidade inquirida na

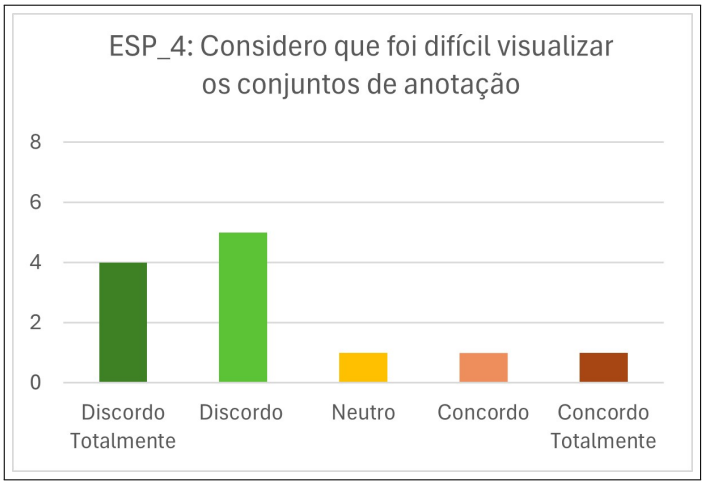


Figura 7.10: Resultados obtidos na questão "ESP_4"

pergunta "SUS_2", houve duas respostas negativas que importa ter em conta. Aquando e após a realização do guião, houve utilizadores que fizeram passos adicionais, carregando noutras pastas. Foi detetado um erro quando um dos utilizadores carregou duas vezes numa mesma pasta, causado pela forma como o *frontend* associa o nome das pastas ao URL do *website*. O comportamento esperado da *interface* neste caso seria ignorar cliques adicionais aquando do carregamento dos conteúdos de uma pasta. Por outro lado, outro dos utilizadores entrou uma pasta com várias imagens de dimensões grandes (média de 4MB), o que tornou o seu carregamento bastante mais demorado que o desejável.

Já no que toca à visualização das características de conjuntos de anotação, funcionalidade inquirida na pergunta "SUS_4", pareceu haver alguma variabilidade nas respostas. Além de terem apresentado mais valores medianos, houve ainda alguns resultados bastante negativos. Estes são compatíveis com a própria natureza do conceito de anotação de conjuntos de dados, sobretudo complexo para utilizadores pouco familiarizados com Inteligência Artificial. Além do mais, esta foi também a página que recebeu maiores críticas ao falar diretamente com os utilizadores, havendo alguns comentários que criticaram a estética da página e o facto de as características dos conjuntos serem apresentadas num formato pouco intuitivo.

Por fim, os resultados obtidos nas questões gerais **Considero que a aplicação é esteticamente agradável.** (ESP_12) e **Considero que a aplicação é rápida e responsiva.** (ESP_13) foram positivos. A partir das pontuações atribuídas pode concluir-se que, genericamente, os utilizadores apreciaram o design e a estética da *interface* gráfica. Além do mais, os resultados da pergunta "ESP_13" mostraram ainda que os utilizadores consideraram o *website* rápido e responsivo, validando não só a rapidez da *interface*, mas também, indiretamente, o baixo tempo de resposta da parte do servidor.

7.4 Conclusões

Este capítulo apresentou vários detalhes de avaliação do sistema, começando por uma análise crítica das suas funcionalidades e extensibilidade, e passando depois para testes de usabilidade prática. Pelo exposto nas várias secções, pode fazer-se um balanço positivo do sistema. A maior parte dos objetivos iniciais foram cumpridos, nomeadamente no que toca ao treino de modelos de aprendizagem automática, utilizando dados específicos, e disponibilizando-os para utilização generalizada.

A plataforma traz aspetos de inovação em relação a outros sistemas relacionados, nomeadamente no que toca à facilidade de utilização de dados, à escolha de parâmetros importantes e à transparência geral das tecnologias usadas para treinar modelos de aprendizagem. Usa ainda a infraestrutura da *Google Cloud Platform* para assegurar a maior parte das funcionalidades, garantindo automaticamente passos adicionais que facilitam a utilização e integração dos seus serviços. A forma como a arquitetura e as funcionalidades estão concebidas asseguram ainda a possibilidade de extensão futura.

Para testar o sistema como um todo foram realizados alguns testes, nomeadamente com utilizadores, de modo a avaliar todas as camadas do sistema. O questionário *System Usability Scale* e o questionário específico das funcionalidades do sistema obtiveram resultados bastante satisfatórios. Os utilizadores conseguiram realizar todas as ações apresentadas no guião de utilização, bem como outras ações que alguns experimentaram de forma autónoma e voluntária. Contudo, as respostas permitiram também perceber que ainda há aspetos a melhorar, tais como a melhoria da *interface* ou a otimização dos tempos de resposta de algumas das funcionalidades da API.

Em suma, esta fase de avaliação garantiu a testagem da maior parte das funcionalidades do sistema e os resultados mostraram que o sistema é capaz de executar os objetivos a que se propõe e que as funcionalidades são robustas, fiáveis e estão bem integradas entre si.

8

Conclusões e Trabalho Futuro

Finalizados todos os capítulos explicativos das tecnologias, modelação, implementação e testagem do sistema, este último capítulo apresenta uma visão de conjunto do que foi desenvolvido. Assim, começa-se por rever os objetivos iniciais e abordar novamente as funcionalidades mais importantes, culminando depois numa proposta de trabalho futuro para melhorias gerais ou específicas do sistema.

8.1 Conclusões

Esta dissertação teve como principal objetivo o desenvolvimento, estudo e testagem de uma plataforma para treinar modelos de Aprendizagem Automática com aplicações na área da Agricultura. Para criar o sistema, foi necessária a aprendizagem de conhecimentos de várias áreas distintas, nomeadamente no que toca a Inteligência Artificial, ao desenvolvimento de *backend*, ao desenvolvimento na Web e ao uso de serviços na Cloud. Foi ainda necessário complementar e integrar as diferentes tecnologias para conseguir obter o protótipo final.

Numa fase prévia à de desenvolvimento, foram definidos uma série de objetivos que um sistema como este deveria cumprir, bem como um plano para os conseguir alcançar. O objetivo mais ambicioso é definitivamente o treino de modelos de aprendizagem, balanceando o uso das melhores tecnologias com a capacidade para poder invocar os modelos em casos reais. Isto levanta uma série de desafios, tais como a escolha do ambiente em que o treino possa ser feito, a integração de dados anotados armazenados no sistema e a parametrização, avaliação e disponibilização dos modelos para uso generalizado através da plataforma.

A solução encontrada tem por base a execução dinâmica de *templates* de código parametrizáveis, complementada com mecanismos de autorização que garantem uma utilização segura dos mesmos. Esta metodologia permite escolher os dados, algoritmos e técnicas de treino que melhor se adaptem a cada situação e tarefa a desempenhar, garantindo a possibilidade para estender as capacidades de treino do sistema no futuro.

Outro dos principais objetivos foi o desenvolvimento de uma API REST capaz de disponibilizar as diversas funcionalidades do sistema, de forma centralizada, segura e

robusta. O *backend* que a assegura é responsável pelo contacto com os serviços da *Google Cloud Platform* e pela sua integração no sistema. Foi ainda desenvolvido e implantado um servidor auxiliar na Cloud, de forma a disponibilizar as funcionalidades de execução de código de forma isolada e segura.

O último grande objetivo centrou-se na disponibilização de uma *interface* através da qual os utilizadores possam utilizar as funcionalidades do sistema. Esta foi desenvolvida e disponibilizada publicamente, na forma de um *website*, para que os utilizadores a pudessem utilizar durante a fase de avaliação. Os resultados dos testes feitos aos utilizadores permitiram perceber que o sistema tem uma boa qualidade, sendo fiável, responsivo e cumprindo de forma adequada uma série de funcionalidades bem integradas.

Concluindo, a maioria dos objetivos desta dissertação foram cumpridos. Foi desenvolvido um sistema completo, incluindo *frontend* e *backend*, capaz de suportar uma série de funcionalidades relevantes na construção de modelos de aprendizagem automática. Isto inclui a capacidade para carregar imagens, organizá-las apropriadamente e anotá-las automaticamente. Estes dados são importantes para treinar os modelos, havendo a possibilidade para escolher os conjuntos de dados anotados e outros parâmetros de treino relevantes. É ainda possível visualizar código parametrizado, os *pipelines* desenvolvidos no sistema e os modelos e métricas resultantes de todo o processo. Por último, o sistema permite a utilização dos modelos treinados no sistema, resultantes da execução dinâmica de código num servidor isolado e protegido.

8.2 Trabalho Futuro

Apesar de a maior parte dos objetivos da dissertação terem sido cumpridos, ainda existem aspetos que podem ser aperfeiçoados ao considerar um trabalho futuro. Num sistema de grande dimensão como aquele que foi desenvolvido, há sempre outras funcionalidades que podem ser adicionadas para aumentar as capacidades do sistema. Isto pode envolver a adição de pedidos à API, a disponibilização de outros pedidos da API do lado do *frontend* ou a extensão das capacidades de utilização de modelos de Aprendizagem Automática.

Por um lado, o sistema poderia beneficiar da possibilidade de gerir *endpoints* para implantação de modelos na Cloud. Para já, os modelos são disponibilizados automaticamente depois de finalizada a execução do *pipeline* que os treinou. No entanto, esta funcionalidade implica custos financeiros alargados quando a implantação é feita por largos períodos de tempo. Para colmatar este problema, seria benéfico ter pedidos na API que conseguissem implantar ou cancelar a implantação de modelos num destes *endpoints*. Da mesma forma, faria sentido aprofundar o estudo e aplicação do mecanismo de *Batch Prediction* para previsões com uma quantidade maior de dados.

Por outro lado, uma vez que o sistema de treino é baseado na execução de código, podia ser benéfico desenvolver mais *templates* capazes de criar modelos direcionados para tarefas de Aprendizagem Automática diferentes. A título de exemplo está a contagem de insetos em armadilhas ou a identificação de frutos em imagens. Ainda nesta temática,

poderia ser interessante fazer a implantação e disponibilização de modelos pré-treinados especificamente para a área da Agricultura.

Por último, poder-se-iam fazer outras melhorias ao nível de engenharia e robustez do *backend* que serve de suporte ao sistema. Neste âmbito, poder-se-ia implementar o uso de *JSON Web Tokens* para tornar o mecanismo de autenticação mais eficiente. Por outro lado, seria útil aplicar mecanismos de *cache* nalgumas funcionalidades mais importantes e otimizar o pedido de obtenção de imagens a partir da API. Por outro lado, faria sentido desenvolver e realizar testes de carga ao servidor, de forma a estudar o seu comportamento em situações de stress, nomeadamente aquelas que envolvem acessos simultâneos de vários utilizadores. Por último, poder-se-ia melhorar a *interface* do ponto de vista estético e de usabilidade, de forma a oferecer uma melhor experiência geral ao utilizador.

Ainda que continue a haver pontos de melhoria, importa destacar que o sistema é capaz de integrar uma série de funcionalidades e serviços para poder efetuar o treino e disponibilização de modelos de Aprendizagem Automática. A forma como foi estruturado teve em conta o objetivo de extensibilidade futura das várias camadas do sistema, oferecendo uma base sólida e fiável para a adição de outras funcionalidades relevantes, nomeadamente no que toca à área de Inteligência Artificial.

Bibliografia

[1] J. M. Aaron Bangor Philip Kortum. «Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale». Em: *Journal of User Experience* 4 (), pp. 114–123. URL: https://uxpajournal.org/determining-what-individual-sus-scores-mean-adding-an-adjective-rating-scale/ (ver p. 114).

[2] R. Akiwatkar. *Top Programming Languages of 2024: A Compilation of Key Statistics*. 2024. URL: https://www.simform.com/blog/top-programming-languages (acedido em 2024-02-16) (ver p. 52).

[3] R. Aljamal, A. El-Mousa e F. Jubair. «A User Perspective Overview of The Top Infrastructure as a Service and High Performance Computing Cloud Service Providers». Em: *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*. 2019, pp. 244–249. DOI: 10.1109/JEEIT.2019.8717453 (ver p. 33).

[4] E. Alpaydin. *Introduction to machine learning*. Second. Cambridge, Massachusetts, London, England: The MIT Press, 2010-02. ISBN: 0-262-01243-X (ver pp. 7, 11, 18).

[5] *Angular Website*. 2024. URL: https://angular.io/ (acedido em 2024-03-04) (ver p. 50).

[6] *Apeer AI Platform*. URL: https://www.apeer.com/home/ (acedido em 2023-02-07) (ver p. 44).

[7] *App Engine*. URL: https://cloud.google.com/appengine? (acedido em 2024-02-06) (ver p. 34).

[8] M. Armbrust et al. «A View of Cloud Computing». Em: *Commun. ACM* 53.4 (2010-04), pp. 50–58. ISSN: 0001-0782. DOI: 10.1145/1721654.1721672. URL: https://doi.org/10.1145/1721654.1721672 (ver pp. 27–29).

[9] *Artifact Registry*. URL: https://cloud.google.com/artifact-registry (acedido em 2024-02-07) (ver p. 37).

[10] *AutoML, NAS and Hyperparameter Tuning: Navigating the Landscape of Machine Learning Automation*. URL: https://towardsai.net/p/machine-learning/automl-nas-and-hyperparameter-tuning-navigating-the-landscape-of-machine-learning-automation (acedido em 2023-02-12) (ver p. 24).

[11] R. Balestriero et al. *A Cookbook of Self-Supervised Learning*. 2023. arXiv: 2304.12210 [cs.LG]. URL: https://arxiv.org/abs/2304.12210 (ver p. 8).

[12] L. Benos et al. «Machine Learning in Agriculture: A Comprehensive Updated Review». Em: *Sensors* 21.11 (2021). ISSN: 1424-8220. DOI: 10.3390/s21113758. URL: https://www.mdpi.com/1424-8220/21/11/3758 (ver p. 25).

[13] C. M. Bishop. *Pattern Recognition and Machine Learning*. First. University of California, Berkeley, USA: Springer Science+Business Media, LLC, 2006. ISBN: 0-387-31073-8 (ver pp. 10, 12, 18, 20).

[14] S. Chiaretta. *Front-end Development with ASP. NET Core, Angular, and Bootstrap*. John Wiley e Sons, 2018 (ver p. 48).

[15] L. Ciampi. *Deep Learning Techniques for Visual Counting*. 2022. DOI: https://doi.org/10.48550/arxiv.2206.03033. URL: https://arxiv.org/abs/2206.03033 (ver p. 10).

[16] *Compute Engine*. URL: https://cloud.google.com/compute? (acedido em 2024-02-06) (ver p. 34).

[17] *Creating production-ready ML pipelines on AWS*. URL: https://docs.aws.amazon.com/prescriptive-guidance/latest/ml-production-ready-pipelines/welcome.html (acedido em 2023-02-07) (ver p. 32).

[18] *Creative Tim: Now UI Dashboard*. URL: https://www.creative-tim.com/product/now-ui-dashboard-angular (acedido em 2024-02-29) (ver p. 72).

[19] J. Duckett. *HTML and CSS: Design and build Websites*. First. 10475 Crosspoint Boulevard Indianapolis: John Wiley e Sons, Inc., 2011. ISBN: 978-1-118-00818-8. URL: https://www.wiley.com/en-us/HTML+and+CSS%3A+Design+and+Build+Websites-p-9781118008188 (ver pp. 48, 49).

[20] R. T. Fielding. «Architectural Styles and the Design of Network-based Software Architectures». Tese de doutoramento. Univesity of California, Irvine, 2000 (ver p. 67).

[21] *Flask WebSite*. URL: https://flask.palletsprojects.com/en/3.0.x/ (acedido em 2024-02-19) (ver p. 53).

[22] M. Flower. *Richardson Maturity Model. steps toward the glory of REST*. URL: https://martinfowler.com/articles/richardsonMaturityModel.html (acedido em 2024-02-27) (ver p. 67).

[23] J. V. Frank Hutter Lars Kotthoff. *Automated Machine Learning*. First. Springer Cham, 2019-05. ISBN: 978-3-030-05318-5. URL: https://link.springer.com/book/10.1007/978-3-030-05318-5 (ver p. 23).

[24] S. Fulton. *What is Google Cloud is and why would you choose it?* 2021. URL: https://www.zdnet.com/article/what-is-google-cloud-is-and-why-would-you-choose-it/ (acedido em 2023-02-07) (ver p. 31).

[25] I. Goodfellow, Y. Bengio e A. Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org (ver pp. 10, 11, 17, 20–22).

[26] *Google Cloud - Datastore*. URL: https://cloud.google.com/datastore (acedido em 2024-02-07) (ver p. 35).

[27] *Google Cloud - Datastore*. URL: https://cloud.google.com/storage (acedido em 2024-02-07) (ver p. 36).

[28] *Guia atualizado de como utilizar a escala SUS (System Usability Scale) no seu produto*. URL: https://brasil.uxdesign.cc/guia-atualizado-de-como-utilizar-a-escala-sus-system-usability-scale-no-seu-produto-ab773f29c522 (acedido em 2023-02-07) (ver p. 114).

[29] X. He, K. Zhao e X. Chu. «AutoML: A survey of the state-of-the-art». Em: *Knowledge-Based Systems* 212 (2021), p. 106622. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2020.106622. URL: http://dx.doi.org/10.1016/j.knosys.2020.106622 (ver p. 24).

[30] J. P. A. Hoekendijk et al. «Counting using deep learning regression gives value to ecological surveys». Em: (2021). ISSN: 2045-2322. DOI: https://doi.org/10.1038/s41598-021-02387-9. URL: https://www.nature.com/articles/s41598-021-02387-9 (ver p. 9).

[31] T. Hunter e S. Porter. *Google Cloud Platform for Developers*. Packt Publishing, 2018. Cap. 1 (ver p. 30).

[77] M. Ibrahim. *A Deep Dive Into Learning Curves in Machine Learning*. URL: https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0#generalization (acedido em 2024-07-02) (ver p. 13).

[32] *Identity and Access Management (IAM)*. URL: https://cloud.google.com/security/products/iam (acedido em 2024-02-08) (ver p. 38).

[33] *Instance Segmentation with Custom Datasets in Python*. URL: https://valueml.com/instance-segmentation-with-custom-datasets-in-python/ (acedido em 2023-02-07) (ver p. 9).

[34] *Introduction to AI Platform*. URL: https://cloud.google.com/ai-platform/docs/technical-overview (acedido em 2023-02-07) (ver p. 31).

[35] *Introduction to AI Platform Pipelines*. URL: https://cloud.google.com/ai-platform/pipelines/docs/introduction (acedido em 2023-02-07) (ver p. 31).

[36] J. S. James R. Lewis. «Item Benchmarks for the System Usability Scale». Em: *Journal of User Experience* 13 (2018), pp. 158–167. URL: https://uxpajournal.org/item-benchmarks-system-usability-scale-sus/ (ver p. 114).

[37] K. Jamsa. *Cloud Computing*. Second. Jones e Bartlett Learning, 2022 (ver p. 28).

[38] K. A. Kern. «Comparing Modern Front-End Frameworks». Bachelor's Thesis to confer the academic degree of Bachelor of Science in the Bachelor's Program Computer Science. Tese de mestrado. Linz, Austria: Johannes Kepler University Linz, 2022-03 (ver p. 51).

[39] Y. LeCun, Y. Bengio e G. Hinton. «Counting using deep learning regression gives value to ecological surveys». Em: (2015). ISSN: 1476-4687. DOI: h10.1038/nature14539. URL: https://doi.org/10.1038/nature14539 (ver pp. 16, 17, 20, 21).

[40] K. G. Liakos et al. «Machine Learning in Agriculture: A Review». Em: *Sensors* 18.8 (2018). ISSN: 1424-8220. DOI: 10.3390/s18082674. URL: https://www.mdpi.com/1424-8220/18/8/2674 (ver p. 25).

[41] *Linear vs Non-linear Data: How to Know*. URL: https://vitalflux.com/how-know-data-linear-non-linear/ (acedido em 2023-02-07) (ver p. 17).

[42] *Linear vs Non-linear Data: How to Know*. URL: https://serokell.io/blog/introduction-to-convolutional-neural-networks (acedido em 2024-03-29) (ver p. 19).

[43] *Lobe AI Platform*. URL: https://www.lobe.ai/ (acedido em 2023-02-07) (ver p. 43).

[44] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/main/template.pdf (ver p. i).

[45] L. R. e. F. N. M.ª Lurdes Inácio. «Nemátodes das galhas radiculares: uma ameaça à produção agrícola». 2018. URL: https://www.vidarural.pt/wp-content/uploads/sites/5/2018/07/aqui.pdf (ver p. 1).

[46] P. Malinina. «How Generative AI will change work in IT in the near future». Bachelor's Thesis. Helsinki: Haaga-Helia University of Applied Sciences, 2023 (ver p. 49).

[47] *Measuring Usability with the System Usability Scale (SUS)*. URL: https://googlecloudcheatsheet.withgoogle.com/ (acedido em 2024-04-01) (ver p. 58).

[48] *Measuring Usability with the System Usability Scale (SUS)*. URL: https://measuringu.com/sus (acedido em 2024-03-31) (ver p. 114).

[49] V. Meshram et al. «Machine learning in agriculture domain: A state-of-art survey». Em: *Artificial Intelligence in the Life Sciences* 1 (2021). ISSN: 2667-3185. DOI: https://doi.org/10.1016/j.ailsci.2021.100010. URL: https://www.sciencedirect.com/science/article/pii/S2667318521000106 (ver p. 25).

[50] M. S. Mikowski e J. C. Powell. *Single Page Web Applications: JavaScript End-to-end*. Manning, 2013. ISBN: 9781617290756. URL: https://books.google.pt/books?id=gGTYswEACAAJ (ver p. 48).

[51] R. Miller. *Cloud infrastructure spending passed on-prem data centers in 2020*. 2021. URL: https://techcrunch.com/2021/03/19/cloud-infrastructure-spending-passed-on-prem-data-centers-in-2020/ (acedido em 2024-03-30) (ver p. 28).

[52] S. Minaee et al. «Image Segmentation Using Deep Learning: A Survey». Em: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.7 (2022), pp. 3523–3542. DOI: 10.1109/TPAMI.2021.3059968 (ver p. 9).

[53] F. Mohr, M. Wever e E. Hüllermeier. «ML-Plan: Automated machine learning via hierarchical planning». Em: *Machine Learning* 107 (2018), pp. 1495–1515. ISSN: 1573-0565. DOI: 10.1007/s10994-018-5735-z. URL: https://doi.org/10.1007/s10994-018-5735-z (ver p. 24).

[54] G. Noer. *Put your archive data on ice with new storage offering*. 2020. URL: https://cloud.google.com/blog/products/storage-data-transfer/archive-storage-class-for-coldest-data-now-available (acedido em 2023-02-07) (ver p. 36).

[55] *Nyckle Platform*. URL: https://www.nyckel.com/ (acedido em 2023-02-07) (ver pp. 40, 41).

[56] N. R. Painia et al. «Global threat to agriculture from invasive species». Em: *SIGPLAN Not.* 113.27 (2016), p. 1. ISSN: 7575-7579. DOI: https://doi.org/10.1073/pnas.1602205113 (ver p. 1).

[57] S. J. Pan. «Transfer Learning.» Em: *Data Classification: Algorithms and Applications* 21 (2014), pp. 537–570 (ver p. 22).

[58] *Postman WebSite*. URL: https://www.postman.com/ (acedido em 2024-02-19) (ver p. 53).

[59] *Python*. URL: https://www.python.org/ (acedido em 2024-02-15) (ver p. 52).

[60] D. R. e S. R. «Cloud providers ranking and selection using quantitative and qualitative approach». Em: *Computer Communications* 154 (2020), pp. 370–379. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2020.02.028. URL: https://www.sciencedirect.com/science/article/pii/S0140366419305067 (ver p. 33).

[61] V. Rajaraman. «Cloud computing». Em: *Resonance* 19 (2014), pp. 242–258. ISSN: 0973-712X. DOI: 10.1007/s12045-014-0030-1. URL: https://doi.org/10.1007/s12045-014-0030-1 (ver p. 30).

[62] I. Rakhmatuiln, A. Kamilaris e C. Andreasen. «Deep Neural Networks to Detect Weeds from Crops in Agricultural Environments in Real-Time: A Review». Em: *Remote. Sens.* 13 (2021), p. 4486. DOI: 10.3390/rs13214486 (ver p. 25).

[63] *React Website*. 2024. URL: https://react.dev/ (acedido em 2024-03-04) (ver p. 50).

[64] I. H. Sarker. «Machine Learning: Algorithms, Real-World Applications and Research Directions». Em: (2021), p. 3. ISSN: 2661-8907. DOI: https://doi.org/10.1007/s42979-021-00592-x. URL: https://link.springer.com/content/pdf/10.1007/s42979-021-00592-x.pdf?pdf=button (ver pp. 6–8).

[65] *SentiSight Platform*. URL: https://www.sentisight.ai/ (acedido em 2023-02-07) (ver p. 42).

[66] *Serviços Oficiais de Inspeção*. URL: https://www.dgav.pt/plantas/conteudo/sanidade-vegetal/inspecao-fitossanitaria/servicos-oficiais-de-inspecao/ (acedido em 2023-02-07) (ver p. 1).

[67] A. Sharma et al. «Machine Learning Applications for Precision Agriculture: A Comprehensive Review». Em: *IEEE Access* 9 (2021), pp. 4843–4873. DOI: 10.1109/ACCESS.2020.3048415 (ver p. 25).

[68] A. M. da Silva Beirão. «Identificação de espécies de Plântulas infestantes». Tese de Mestrado. Universidade NOVA de Lisboa, 2022 (ver p. 23).

[69] S. Skansi. *Introduction to Deep Learning: From Logical Calculus to ArtificialIntelligence*. First. Springer International Publishing AG, part of Springer Nature 2018, 2018-02. Cap. 4. ISBN: 978-3-319-73004-2. URL: https://www.academia.edu/42933956/Introduction_to_Deep_Learning_From_Logical_Calculus_to_Artificial_Intelligence (ver p. 17).

[70] *Smart Farm 4.0*. URL: https://ciencias.ulisboa.pt/pt/smart-farm-4-0 (acedido em 2023-02-07) (ver p. 2).

[71] A. A. Soofi e A. Awan. «Classification Techniques in Machine Learning: Applications and Issues». Em: *Journal of Basic; Applied Sciences* 13 (2017-01), pp. 459–465. DOI: 10.6000/1927-5129.2017.13.76. URL: https://setpublisher.com/pms/index.php/jbas/article/view/1715 (ver p. 9).

[72] *The unit that makes neural networks neural: Perceptron*. URL: https://blog.camelot-group.com/2022/01/neural-networks-perceptron/ (acedido em 2023-02-07) (ver p. 18).

[73] J. F. Trevor Hastie Robert Tibshirani. *The Elements of Statistical Learning*. Second. Springer New York, NY, 2009. Cap. 7. ISBN: 978-0-387-84857-0. DOI: https://doi.org/10.1007/978-0-387-84858-7 (ver p. 13).

[74] *User stories with examples and a template*. URL: https://www.atlassian.com/agile/project-management/user-stories (acedido em 2023-02-07) (ver p. 58).

[75] *Vue.js Website*. 2024. URL: https://vuejs.org/ (acedido em 2024-02-16) (ver p. 50).

[76] R. Vyas. «Comparative Analysis on Front-End Frameworks for Web Applications». Em: *International Journal for Research in Applied Science and Engineering Technology* 10.7 (2022), pp. 298–307 (ver pp. 49, 51).

[78] S. Wensley. *Global Threats to Agriculture*. 2020-07. URL: https://farmtogether.com/learn/blog/global-threats-on-agriculture (acedido em 2023-02-07) (ver p. 1).

[79] S. Wensley. *Prospeção de Pragas e Doenças*. URL: http://www.draplvt.mamaot.pt/alimentacao/Prospecao-pragas-doencas/Pages/Prospecao-pragas-doencas.aspx (acedido em 2023-02-07) (ver p. 1).

[80] *What a Machine Learning Pipeline is and Why It's Important*. URL: https://www.datarobot.com/blog/what-a-machine-learning-pipeline-is-and-why-its-important/ (acedido em 2023-02-07) (ver p. 16).

[81] *What are Azure Machine Learning pipelines?* 2023. URL: https://learn.microsoft.com/en-us/azure/machine-learning/concept-ml-pipelines (acedido em 2023-02-07) (ver p. 33).

[82] *What is a Machine Learning Pipeline?* URL: https://valohai.com/machine-learning-pipeline/ (acedido em 2023-02-07) (ver p. 15).

[83] *What Is Amazon SageMaker?* URL: https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html (acedido em 2023-02-07) (ver p. 32).

[84] *What is Java technology and why do I need it?* URL: https://www.java.com/en/download/help/whatis_java.html (acedido em 2024-02-15) (ver p. 52).

[85] M. Wittig e A. Wittig. *Amazon Web Services in Action*. Second. Simon e Schuster, 2018 (ver p. 32).

[86] A. Zheng. *Evaluation Machine Learning Models*. First. O'Reilly Media, Inc. Sebastopol, CA, 2015-09. ISBN: 978-1-491-93246-9 (ver p. 13).

A

Requisitos Funcionais

Para garantir que os objetivos da dissertação fossem concretizados, foi necessário desenvolver a plataforma para satisfazer os seguintes requisitos funcionais, enumerados abaixo de acordo com grupos de interesse:

- A plataforma assegura o acesso dos utilizadores ao sistema, sendo necessárias as seguintes funcionalidades:
  - Função de *login*, com devolução de tokens de acesso;
  - Função de validação dos tokens de acesso;
  - Função de refrescamento dos tokens de acesso;
  - Função de *logout*, invalidando os tokens de acesso;
  - Registo no sistema, de forma a criar uma conta de utilizador;
  - Consulta de dados da própria conta;
  - Consulta de dados de outras contas, caso seja permitido pelos papéis de ambos os utilizadores;
  - Edição de dados da própria conta;
  - Remoção de contas (por parte do utilizador ou de um administrador, mas sem remover os dados das contas);
  - Suporte de papéis no sistema (administrador, moderador e utilizador normal). Cada papel pode aceder a determinadas funcionalidades específicas do seu nível, sendo que níveis com menos permissões apenas possam alterar as suas próprias contas;
  - Suporte de cargos no sistema (técnico informático ou técnico agrícola). Cada cargo tem funcionalidades específicas;
  - Suporte à atribuição de cargos por parte de utilizadores, sustentada pela hierarquia de papéis e cargos no sistema (por exemplo, apenas um administrador pode promover um utilizador normal a um cargo de pessoal técnico).

- A plataforma assegura vários aspetos relacionados com o armazenamento das imagens no sistema e a sua anotação com etiquetas classificativas:
  - Disponibilização de um repositório de imagens público;
  - Criação automática de um repositório pessoal, aquando do registo no sistema;
  - Criação de pastas para guardar imagens em ambos os repositórios;
  - Remoção de pastas em ambos os repositórios;
  - Inserção de imagens em pastas;
  - Remoção de imagens das pastas. Neste caso, as permissões de remoção variam entre utilizadores. Utilizadores normais apenas podem remover imagens no seu repositório pessoal e a moderadores ou administradores é acrescida a possibilidade de remover imagens do repositório público;
  - Relocação de imagens de umas pastas para outras;
  - Armazenamento persistente de toda e qualquer imagem inserida no sistema;
  - Consulta de imagens dos repositórios público e pessoal;
  - Consulta das pastas presentes nos repositórios público e pessoal;
  - Criação de um conjunto de anotações relativos um grupo de imagens, de forma automática e com base na sua organização nos repositórios;
  - Consulta dos conjuntos de anotações;
  - Edição de um conjunto de anotações;
  - Remoção de um conjunto de anotações.
- A plataforma tem funcionalidades relacionadas com *templates* de código para treino de pipelines e modelos de aprendizagem automática:
  - Criação de *templates* por *upload* de *scripts* ou *notebooks* de código;
  - Consulta do conteúdo dos *templates*;
  - Consulta das meta-informações associadas aos *templates*;
  - Edição das meta-informações dos *templates*;
  - Remoção dos *templates* de código;
  - Inferência automática de parâmetros no código aquando do *upload* de um *template* e criação de uma variável para os guardar separadamente;
  - Criação de conjuntos de parâmetros associados a *templates*, de acordo com os parâmetros inferidos;
  - Consulta de conjuntos de parâmetros;
  - Edição de conjuntos de parâmetros;

  - Remoção de conjuntos de parâmetros;
  - Obtenção de *templates* instanciados com conjuntos de parâmetros;

- A plataforma assegura a possibilidade de utilizar o código instanciado para criar *pipelines* de aprendizagem automática, incluindo as várias componentes por estes asseguradas:
  - Obtenção e visualização do código instanciado;
  - Execução de código previamente obtido;
  - Visualização de resultados associados à execução de código em tempo real;
  - Criação de *pipelines* de aprendizagem, de acordo com parâmetros escolhidos pelo utilizador e com dados guardados no sistema;
  - Criação de modelos de aprendizagem com base na utilização de *pipelines*;
  - Implantação automática de modelos de aprendizagem com base na utilização de *pipelines*.

- A plataforma tem ainda outras funcionalidades relacionadas com *pipelines* e modelos de aprendizagem automática:
  - Consulta de *pipelines* presentes no sistema;
  - Carregamento de imagens associadas a *pipelines*;
  - Obtenção de imagens associadas a *pipelines*;
  - Remoção de imagens associadas a *pipelines*;
  - Consulta de modelos de aprendizagem automática;
  - Consulta das métricas associadas a modelos de aprendizagem automática;
  - Armazenamento de métricas relacionadas com modelos;
  - Invocação de modelos de classificação de imagens e obtenção dos resultados da previsão;

- A plataforma é disponibilizada ao utilizador através de uma interface gráfica que permite executar a maior parte das funcionalidades apresentadas anteriormente.

B

User Stories

Notas:

Sempre que se falam de contas, usa-se o termo utilizador conectado. No entanto, para as restantes funcionalidades, usa-se apenas o termo utilizador pressupondo que este se encontra conectado.

Sempre que se fala em utilizador, incluem-se todos os tipos de utilizador (moderadores e administradores também).

Abaixo enunciam-se as User Stories do sistema, respeitando o seguinte formato:

Como um <tipo de utilizador>, eu quero <objetivo>, (para que <motivo>).

- Como um utilizador anónimo, eu quero poder criar uma conta, para poder utilizar a plataforma;
- Como um utilizador anónimo, eu quero fazer o log-in(conectar), para poder aceder à plataforma;
- Como um utilizador anónimo, eu quero poder manter-me conectado através de tokens de acesso, para não ter de efetuar o log-in sempre que quiser utilizar a plataforma;
- Como um utilizador conectado, eu quero poder fazer o log-out(desconectar), para poder sair da plataforma;
- Como um utilizador conectado, eu quero ver os dados da minha conta, para poder verificar as minhas informações;
- Como um utilizador conectado, eu quero alterar os dados da minha conta, para manter as minhas informações atualizadas;
- Como um utilizador conectado, eu quero eliminar a minha conta, para que eu e outras pessoas deixem de poder aceder à plataforma através da minha conta de acesso;

- Como um moderador/administrador, eu quero eliminar ou banir contas de utilizadores normais, para garantir que apenas quem respeita as condições da plataforma pode continuar a usá-la;
- Como um moderador/administrador, eu quero poder ver a lista de todos os utilizadores da plataforma;
- Como um moderador/administrador, eu quero poder ver informações de outros utilizadores da plataforma;
- Como um moderador/administrador, eu quero promover um utilizador normal a moderador, para poder garantir-lhe mais poderes dentro da plataforma;
- Como um moderador/administrador, eu quero atribuir um cargo de Técnico Agrícola ou de Técnico Informático a um utilizador normal;
- Como um utilizador, eu quero aceder ao repositório público, para poder ver as imagens lá armazenadas;
- Como um utilizador, eu quero inserir imagens no repositório público, para as poder armazenar permanentemente no sistema e de forma a que outros utilizadores lhes possam aceder;
- Como um moderador/administrador, eu quero criar pastas de imagens no repositório público, para poder organizar as imagens de acordo com as suas categorias;
- Como um moderador/administrador, eu quero eliminar pastas de imagens no repositório público, para remover uma determinada categoria de imagens do sistema;
- Como um moderador/administrador, eu quero eliminar imagens do repositório público, para as remover permanentemente do sistema;
- Como um moderador/administrador, eu quero poder mover imagens de umas pastas para outras dentro do repositório público, para que possa manter a sua organização;
- Como um utilizador, eu quero aceder ao meu repositório pessoal, para poder ver as imagens lá armazenadas;
- Como um utilizador, eu quero inserir imagens no meu repositório pessoal, para as armazenar permanentemente no sistema e de forma a que apenas eu lhes tenha acesso;
- Como um utilizador, eu quero criar pastas de imagens no meu repositório pessoal, para organizar as imagens de acordo com as suas categorias;
- Como um utilizador, eu quero eliminar pastas de imagens do meu repositório pessoal, para remover uma determinada categoria de imagens do sistema;

- Como um utilizador, eu quero remover imagens do meu repositório pessoal, para garantir que estas deixam de estar armazenadas no sistema;
- Como um utilizador, eu quero mover imagens de umas pastas para outras dentro do meu repositório pessoal, para que possa manter a sua organização;
- Como um utilizador, eu quero criar um conjunto de anotações relacionadas com as imagens dos repositórios, para poder catalogar as imagens com base na forma como estão organizadas;
- Como um utilizador, eu quero consultar os conjuntos de anotações, para poder verificar as suas características;
- Como um utilizador, eu quero editar um conjunto de anotações, para poder atualizar as suas características;
- Como um moderador/administrador, eu quero eliminar conjuntos de anotações, para remover permanentemente as suas informações;
- Como um administrador ou técnico informático, eu quero carregar *templates* de código para a plataforma, para que possam armazenados e os seus parâmetros extraídos;
- Como um administrador ou técnico informático, eu quero editar *templates* de código, para poder atualizar o seu código e as suas caraterísticas;
- Como um administrador ou técnico informático, eu quero elimnar *templates* de código, para poder remover permanentemente o seu código do sistema;
- Como um utilizador, eu quero visualizar o conteúdo de *templates*, para poder verificar o seu código;
- Como um utilizador, eu quero visualizar as meta-informações de *templates* de código, para poder verificar as suas características;
- Como um utilizador, eu quero criar um conjunto de parâmetros que se adapte a um *template*, para poder parametrizar o seu código;
- Como um utilizador, eu quero visualizar um conjunto de parâmetros, para os poder verificar;
- Como um utilizador, eu quero eliminar um cojunto de parâmetros, para os poder remover do sistema;
- Como um utilizador, eu quero obter um *template* instanciado com um conjunto de parâmetros, para poder ver o código parametrizado da forma que pretendo;

- Como um administrador ou técnico informático, eu quero desencadear a execução de um *template* instanciado com um conjunto de parâmetros, para executar uma tarefa de inteligência artificial;
- Como um administrador ou técnico informático, eu quero visualizar registos relacionados com a execução de código em tempo real, para consultar o seu estado de execução;
- Como um utilizador, eu quero visualizar *pipelines* criados através da execução de código, para poder ver o seu resultado prático;
- Como um administrador ou técnico informático, eu quero carregar uma imagem representativa de um *pipeline*, para que possa ser utilizada como representação gráfica do seu resultado;
- Como um administrador ou técnico informático, eu quero eliminar uma imagem representativa de um *pipeline*, para que a possa remover do sistema;
- Como um utilizador, eu quero visualizar uma imagem de um *pipeline*, para verificar a sua representação gráfica;
- Como um utilizador, eu quero visualizar modelos presentes no sistema, para ver algumas das suas características;
- Como um utilizador, eu quero visualizar as métricas de avaliação de um modelo, para saber detalhes acerca do seu desempenho;
- Como um utilizador, eu quero invocar modelos presentes no sistema, para poder obter previsões sobre os dados de entrada;

C

Formulário de Testagem com Utilizadores

Guião para Testes com Utilizadores

Com o propósito de automatizar e agilizar o processo de desenvolvimento de modelos de inteligência artificial, foi desenvolvida esta plataforma para treinar modelos de aprendizagem automática. Os seus principais objetivos são: o suporte para autenticação e autorização com base em papéis dentro do sistema; a criação, organização e anotação de bancos de dados, nomeadamente de imagens; a gestão, parametrização e execução de código para automatizar o desenvolvimento de modelos, usando o conceito de pipelines de aprendizagem; a utilização de modelos de aprendizagem para as tarefas para as quais foram treinados.

Com isto em mente, a plataforma abordada neste guião de utilização e consequente questionário deve ser testada para a maior parte das tarefas disponibilizadas, focando aspetos de usabilidade e qualidade geral. Para tal, esta sessão está dividida em duas partes: realização de um conjunto de tarefas, seguido de dois questionários acerca das funcionalidades experimentadas. Durante este processo, o utilizador deverá ler as tarefas indicadas pelo guião e executá-las sequencialmente no sistema, respondendo depois às questões apresentadas.

Em caso de dúvidas contacte-me pelo email: j.palmeiro@campus.fct.unl.pt

Obrigado pela Colaboração!

Antes de iniciar o guião de utilização, responda às seguintes questões genéricas:

- **1.** – Selecione o seu Género
- **2.** – Selecione a sua Idade
- **3.** – Selecione o seu Grau Académico
- **4.** – Selecione a sua experiência na utilização de *websites* (1 a 5)

- **5.** – Selecione a sua experiência na utilização de Aprendizagem Automática (1 a 5)

Parte 1 – Guião de utilização

Nota: A plataforma tem imagens organizadas estrategicamente em pastas. Para que as imagens possam ser usadas para treinar modelos de aprendizagem automática, é necessário anotá-las. No contexto do sistema, a anotação corresponde à atribuição de etiquetas classificativas às imagens.

Complete as seguintes operações:

- **1. Obtenção de imagens auxiliares:** Para começar, por favor aceda ao seguinte link da Google Drive: https://drive.google.com/drive/folders/1TSA2fz8RbIdm3KHJ8r205JDBsjnRgFs_?usp=sharing. Aqui encontrará duas imagens que deverá transferir para a sua máquina local e que serão utilizadas durante a sessão.
- **2. Acesso à plataforma:** Aceda ao website através do seguinte link: https://frontend-dot-mlplatform-310324.oa.r.appspot.com/dashboard.
- **3. Criar uma conta e fazer o login:** Acabado de entrar no website, deve ver a página de login. Antes de poder executar essa operação, é necessário carregar no botão que diz "Sign Up", inserir credenciais válidas (password com mais de 8 caracteres e username com um mínimo de 5 caracteres) e pressionar o botão "Sign Up" para confirmar. Caso tudo tenha corrido como expectável, foi redirecionado para a página inicial, onde deverá executar a operação de login para poder entrar no sistema;
- **4. Visualização de pastas no repositório público:** Ao entrar no sistema, deverá poder ver a página inicial ("Dashboard"), com ligações paras outras páginas. Faça scroll down e carregue na opção que diz "Public Repositories". Verifique se existe uma pasta chamada "Seedlings" e carregue sobre ela. Agora deve ver 5 pastas com nomes de plântulas, devendo carregar na primeira, "Amni Majus", e esperar que o sistema carregue as imagens (deve demorar um pouco);
- **5. Upload de imagem para o repositório público:** Carregue agora no ícone que permite adicionar uma fotografia, escolha a imagem da Drive chamada "amni_majus11" e carregue em "Upload". Espere um pouco e verifique se a imagem foi adicionada ao sistema (deve ter ficado na linha de cima);
- **6. Visualização de conjuntos de dados anotados:** Agora, olhe para a barra lateral (caso não consiga ver a barra lateral, faça zoom out até que ela apareça) à esquerda, carregue em "Repositories" e depois em "Datasets". Ao entrar na página correspondente, faça scroll down até conseguir ver a lista de conjuntos de dados anotados e as suas características. Verifique que já existe um conjunto de dados criado para a pasta que visualizou anteriormente ("Seedlings"), em que a cada etiqueta correspondem 10 imagens;

- **7. Anotação de um conjunto de dados:** Depois das verificações do passo anterior, crie uma nova anotação para essa pasta. Deverá selecionar a opção "Seedlings" no selecionador e carregar no botão "Label Dataset". Isto criará automaticamente uma nova anotação classificativa para as imagens contidas nessa pasta, que já inclui a imagem que carregou no passo 5;
- **8. Parametrização de um template de código:** Agora, volte à "Dashboard" através da barra lateral e verifique o seu conteúdo, carregando depois na opção "Development". Já nesta página, deverá conseguir ver uma caixa com a inscrição "Select Template". Carregue nela e selecione a opção "gcp_pipeline_automl". Ao fazer isto, deverá ver abaixo um conjunto de parâmetros. No campo "Code Name" insira "sim_code", em "Validation Set" insira 0.2 e em "Test Set" insira 0.1. Na opção "Gcs Source" selecione uma das opções disponíveis (que inclui a anotação que criou no passo 5.), carregando depois em "Develop Code";
- **9. Visualização de código parametrizado:** Depois do último passo, vá à barra lateral, carregue em "Machine Learning" e depois em "Training". Esta página permite selecionar o código que poderá ser executado no sistema. Para tal, carregue no selecionador e escolha a opção "sim_code", que corresponde ao código que foi parametrizado por si no passo 6. Carregue em "Fetch Code" e, depois de obter a notificação de sucesso, carregue em "Preview Code". O botão "Run Code" apenas está disponível para determinados papéis na plataforma, mas o seu carregamento desencadearia o treino de um modelo para classificação das imagens anotadas no ponto 5, utilizando as tecnologias de pipeline e AutoML;
- **10. Visualização de Pipelines:** Carregue agora na opção "Pipelines" da barra lateral. Esta página apresenta os pipelines criados através desta plataforma, incluindo o seu ID e uma imagem representativa;
- **11. Visualização e utilização de modelos de classificação:** Carregue agora em "Classification" na barra lateral. Será redirecionado para uma página onde poderá visualizar os modelos presentes no sistema e as suas métricas, sendo possível invocá-los para tarefas de classificação. Para tal, encontre o modelo "Seedlings Classification" e carregue em "Choose Image". Isto deverá abrir o seu sistema de ficheiros, onde deverá escolher a imagem "amni_majus12". Carregue depois em "Invoke Model" e verifique o resultado do modelo;
- **12. Logout:** Carregue agora no botão "Logout" na barra lateral esquerda, que o fará sair do sistema. Isto representa o fim deste guião de utilização, obrigado.

Parte 2 – Questionário SUS

- **1** – Acho que gostaria de utilizar este produto com mais frequência.

- **2** – Considerei o produto mais complexo que o necessário.
- **3** – Achei o produto fácil de utilizar.
- **4** – Acho que necessitaria de ajuda de um técnico para conseguir utilizar este produto.
- **5** – Considerei que várias funcionalidades deste produto estavam bem integradas.
- **6** – Achei que este produto tinha muitas inconsistências.
- **7** – Suponho que a maioria das pessoas aprenderia rapidamente a utilizar este produto.
- **8** – Considerei o produto muito complicado de utilizar.
- **9** – Senti-me muito confiante a utilizar este produto.
- **10** – Tive de aprender muito antes de lidar com este produto.

Parte 3 – Questionário Específico do Sistema

- **1** – Considero que foi simples aceder às diversas páginas.
- **2** – Considero difícil aceder aos repositórios e visualizar imagens.
- **3** – Considero que foi fácil fazer o *upload* de imagens para o sistema.
- **4** – Considero que foi difícil visualizar as características de conjuntos de anotação.
- **5** – Considero que foi simples criar um conjunto anotado a partir do nome de uma pasta.
- **6** – Considero que foi fácil selecionar *templates* de código.
- **7** – Considero que foi fácil alterar parâmetros nos *templates* de código.
- **8** – Considero que foi difícil pré-visualizar o código parametrizado.
- **9** – Considero pouco intuitiva a visualização dos pipelines treinados na plataforma.
- **10** – Considero simples visualizar os modelos presentes na plataforma e as suas métricas.
- **11** – Considero que foi fácil invocar um modelo de classificação e visualizar os seus resultados.
- **12** – Considero que a aplicação é esteticamente agradável.
- **13** – Considero que a aplicação é rápida e responsiva.