Área Departamental de Engenharia Eletrónica e
Telecomunicações e de Computadores

Aplicação móvel para a procura ou identificação de plantas e possíveis doenças

HENRIQUE SILVA, 43799
Licenciado

Trabalho de Projeto de natureza científica para obtenção do grau de Mestre em Engenharia Informática e Multimédia

Orientadores:
Prof. Doutor Rui Jesus
Prof. Doutor Gonçalo Marques

Júri:
Prof. Doutor Pedro Mendes Jorge
Prof. Doutor David Semedo
Prof. Doutor Gonçalo Marques

07 de julho de 2023

Resumo

As plantas são uma forma das gerações mais jovens preencherem a necessidade de cultivar, para além de proporcionar benefícios tanto físicos como mentais. As doenças em plantas domésticas são decorrentes, em particular, quando não se tem conhecimento para as prevenir. Para tentar mitigar este problema este trabalho pretende desenvolver uma solução automática e fiável de forma a obter informação sobre uma planta ou providenciar um diagnóstico sobre a mesma. A automatização do trabalho é hoje em dia uma prática cada vez mais frequente em diversas áreas, diminuindo a mão de obra e melhorando os erros comuns ao fator humano. Neste sentido, foi desenvolvida uma aplicação móvel que utiliza técnicas de *Deep Learning*, nomeadamente Redes Neuronais Convolucionais, para a classificação de plantas e possíveis doenças em imagens obtidas com recurso à câmara do dispositivo móvel. Deste modo, foram construídos vários conjuntos de dados de imagens que serviram de base de treino aos modelos usados. Foram ainda feitos estudos de métodos para o pré-processamento e melhoria de imagens. Foram usados modelos para a deteção e classificação de planta, deteção e classificação de doença e deteção se a planta está saudável. Foram usados métodos para a deteção de imagens escuras e desfocadas, nomeadamente, a média e a variância de Laplace, respetivamente. Os modelos foram treinados recorrendo às Redes Neuronais Convolucionais *InceptionResNetV2* e *MobileNetV2*. A primeira rede foi usada para a deteção e classificação das doenças e o resto dos modelos foram implementados utilizando a segunda rede. Foram realizados vários testes para avaliar os vários modelos. Em geral, foi conseguida uma taxa de acertos média de 95% nos modelos. Nos algoritmos de pré-processamento para melhorar as imagens, foi obtida uma precisão de 99% no método para a deteção de imagens escuras e 84.39% para a deteção de imagens desfocadas. Com o algoritmo proposto para detetar e classificar doenças em plantas foi obtida uma precisão de 92.9

Palavras-chave: Deep Learning, Redes Neuronais Convolucionais, Deteção e classificação de doenças em plantas domésticas, Métodos de melhoria de

imagens, *Web Scraping*, Computação móvel

Abstract

Plants are a way for younger generations to fill the need to farm, as well as providing both physical and mental benefits. Diseases in houseplants are particularly prevalent when there is no knowledge to prevent them. To try to mitigate this problem this work aims to develop an automatic and reliable solution to obtain information about a plant or provide a diagnosis about it. The automation of work is nowadays an increasingly common practice in several areas, reducing labor and improving the errors common to the human factor. In this sense, a mobile application was developed that uses Deep Learning techniques, namely Convolutional Neural Networks, for the classification of plants and possible diseases in images obtained using the camera of the mobile device. This way, several image data sets were built and used as a training basis for the models used. Methods for pre-processing and image enhancement were also studied. Models were used for plant detection and classification, disease detection and classification, and healthy plant detection. Methods were used for the detection of dark and blurred images, namely, the Laplace mean and Laplace variance, respectively. The models were trained using Convolutional Neural Networks *InceptionResNetV2* and *MobileNetV2*. The first network was used for disease detection and classification and the rest of the models were implemented using the second network. Several tests were performed to evaluate the various models. Overall, an average accuracy of 95% was achieved on the models. In the preprocessing algorithms for enhancing the images, an accuracy of 99% was obtained in the method for detecting dark images and 84.39% for detecting blurry images. An accuracy of 92.9% was obtained with the proposed algorithm for detecting and classifying diseases on plants.

Keywords: Deep Learning, Convolutional Neural Networks, Disease detection and classification in houseplants, Image enhancement methods, *Web Scraping*, Mobile computing

Agradecimentos

Chegar a esta etapa não teria sido possível sem a ajuda, carinho e dedicação por parte de várias pessoas ao longo da minha formação. Por isso mesmo, não quero perder a oportunidade de agradecer a todos aqueles que, de perto ou de longe, contribuíram para o meu sucesso e a minha chegada até aqui. Quero agradecer a todos aqueles que estiveram presentes nesta minha jornada académica, que me ajudaram a ultrapassar as dificuldades e obstáculos, contribuindo para a minha evolução não só como programador mas como pessoa. Aos meus amigos, aos meus colegas de curso, ao Instituto Superior de Engenharia de Lisboa, aos professores e funcionários o meu muito obrigado.

Um especial agradecimento para os Professores Rui Jesus e Gonçalo Marques, pela paciência e indispensável ajuda na elaboração deste trabalho.

Ao meu colega e amigo Carlos Viegas que fez este percurso comigo, agradeço pela dedicação, empenho e diligência para ajudar quando mais precisava.

Á minha amiga, Ana Patrícia, por toda a paciência, compreensão e apoio nos momentos mais difíceis.

Á minha família, pelo o apoio perante os desafios, pela alegria e distração nos momentos menos bons.

Obrigado a todos que me apoiaram e acreditaram!

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

1D Espaço de uma dimensão, e.g. vetor.

2D Espaço de duas dimensões, e.g. grelha.

API Application Programming Interface.

CNN Convolutional Neural Network.

DFT Transformada Discreta de Fourier.

DL Deep Learning.

DNN Deep Neural Network.

GPU Graphics Processing Unit.

IDFT Transformada Inversa de Fourier.

ILSVRC ImageNet Large-Scale Visual Recognition Competition.

MLP Multilayer Perceptron.

MSE Mean Squared Error.

PDR Plant Diseases Recognition.

PLD Plant Leaf Diseases.

RNN Recurrent Neural Network.

SGD Stochastic gradient descent.

TPU Tensor Processing Unit.

Capítulo 1

Introdução

A classificação e deteção de plantas e doenças são uma importante ferramenta em muitos domínios que vão desde a investigação cientifica até a aplicações comerciais. Nos últimos anos, temos assistido a um enorme crescimento na procura de plantas domésticas [1, 2, 3, 4], particularmente entre os *millennials* e *Gen Z*. Para além dos benefícios físicos e mentais que podem proporcionar, pensa-se que as plantas são uma forma de as gerações mais jovens preencherem a necessidade de cultivar. A tendência das plantas domésticas tem sido alimentada pelos meios de comunicação social, criando uma geração de primeiros proprietários de plantas. Os métodos convencionais de identificação de doenças de plantas, tais como, a inspeção visual pelo o homem, em alguns casos, provaram ser ineficazes, pelo que é imperativo desenvolver técnicas melhoradas de identificação e classificação das doenças de plantas para prevenir potenciais perdas.

Já existem aplicações para a deteção e classificação de diversas plantas e respetivas doenças a partir de imagens, onde são utilizadas técnicas de extração de caraterísticas da planta usando *Deep Neural Network* (DNN). É uma tendência crescente a utilização de técnicas de *Deep Learning* (DL) supervisionadas, por exemplo através de *Convolutional Neural Network* (CNN), para problemas de deteção e classificação onde o homem poderá ter dificuldades ou ser menos eficiente, inspirando assim os investigadores a aprofundar este domínio.

Detetar e classificar plantas e doenças através de imagens pode ser bastante desafiante, uma vez que pode haver vários fatores que introduzem dificuldades. Nomeadamente, a posição da planta para a câmara, a distância com que é capturada a imagem, a imagem não estar focada e a falta de iluminação no momento de captura. O desempenho do sistema de classificação e deteção poderá ser melhorado se estas dificuldades forem ultrapassadas, utilizando, por exemplo, métodos de pré-processamento adequados. Neste

projeto é proposto um sistema para classificação e deteção de doenças em plantas baseado em técnicas de DL que inclui um bloco de pré-processamento para melhorar o modelo. Para melhorar a capacidade de generalização do sistema, isto é, para que o sistema tenha um desempenho consistente em diferentes situações, também são utilizadas técnicas como *transfer learning*, *fine-tuning* e *data augmentation*.

1.1 Descrição do Problema

O objetivo deste projeto é investigar o efeito da qualidade dos dados introduzidos utilizando métodos de pré-processamento no desempenho de um modelo baseado em CNN na deteção e classificação de doenças e plantas. Este estudo dará também uma perspetiva sobre métodos de pré-processamento adequados para melhorar o desempenho de modelos que possam resolver este problema.

1.2 Âmbito e Objetivos

Neste projeto iremos analisar a melhoria do desempenho de modelos baseados em CNN, através do estudo e implementação de métodos de pré-processamento adequados e da investigação da capacidade dos modelos CNN em diferentes ambientes.

Iremos também investigar a implementação de diferentes arquiteturas CNN para resolver o problema na deteção e classificação de plantas e doenças, experimentando diferentes hiper-parâmetros, arquiteturas CNN e técnicas como *transfer learning*, *fine-tuning* e *data augmentation*. E por fim, a implementação de uma aplicação móvel, *HousePlants*, que usará as técnicas estudadas para a resolução do problema em tempo real. Assim, temos os seguintes objetivos principais:

- Estudar e implementar diferentes arquiteturas CNN;
- Implementar diferentes métodos de pré-processamento para melhorar os modelos CNN;
- Implementação de uma aplicação móvel que use os métodos e modelos implementados para a resolução do problema em tempo real.

1.3 Estrutura do documento

Este documento é divido em oito capítulos. No segundo capítulo, será apresentado todo o estudo efetuado para este projeto, começando por uma breve apresentação sobre a aprendizagem automática. Será ainda apresentado um estudo das redes neuronais e das redes neuronais convolucionais, tal como, o estudo de diferentes arquiteturas e as suas comparações. Ainda referente a este capítulo, serão igualmente abordadas as tecnologias e bibliotecas disponíveis, acabando com a apresentação de soluções existentes. No terceiro capítulo é discutida a análise do sistema, onde são abordadas as suas funcionalidades, a arquitetura e o sistema de classificação que usará os métodos de pré-processamento e modelos CNN. No quarto capítulo, serão apresentados os métodos e os modelos que foram estudados e implementados. No quinto capítulo, são discutidos os diferentes conjuntos de dados para os treinos dos métodos. No sexto capítulo será demonstrado todo o processo de implementação da aplicação. No sétimo, será discutido as diferentes metodologias feitas para a avaliação dos resultados, tais como a divisão dos dados, métricas e parâmetros. Será ainda demonstrado o processo de treino dos modelos e resultados na aplicação. Por fim, são apresentadas as conclusões e direções para trabalho futuro.

Capítulo 2

Conceitos e trabalho relacionado

Este capítulo descreve os principais conceitos que dão suporte a todo o trabalho efetuado. Começa com noções de aprendizagem automática, dando ênfase à aprendizagem supervisionada. É também apresentada uma secção dedicada às redes neuronais. Dado que este trabalho é baseado em CNN, é apresentada uma secção dedicada a este tipo de redes, abordando diferentes arquiteturas.

Ainda referente a este capítulo é apresentada uma secção dedicada ao estudo das tecnologias e bibliotecas utilizadas ao longo deste trabalho. O capítulo termina com a apresentação de soluções existentes e relacionadas com os objetivos deste projeto.

2.1 Aprendizagem Automática

A aprendizagem automática é o processo de utilizar modelos matemáticos de dados para ajudar um computador a aprender sem instruções diretas. É considerado um subconjunto da inteligência artificial. A aprendizagem automática utiliza algoritmos para identificar padrões dentro dos dados e esses padrões são depois utilizados para criar um modelo de dados capaz de fazer predições. Com mais dados e experiência, os resultados da aprendizagem automática são mais precisos, tal como os humanos melhoram com a prática.

2.1.1 Aprendizagem Supervisionada

No ramo da aprendizagem supervisionada, os dados de entrada estão previamente etiquetados e são usados para treinar um modelo, geralmente através da minimização de uma função de erro.

As técnicas mais conhecidas para aprendizagem supervisionada incluem regressão linear, regressão logística, redes neuronais artificiais, máquina de suporte vetorial, árvores de decisão e k-vizinhos mais próximos. A aprendizagem supervisionada é uma das áreas mais utilizadas e bem-sucedidas, com problemas bem definidos. As técnicas podem ser agrupadas em classificação e regressão, sendo a primeira utilizada para prever valores discretos (como classes) e a segunda para valores contínuos. Neste trabalho, optou-se pela técnica de classificação.

2.2 Redes Neuronais

A aprendizagem profunda é uma forma especializada de aprendizagem automática que utiliza redes neuronais para dar respostas. As redes neuronais são uma série de algoritmos utilizados para identificar a relação subjacente num conjunto de determinados dados. O processo tenta imitar a forma como o cérebro humano faz a mesma operação. As redes neuronais adaptam-se às alterações de entrada e não requerem intervenção humana.

Uma rede neuronal é uma função não-linear [5] que descreve uma previsão da saída $\hat{y}$ como uma função não-linear das suas variáveis de entrada,

$$\hat{y} = f\left(x_1, \ldots x_p, \theta\right), \tag{2.1}$$

onde a função f é parametrizada [6] por θ. Numa rede neuronal são usados vários modelos de camadas de regressão linear e funções de ativação não lineares. A rede neuronal como um modelo de regressão linear é dado por:

$$\hat{y} = b + \sum_{i=1}^{d} W_i x_i, \tag{2.2}$$

onde, x_i é a entrada da rede, W_i os pesos e b o *bias*. Os pesos e o *bias* são considerados como parâmetros treináveis. Numa forma mais visual, a entrada x_i é representado como um nó e os pesos W_i como uma ligação. O termo *bias* b também será representado como uma ligação, mas à parte.

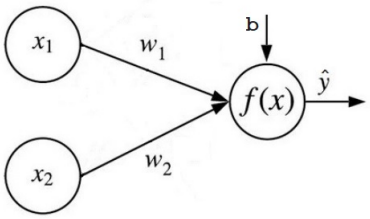


Figura 2.1: Exemplo de uma rede neuronal com duas entradas e uma saída (perceptrão).

Para descrever a relação não linear entre as variáveis de entrada x_i e $\hat{y}$, é utilizada uma função de ativação φ. No caso de se usar uma função de ativação *sigmoid*, o modelo deixa de ser uma regressão linear e passa a ser um tipo de regressão logística.

$$\hat{y} = \varphi\left(b + \sum_{i=1}^{d} W_i x_i\right), \tag{2.3}$$

Uma função de ativação [7] serve para converter valores para uma outra escala e fornece à rede a capacidade de efetuar transformações não-lineares. Assim sendo, a função de ativação terá como objetivo colocar valores para um intervalo especifico, adaptando-se de acordo com as exigências do problema em questão. A rede opera com valores que estejam no mesmo intervalo e assim faz com que treine mais rápido, melhorando o seu desempenho. A função de ativação retorna um número, esse número pode ser 0 ou 1, ou estar entre um intervalo, como por exemplo [0,1], ou [-1,1] ou mesmo estar entre $[-\alpha, +\infty]$, existindo assim várias funções de ativação (Figura 2.2).

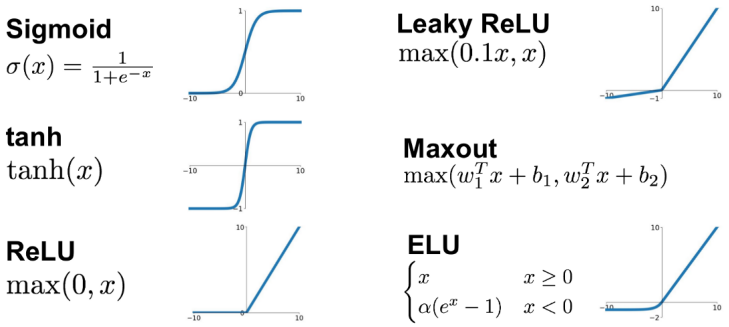


Figura 2.2: Funções de ativação

Normalmente, uma função *sigmoid* [8] é muito usada para tarefas de classificação binária, já que os valores de saída estão entre [0,1]. Já uma função *softmax* [8], é usado para tarefas como classificação multi-classe. Esta equação converte as saídas para probabilidades de cada classe alvo sobre todas as classes alvo possíveis. Onde essas probabilidades calculadas serão úteis para determinar a classe alvo para as entradas dadas. Mas como podemos ver, existem inúmeras funções de ativação que podem ser usadas e que dependem de cada problema.

A Figura 2.3 é um exemplo de uma rede neuronal de três camadas (entrada, escondida e saída) onde a saída $\hat{y}$ é calculada a partir dos neurónios anteriores (Figura 2.3).

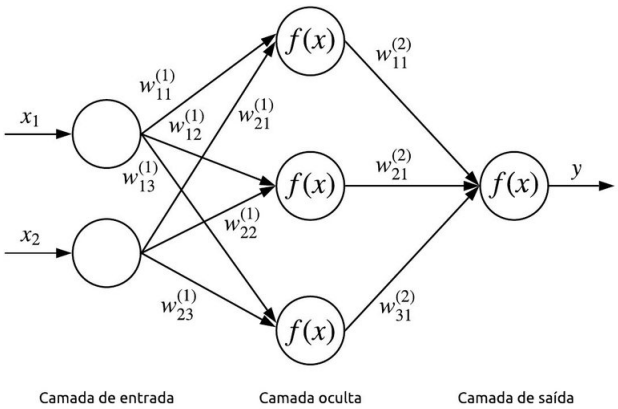


Figura 2.3: Exemplo de uma rede neuronal.

Uma rede neuronal, modifica os seus pesos ao longo de cada iteração através do algoritmo da retro-propagação do erro (*backpropagation*). Tendo

como objetivo, minimizar o erro. Para minimizar o erro são usados métodos de otimização. Dentro desses métodos, temos a Stochastic gradient descent (SGD). Ou seja, o método SGD atualiza os parâmetros de um modelo para minimizar uma função de perda, usando gradientes da função de perda em relação aos parâmetros. O *backpropagation* fornece uma maneira de propagar esses gradientes.

Dentro do domínio das redes neuronais existem diversas variantes, sendo as principais as CNN, as *Multilayer Perceptron* (MLP) e as *Recurrent Neural Network* (RNN).

2.3 Rede Neuronal Convolucional

As *CNN* são um tipo especializado de Rede Neuronal Artificial que conseguem lidar com dados de topologia de grelha. E são popularmente utilizadas quando se trata de dados de imagem (ou seja, grelha 2D de *pixels*). Portanto, as *CNN* são redes neuronais que utilizam convolução, Figura 2.4.

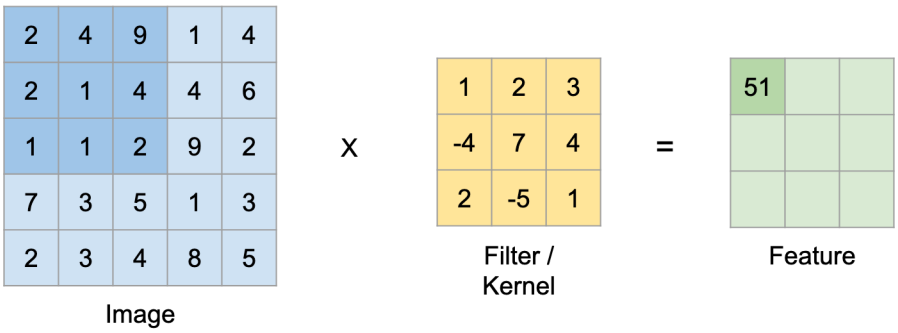


Figura 2.4: Um exemplo de entrada em convolução 2D com um filtro. A figura é adotada a partir de [9].

Ao usar uma rede neuronal de várias camadas, ou seja, uma *MLP*, teríamos de passar a imagem 2D em um vetor 1D, mas a classificação pode não funcionar bem com imagens complexas. A transformação de uma imagem 2D em um vetor 1D pode levar à perda de informações cruciais sobre a disposição espacial e padrões locais na imagem, resultando na dificuldade de capturar eficientemente a estrutura hierárquica e as relações espaciais entre pixeis, essenciais para compreender características complexas como texturas, bordas e padrões. Enquanto a CNN é capaz de obter melhores resultados em relação

aos pixeis (espacial e temporal) através do uso de filtros relevantes que são aprendidos dinamicamente pela rede e que não requerem qualquer seleção manual. Esses filtros servem para detetar certas caraterísticas de um conjunto de imagens. Daí as *CNN* serem amplamente usadas em processamento de visão.

As *CNNs* consistem em várias camadas convolucionais e de *pooling*, seguidas por uma camada densa ou várias camadas densas (*MLP*), dependendo do problema em questão. A primeira camada convolucional extrai as características da imagem, enquanto a camada de *pooling* reduz a dimensionalidade da imagem, ajudando a reduzir o peso computacional. A última camada da *CNN* (*fully-connected*) é responsável pela previsão final da imagem. No entanto, é importante destacar que camadas ou parâmetros desnecessários podem levar a sobre-aprendizagem, tornando a rede inútil para situações reais.

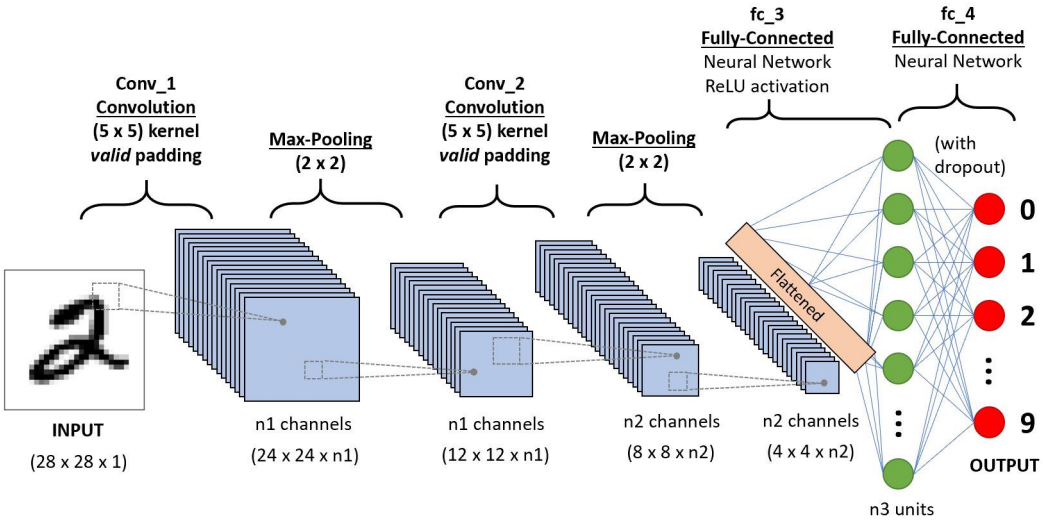


Figura 2.5: Um exemplo de uma arquitetura *CNN* que consiste em duas camadas convolucionais, duas camadas *pooling* e duas camadas densas utilizadas para a classificação. A figura é adotada a partir de [9].

Como se observa na Figura 2.5, temos uma classificação multi-classe, de um conjunto de 10 classes. O quadrado mais à esquerda, que contém um número '2' desenhado, é a imagem de entrada, onde está dividido por quadrados mais pequenos, os pixeis. Seguidamente, tem duas camadas convolucionais intercaladas com camadas *pooling* organizadas em linhas, colunas e

canais ($r \times c \times n$), onde cada quadrado representa uma unidade escondida. A rede termina com duas camadas densas e uma função final (*fully-connected*). A camada *pooling* que está a ser usada é um redimensionamento onde será escolhido o valor mais alto (*max pooling*). A função apresentada é o *softmax*, produzindo as probabilidades de classe como saída. Cada valor agregado após a convolução é transformado usando uma função de ativação ($ReLU$). E ainda podemos ver, que antes das duas camadas densas, os dados foram passados para um vetor 1D (ao que chamamos a essa técnica de *flattening*).

2.3.1 Arquiteturas CNN populares

As CNN são redes neuronais com múltiplas camadas, desenhadas principalmente para reconhecimento e caracterização visual de padrões utilizando por exemplo, pixéis das imagens com algum pré-processamento de imagem. As CNN dispõem de várias arquiteturas possíveis de adotar, destacando *AlexNet*, *GoogleNet*, *ResNet*, *VGGNet* e *MobileNet*. Para além destas, existem outras que serão mencionadas ao longo do trabalho.

AlexNet

A rede *AlexNet* (Figura 2.6) foi projetada pelo grupo *SuperVision*, composto por: Alex Krizhevsky, Geoffrey Hinton e Ilya Sutskever. A arquitetura *Alexnet* é constituída por oito camadas: 5 camadas de convolução e 3 camadas *fully-connected*. *AlexNet* utiliza Unidades Lineares Retificadas ($ReLU$) em vez de *tanh* que era uma função de ativação padrão utilizada anteriormente nas CNNs.

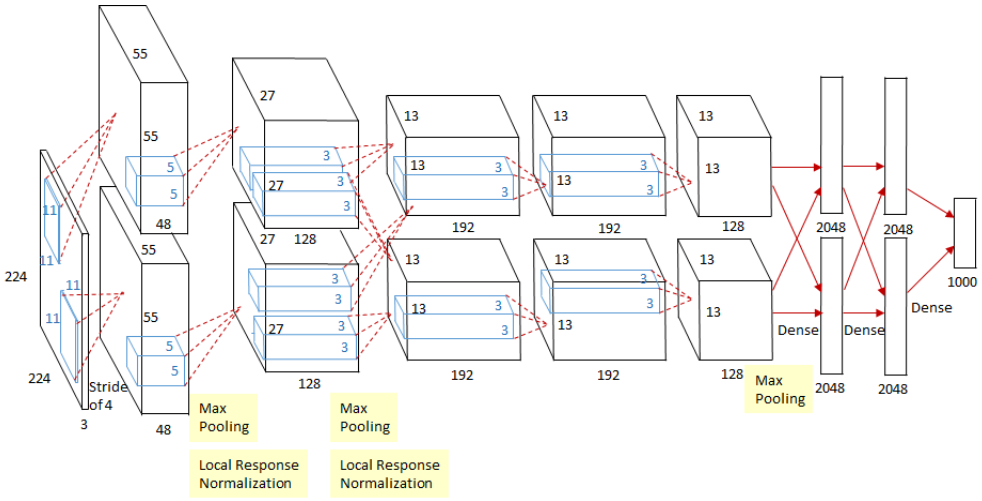


Figura 2.6: Ilustração da arquitetura *AlexNet*. A figura é adotada a partir de [10].

AlexNet tem 60 milhões de parâmetros, e é propensa a entrar em *overfitting*. Os autores desta rede utilizaram técnicas de aumento de dados e *dropout* para superar o problema de *overfitting*. A *AlexNet* pode obter um bom desempenho devido à sua simples estrutura e baixa profundidade comparadas com outras redes.

GoogleNet

A *GoogleNet*, também conhecida como *Inception V1* foi lançado pela Google. A *GoogleNet* usa uma modelo de CNN inspirado por *LeNet* [11, 12], mas implementando um novo conceito, *inception module*. O que reduziu o número de 60 milhões de parâmetros para para 5 milhões, sensivelmente 12 vezes menor que a *AlexNet*. Esta rede com 22 camadas permitiu um menor consumo de memória (Figura 2.7).

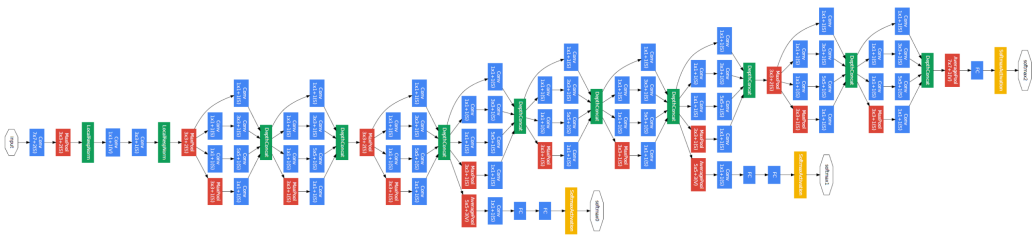


Figura 2.7: Arquitetura GoogleNet. A figura é adotada a partir de [10].

Na Figura 2.7 os blocos azuis são de convolução, os blocos a vermelho

são de *pooling*, os blocos amarelos representam uma função *softmax* e os blocos a verde representam uma função para concatenar e normalizar. Um dos problemas na aprendizagem automática está relacionado com o facto dos gradientes ficarem muito pequenos (*vanishing gradients* [13]), pelo que pode levar a uma situação em que a rede para de aprender. Os engenheiros da GoogleNet abordaram esse problema [14] adicionando camadas intermediárias, de modo que o custo final seja uma combinação do custo intermediário e do custo final.

ResNet

Desde a introdução da *AlexNet*, tomou-se a iniciativa de desenvolver estados de arte das arquiteturas CNN, tal como a *GoogleNet* (também conhecido como *Inception_v1*), que tinha 22 camadas de convolução quando comparado com 5 camadas de convolução da rede *Alexnet*. Aumentar a profundidade da rede não garante a melhoria do desempenho. A *ResNet* faz uso de *Identity shortcut connections*, como se vê na Figura 2.8.

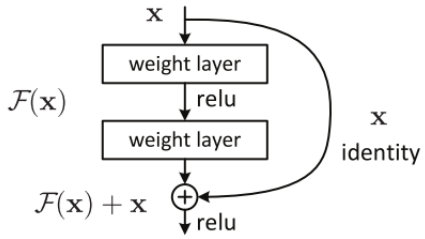


Figura 2.8: Ilustração do salto das conexões. A figura é adotada a partir de [10].

O uso de *Identity shortcut connections* permite saltar conexões para as camadas anteriores, o que ajuda a diminuir os efeitos do gradiente de desvanecer ao longo das camadas sem que prejudique o desempenho da arquitetura. Na Figura 2.9, é mostrado um exemplo de uma arquitetura de uma rede a implementar esse método, onde as setas representam os saltos de conexões.

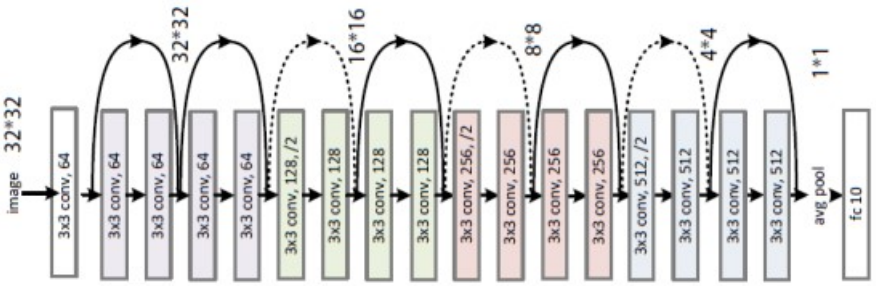


Figura 2.9: Ilustração de saltos de conexões denotados por setas estendidas na rede. A figura é adotada a partir de [10].

VGGNet

A rede *VGGNet* foi desenvolvida pela Universidade de Oxford. Esta CNN destacou-se por ser simples e elegante. Existem duas versões, a *VGG16* e a *VGG19*, sendo que a primeira é composta por 16 camadas (Figura 2.10), sem contar com as camadas de *pooling* e a segunda é composta por 19 camadas. Ambas as redes contêm aproximadamente 140 milhões de parâmetros, o que as torna difíceis de configurar.

Figura 2.10: *Arquitetura VGGNet.*

MobileNet

A *MobileNet*, também desenvolvida pela Google, foi concebida para ser uma rede neuronal pequena, rápida e fácil de integrar em aplicações para telemóveis. Para isso, esta rede (Figura 2.11) usa um conceito novo para reduzir a sua complexidade computacional: *Depthwise Separable Convolutions* [10, 15]. Esta rede neuronal, consegue ter um maior rendimento, devido à relação entre a *accuracy* registada face ao volume da rede neuronal.

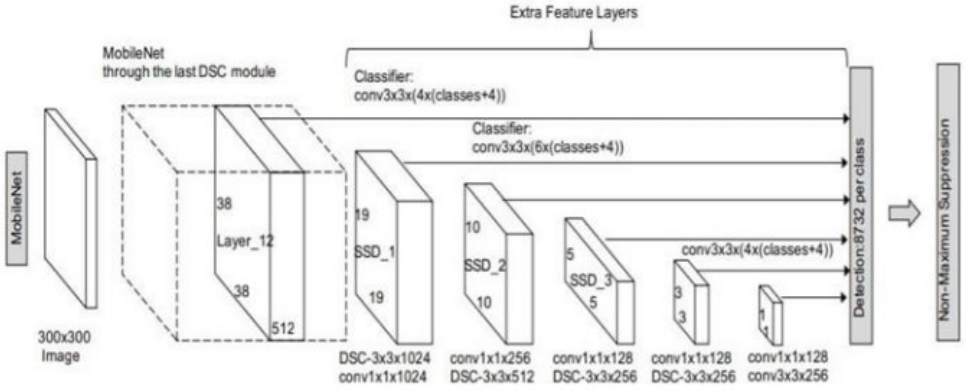


Figura 2.11: Arquitetura MobileNet

2.3.2 Comparação entre as arquiteturas CNN

A publicação [16], apresenta uma análise aprofundada da maioria das redes propostas no estado da arte para o reconhecimento da imagem. Para cada *DNN*, é avaliado a *accuracy*, a complexidade do modelo, a carga computacional, a utilização de memória e o tempo. Os testes são feitos no conjunto de dados o *ImageNet Large-Scale Visual Recognition Competition* (ILSVRC) para 1000 classes (ImageNet-1k) [17]. A *ImageNet* é uma base de dados com mais de 15 milhões de registos trabalhados para serem utilizados no treino de redes neuronais. A *ILSVRC* foi criada em 2010, com o intuito de utilizar como ferramenta que serve para melhorar o estado da arte de deteções de objetos e classificação de imagens em larga escala. Para a validação do treino de cada *DNN*, foi efetuado em média 10 iterações com diferentes tamanhos de *batch*.

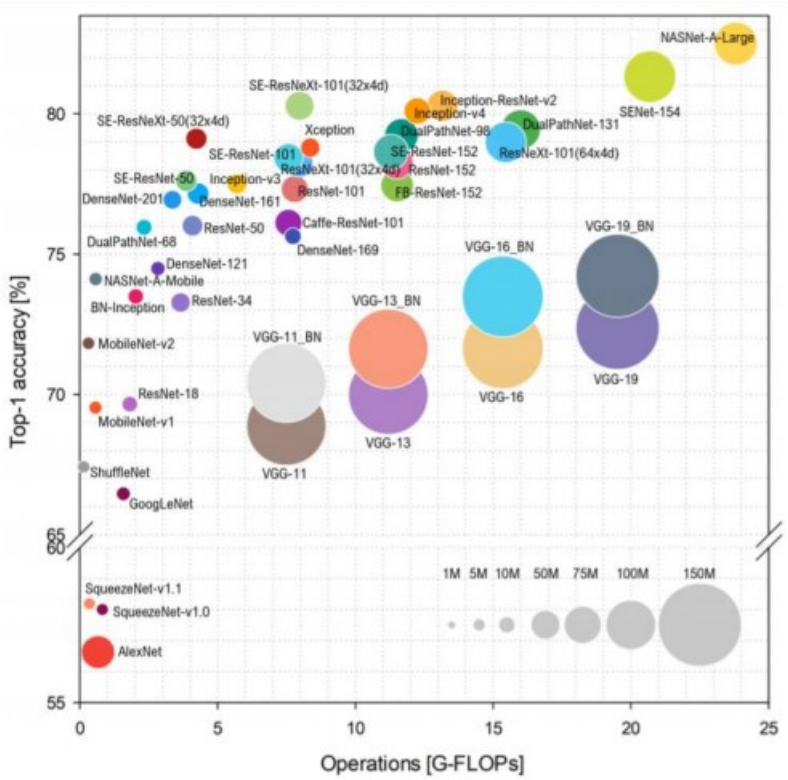
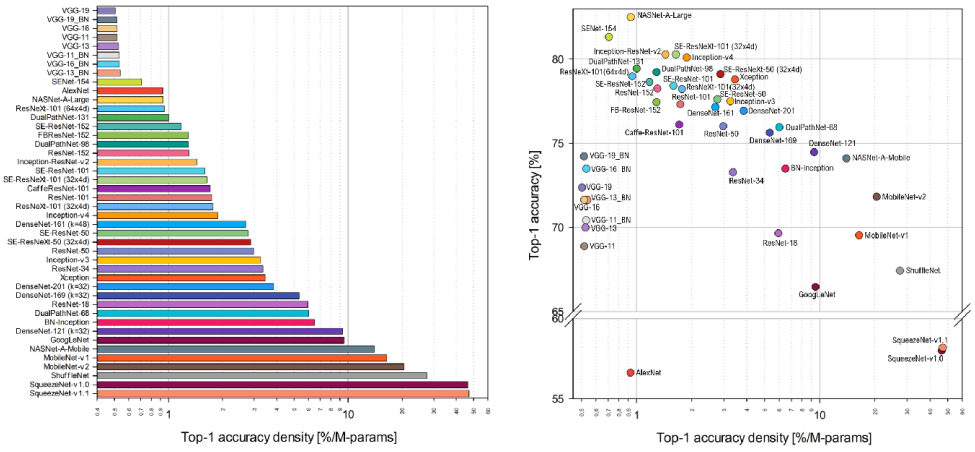


Figura 2.12: Análise da carga dos modelos

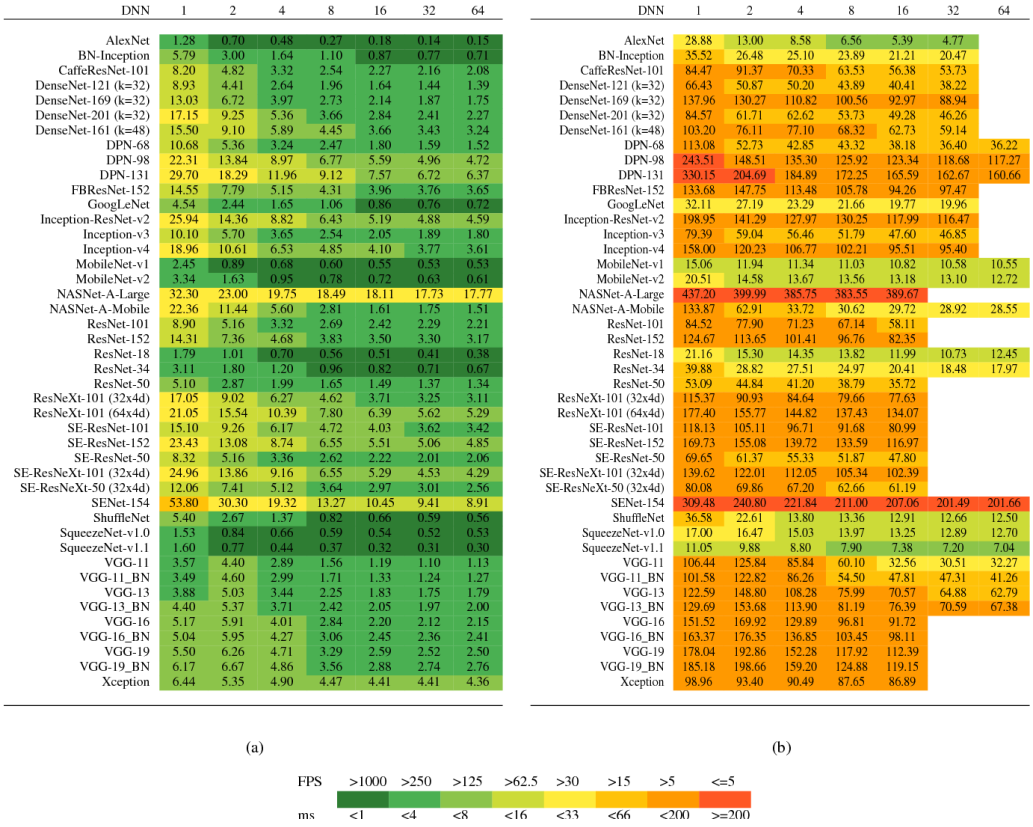
Na Figura 2.12, são apresentados o *accuracy* e a carga computacional para as diversas arquiteturas. Ou seja, demonstra o estudo feito, comparando o resultado da *accuracy* no conjunto de dados de validação, em função dos *FLOPS* (medida de desempenho do computador) em cada iteração da rede, onde o tamanho de cada bola representa a complexidade do modelo.



(a) (b)

Figura 2.13: Número de parâmetros Top-1 *accuracy* (a) Top-1 accuracy vs Densidade de parâmetros Top-1 *accuracy* (b). Análise da eficiência da utilização dos parâmetros usando a *accuracy*

A Figura 2.13, pretende demonstrar a eficiência com que cada modelo utiliza os parâmetros, que pode ser vista pela divisão entre a *accuracy* obtida e o número de parâmetros. Quanto mais alto for este valor, mais alto é a eficiência. O gráfico que mostra isso é apresentado na Figura 2.13(a), onde pode ser visto que os modelos que mais utilizam os seus parâmetros eficazmente são as *SqueezeNets*, *ShuffleNet*, as *MobileNets* e *NASNet-A-Mobile*. Desta forma, é possível ver facilmente que entre os modelos mais eficientes, *NASNet-A-Mobile* e *MobileNet-v2* são as duas que proporcionam uma *accuracy* Top-1 muito superior. Entre os modelos com a maior *accuracy* Top-1 (ou seja, superior a 80%) podemos observar que os modelos que usam os seus parâmetros de forma mais eficiente são *Inception-v4* e *SE-ResNeXt-101*.



DNN	1	2	4	8	16	32	64
AlexNet	1.28	0.70	0.48	0.27	0.18	0.14	0.15
BN-Inception	5.79	3.00	1.64	1.10	0.87	0.77	0.71
CaffeResNet-101	8.20	4.82	3.32	2.54	2.27	2.16	2.08
DenseNet-121 (k=32)	8.93	4.41	2.64	1.96	1.64	1.44	1.39
DenseNet-169 (k=32)	13.03	6.72	3.97	2.73	2.14	1.87	1.75
DenseNet-201 (k=32)	17.15	9.25	5.36	3.66	2.84	2.41	2.27
DenseNet-161 (k=48)	15.50	9.10	5.89	4.45	3.66	3.43	3.24
DPN-68	10.68	5.36	3.24	2.47	1.80	1.59	1.52
DPN-98	22.31	13.84	8.97	6.77	5.59	4.96	4.72
DPN-131	29.70	18.29	11.96	9.12	7.57	6.72	6.37
FBResNet-152	14.55	7.79	5.15	4.31	3.96	3.76	3.65
GoogLeNet	4.54	2.44	1.65	1.06	0.86	0.76	0.72
Inception-ResNet-v2	25.94	14.36	8.82	6.43	5.19	4.88	4.59
Inception-v3	10.10	5.70	3.65	2.54	2.05	1.89	1.80
Inception-v4	18.96	10.61	6.53	4.85	4.10	3.77	3.61
MobileNet-v1	2.45	0.89	0.68	0.60	0.55	0.53	0.53
MobileNet-v2	3.34	1.63	0.95	0.78	0.72	0.63	0.61
NASNet-A-Large	32.30	23.00	19.75	18.49	18.11	17.73	17.77
NASNet-A-Mobile	22.36	11.44	5.60	2.81	1.61	1.75	1.51
ResNet-101	8.90	5.16	3.32	2.69	2.42	2.29	2.21
ResNet-152	14.31	7.36	4.68	3.83	3.50	3.30	3.17
ResNet-18	1.79	1.01	0.70	0.56	0.51	0.41	0.38
ResNet-34	3.11	1.80	1.20	0.96	0.82	0.71	0.67
ResNet-50	5.10	2.87	1.99	1.65	1.49	1.37	1.34
ResNeXt-101 (32x4d)	17.05	9.02	6.27	4.62	3.71	3.25	3.11
ResNeXt-101 (64x4d)	21.05	15.54	10.39	7.80	6.39	5.62	5.29
SE-ResNet-101	15.10	9.26	6.17	4.72	4.03	3.62	3.42
SE-ResNet-152	23.43	13.08	8.74	6.55	5.51	5.06	4.85
SE-ResNet-50	8.32	5.16	3.36	2.62	2.22	2.01	2.06
SE-ResNeXt-101 (32x4d)	24.96	13.86	9.16	6.55	5.29	4.53	4.29
SE-ResNeXt-50 (32x4d)	12.06	7.41	5.12	3.64	2.97	3.01	2.56
SENet-154	53.80	30.30	19.32	13.27	10.45	9.41	8.91
ShuffleNet	5.40	2.67	1.37	0.82	0.66	0.59	0.56
SqueezeNet-v1.0	1.53	0.84	0.66	0.59	0.54	0.52	0.53
SqueezeNet-v1.1	1.60	0.77	0.44	0.37	0.32	0.31	0.30
VGG-11	3.57	4.40	2.89	1.56	1.19	1.10	1.13
VGG-11_BN	3.49	4.60	2.99	1.71	1.33	1.24	1.27
VGG-13	3.88	5.03	3.44	2.25	1.83	1.75	1.79
VGG-13_BN	4.40	5.37	3.71	2.42	2.05	1.97	2.00
VGG-16	5.17	5.91	4.01	2.84	2.20	2.12	2.15
VGG-16_BN	5.04	5.95	4.27	3.06	2.45	2.36	2.41
VGG-19	5.50	6.26	4.71	3.29	2.59	2.52	2.50
VGG-19_BN	6.17	6.67	4.86	3.56	2.88	2.74	2.76
Xception	6.44	5.35	4.90	4.47	4.41	4.41	4.36

(a)

DNN	1	2	4	8	16	32	64
AlexNet	28.88	13.00	8.58	6.56	5.39	4.77	
BN-Inception	35.52	26.48	25.10	23.89	21.21	20.47	
CaffeResNet-101	84.47	91.37	70.33	63.53	56.38	53.73	
DenseNet-121 (k=32)	66.43	50.87	50.20	43.89	40.41	38.22	
DenseNet-169 (k=32)	137.96	130.27	110.82	100.56	92.97	88.94	
DenseNet-201 (k=32)	84.57	61.71	62.62	53.73	49.28	46.26	
DenseNet-161 (k=48)	103.20	76.11	77.10	68.32	62.73	59.14	
DPN-68	113.08	52.73	42.85	43.32	38.18	36.40	36.22
DPN-98	243.51	148.51	135.30	125.92	123.34	118.68	117.27
DPN-131	330.15	204.69	184.89	172.25	165.59	162.67	160.66
FBResNet-152	133.68	147.75	113.48	105.78	94.26	97.47	
GoogLeNet	32.11	27.19	23.29	21.66	19.77	19.96	
Inception-ResNet-v2	198.95	141.29	127.97	130.25	117.99	116.47	
Inception-v3	79.39	59.04	56.46	51.79	47.60	46.85	
Inception-v4	158.00	120.23	106.77	102.21	95.51	95.40	
MobileNet-v1	15.06	11.94	11.34	11.03	10.82	10.58	10.55
MobileNet-v2	20.51	14.58	13.67	13.56	13.18	13.10	12.72
NASNet-A-Large	437.20	399.99	385.75	383.55	389.67		
NASNet-A-Mobile	133.87	62.91	33.72	30.62	29.72	28.92	28.55
ResNet-101	84.52	77.90	71.23	67.14	58.11		
ResNet-152	124.67	113.65	101.41	96.76	82.35		
ResNet-18	21.16	15.30	14.35	13.82	11.99	10.73	12.45
ResNet-34	39.88	28.82	27.51	24.97	20.41	18.48	17.97
ResNet-50	53.09	44.84	41.20	38.79	35.72		
ResNeXt-101 (32x4d)	115.37	90.93	84.64	79.66	77.63		
ResNeXt-101 (64x4d)	177.40	155.77	144.82	137.43	134.07		
SE-ResNet-101	118.13	105.11	96.71	91.68	80.99		
SE-ResNet-152	169.73	155.08	139.72	133.59	116.97		
SE-ResNet-50	69.65	61.37	55.33	51.87	47.80		
SE-ResNeXt-101 (32x4d)	139.62	122.01	112.05	105.34	102.39		
SE-ResNeXt-50 (32x4d)	80.08	69.86	67.20	62.66	61.19		
SENet-154	309.48	240.80	221.84	211.00	207.06	201.49	201.66
ShuffleNet	36.58	22.61	13.80	13.36	12.91	12.66	12.50
SqueezeNet-v1.0	17.00	16.47	15.03	13.97	13.25	12.89	12.70
SqueezeNet-v1.1	11.05	9.88	8.80	7.90	7.38	7.20	7.04
VGG-11	106.44	125.84	85.84	60.10	32.56	30.51	32.27
VGG-11_BN	101.58	122.82	86.26	54.50	47.81	47.31	41.26
VGG-13	122.59	148.80	108.28	75.99	70.57	64.88	62.79
VGG-13_BN	129.69	153.68	113.90	81.19	76.39	70.59	67.38
VGG-16	151.52	169.92	129.89	96.81	91.72		
VGG-16_BN	163.37	176.35	136.85	103.45	98.11		
VGG-19	178.04	192.86	152.28	117.92	112.39		
VGG-19_BN	185.18	198.66	159.20	124.88	119.15		
Xception	98.96	93.40	90.49	87.65	86.89		

(b)

Figura 2.14: Análise da inferência de tempo

O tempo médio inferido por imagem para uma média de 10 iterações para todos os modelos *DNN* considerados, podem ser visualizados na Figura 2.14 para tamanho de *batch* igual a 1, 2, 4, 8, 16, 32 e 64, usando dois GPU, o *Titan Xp* e o *Jetson*. A partir da Figura 2.14 é possível ver que todos os modelos *DNN* considerados, são capazes de atingir altos desempenhos em tempo real com o *Titan Xp*, com a única exceção do *SENet-154*, quando é considerado um tamanho de *batch* de 1. No *Jetson* em vez disso, apenas alguns modelos são capazes de atingir bons desempenhos em tempo real quando se considera um *batch* de tamanho 1, nomeadamente as *SqueezeNets*, as *MobileNets*, *ResNet-18*, *GoogLeNet*, e *AlexNet*. As medições em falta devem-se à falta de memória de sistema suficiente para processar o maior *batch*. Mas, podemos dizer que os melhores modelos que atingem melhores desempenhos são, principalmente, a *AlexNet*, *MobileNets*, *SqueezeNets* e a *ResNet*.

O *GitHub* é uma ferramenta essencial de sistema de controlo de projetos e versões de código de desenvolvimento para desenvolvedores de software, utilizada mundialmente. Atualmente, o *GitHub* dispõe mais de 83 milhões

de utilizadores. Isso significa que há um número considerável de profissionais que usam o *GitHub* para melhorar o fluxo de trabalho e a sua colaboração na plataforma. Sendo, uma plataforma usada por muitos utilizadores, foi elaborado uma pesquisa na plataforma sobre o número de correspondências para cada uma das redes com os melhores desempenhos: *AlexNet*, *MobileNet*, *SequeezeNet* e *ResNet*. De acordo com os resultados encontrados através das correspondências, a palavra "*AlexNet*" correspondeu a cerca de 2,067 repositórios criados, a "*MobileNet*" correspondeu a 5,751 repositórios e a "*ResNet*"a 7,599. Por outro lado, a "*SequeezeNet*" correspondeu só a 337 repositórios. Sendo esta uma plataforma muito utilizada por profissionais de software, e aliando a informação recolhida no estudo anterior, fica demonstrado um indício forte que a arquitetura da *SequeezeNet* não é uma arquitetura muito utilizada, enquanto que a *AlexNet*, *MobileNet* e a *ResNet* oferecem mais garantias.

2.4 Transfer Learning

O *transfer learning* [18, 19] é uma metodologia que permite reaproveitar modelos previamente treinados. Uma máquina utiliza o conhecimento adquirido de uma tarefa anterior para aumentar a previsão sobre uma nova tarefa. Por exemplo, utilizar a informação obtida durante a aprendizagem para distinguir gatos no treino de um classificador para cães (Figura 2.15).

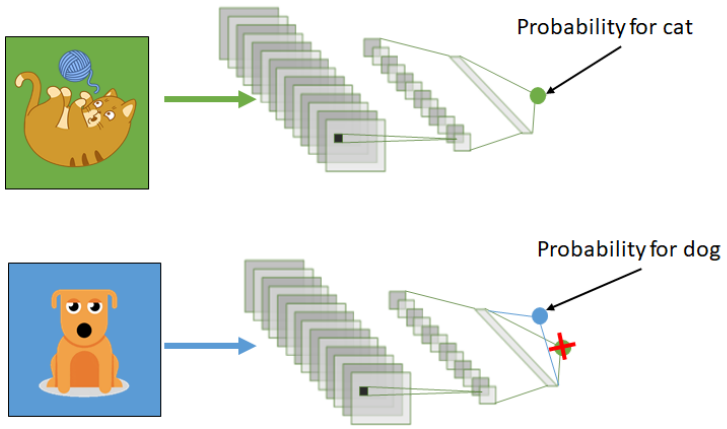


Figura 2.15: Transfer Learning. Figura adotada de [18]

Com o *Transfer Learning*, utiliza-se o que aprendemos numa tarefa para

compreender melhor os conceitos em outra tarefa. Ou seja, os pesos do modelo, estão a ser automaticamente transferidos (daí a palavra *Transfer Learning*) para uma rede que realiza a tarefa A de uma rede que executa a tarefa B.

Transfer Learning é aplicado em diversas áreas, como por exemplo em tarefas como processamento de visão. No processamento de visão, as redes visam tipicamente detetar arestas na primeira camada, formas na camada intermédia e caraterísticas especificas nas últimas camadas. As camadas iniciais e centrais são utilizadas no *transfer learning*, e as últimas camadas (as *fully-connected*) são re-treinadas.

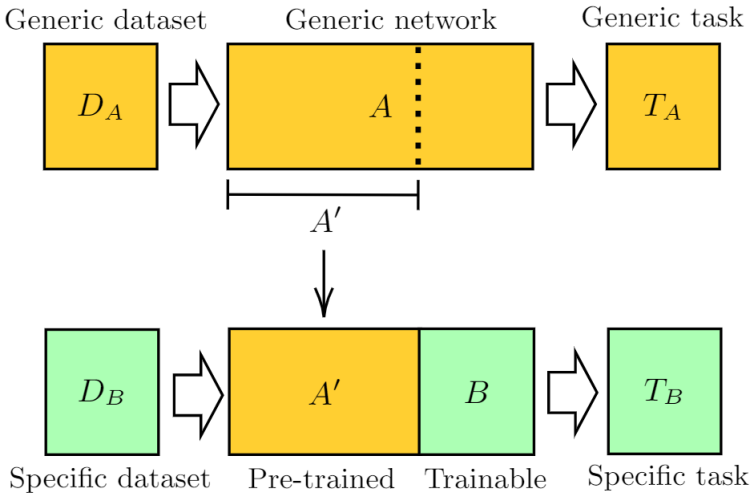


Figura 2.16: Transfer Learning para a execução de uma nova tarefa. Figura adotada de [19].

Na Figura 2.16, podemos ver que a rede A que executa a tarefa A, foi lhe retirada as camadas finais e adicionada novas camada, sendo essas as camadas treináveis. Assim, mantemos os pesos da rede e treinamos a nova rede com as novas camadas finais para executar uma nova tarefa.

As vantagens mais importantes são a redução do tempo de treino do modelo, a melhoria do desempenho e a ausência da necessidade de uma grande quantidade de dados. Porque para treinar um modelo a partir do zero, são normalmente necessários muitos dados.

2.5 Estudo de Frameworks

Nesta secção serão apresentadas as tecnologias envolvidas no domínio das CNN.

2.5.1 Frameworks e bibliotecas

Um *framework* é uma definição que vai além do mercado de *software*. Em outros contextos, refere-se a uma série de ações e estratégias que visam solucionar um problema bem específico. Assim, quando se deparam com esse cenário, os profissionais recorrem a um conjunto de abordagens e otimizam os seus resultados.

Na área de tecnologia, a definição é semelhante, trata-se de uma série de bibliotecas para conseguir executar uma funcionalidade específica, é um padrão que pode ser incorporado a sistemas para agilizar a codificação de certas partes. Um *framework* fornece uma estrutura para o sistema de forma a facilitar o desenvolvimento e implementação de decisões específicas. Já uma biblioteca é uma ferramenta que o próprio sistema decide o que utilizar.

Na área de *Deep Learning*, já existem alguns *frameworks*, sendo os mais populares: *TensorFlow*, *Keras*, *PyTorch*, *Caffe* and *CNTK*. Estes *frameworks* já dispõem de vários modelos com pesos pré-treinados. Existem aspetos básicos para a utilização de um *framework*, nomeadamente a facilidade de instalação, configuração, implementação e usabilidade, no entanto, existem ainda quatro [20] requisitos fundamentais para as ferramentas de Deep Learning:

- possibilidade de trabalhar com tensores, como foi visto na secção 2.3;
- ferramentas de auto-diferenciação;
- existência de extensões que permitam trabalhar com BLAS/cuBLAS2 (implementação subrotinas de álgebra linear elementar que é executada nos processadores gráficos - GPUs) e *cuDNN3* (bibliotecas que permitem a implementação de redes neuronais profundas que são executadas nos processadores gráficos)

Tensorflow

TensorFlow é uma plataforma *open-source* de aprendizagem automática com foco particular nas redes neuronais, desenvolvida pela equipa o *Google Brain team*. Empresas como a *Airbnb*, *Twitter*, *Snapchat*, *NVIDIA*, *Dropbox* e a própria Google, utilizam o *framework*.

O *TensorFlow* é multi-plataforma, podendo ser executado no *Windows*, *MacOS* ou *Linux*. Além disso, também pode ser executado em *CPUs*, *GPUs* (placas de vídeo, que aceleram o processamento), ou ainda TPU (*Tensor Processing Unit*, que são circuitos integrados específicos desenvolvidos pelo *Google* para acelerar a aprendizagem automática).

Existem *APIs* do *TensorFlow* em várias linguagens: *Python*, *JavaScript*, *C++*, *Java*, *Go*, *Swift*, *C*, *Haskell*, *Julia*, *Ruby*, *Rust* e *Scala*.

PyTorch

O *PyTorch* é um *framework open-source* computacional, que suporta algoritmos de *machine learning*, disponibilizada pelo *Facebook*. A *Pytorch* é capaz de realizar cálculos utilizando tensores. Tendo como vantagem o uso do *CPU* ou *GPU*, como o *TensorFlow*.

Caffe

Caffe é um *framework Deep Learning*, desenvolvido pela *Berkeley AI Research* (BAIR) e por colaboradores da comunidade. *Yangqing Jia* criou o projeto durante o seu doutoramento na *UC Berkeley*.

Os modelos e a otimização são definidos por configuração sem *hard-coding*. É possível alternar entre *CPU* e *GPU*, definindo apenas com uma única *flag* para treinar uma máquina *GPU* e, em seguida, instalar em dispositivos móveis.

Tem vindo a ser desenvolvido por mais de 1000 criadores e teve muitas mudanças significativas. A velocidade torna o *Caffe* perfeito para experiências de investigação e implantação na indústria. O *Caffe* pode processar mais de 60M de imagens por dia com uma única *GPU NVIDIA k40*.

O *Caffe* já impulsiona projetos de investigação académica, protótipos de inicialização, e mesmo aplicações industriais em larga escala em visão, fala, e multimédia.

CNTK

O *Microsoft Cognitive Toolkit* (CNTK) é um conjunto de ferramentas de código aberto para Deep Learning distribuída a nível comercial. Descreve as redes neuronais como uma série de passos computacionais através de um gráfico. O *CNTK* permite ao utilizador realizar e combinar facilmente tipos de modelos populares, tais como *feed-forward DNNs*, redes neuronais de convolução (*CNNs*) e redes neuronais recorrentes (*RNNs/LSTMs*). A *CNTK* implementa a descida de gradiente estocástico (*SGD*, *backpropagation* de erro) com diferenciação automática e paralelização através de múltiplas

GPUs e servidores.

A *CNTK* pode ser incluída como biblioteca em programas *Python*, *C*, ou *C++*, ou utilizada como ferramenta de *machine learning* através da própria linguagem de descrição de modelos (*BrainScript*). Além disso, pode-se usar a funcionalidade de avaliação de modelos *CNTK* a partir de programas *Java*.

O *CNTK* suporta sistemas operativos Linux de 64 bits ou sistemas operativos *Windows* de 64 bits.

Keras

Keras é uma *API* de *Deep Learning* escrita em *Python*, que é executada usando diversos *backends*, nomeadamente *TensorFlow*, *Theano* e *CNTK*.

Keras é simples, flexível e é usado por organizações como, a *NASA*, *Youtube* e *Waymo*. Pode ser executada, quer em *CPU*, quer em *GPU*. É um *framework* desenvolvida e mantida por *Francois Chollet*, no entanto, faz parte das equipas do *TensorFlow*, o que faz com que desse modo, o *backend* preferencial seja o *TensorFlow*.

2.5.2 Comparação de Frameworks

A Tabela 2.1, mostra um resumo e a comparação dos *frameworks* que apresentámos anteriormente.

Software	Torch	Theano	Caffe	Keras	TensorFlow	CNTK	PyTorch
Initial release	2002	2007	2013	2015	2015	2016	2016
Actively developed	No	No	No	Yes	Yes	No	Yes
Open source	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Written in	C, Lua	Python	C++	Python	C++, Python, CUDA	C++	Python, C, C++, CUDA
CUDA support	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Tabela 2.1: Resumo e comparação das *frameworks* que foram mencionadas

Para além dos *frameworks* que foram apresentados acima, é possível visualizar outras onde foi retirado esta tabela [21]. Onde a maioria dos *frameworks* já não são atualmente mantidos pelos criadores. Por essa razão, é que foi escolhido os *frameworks* que se apresentam ativos e que são mais conhecidas.

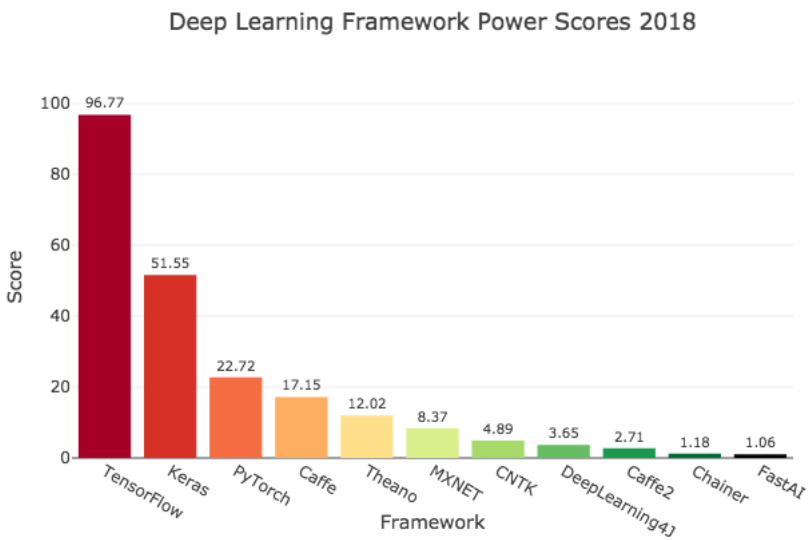


Figura 2.17: Resultados de frameworks. Imagem adotada de [22]

Os dados da Figura 2.17 [22, 23] mostram os resultados de registos encontrados de cada *framework*. Apesar de estes dados não estarem atualizados, são muito próximos dos resultados atuais. Muitos dos profissionais preferem o *TensorFlow* usando como complementar o *Keras*. Para esse confirmação, foi usado o *GitHub* para tal análise, onde foi pesquisado o número de registos. Para o "*TensorFlow*"temos mais de 24M de registos, para o "*Keras*"mais de 5M, para o "*pytorch*"mais de 4M e por fim, temos o "*Caffe*"com mais ou menos 2M. Foi ainda feito uma segunda análise, usando o *Google Trends*.

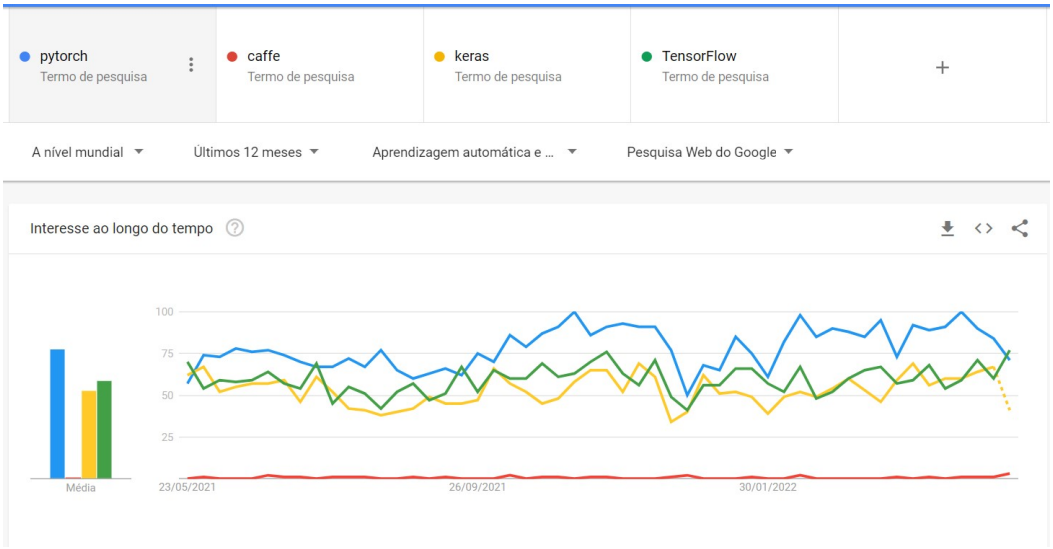


Figura 2.18: Resultados de frameworks com a ferramenta Google Trends

Podemos ver que, os *frameworks* que predominam a área dos *frameworks Deep Learning* são: *TensorFlow*, *Keras* e *PyTorch*. Como o *Keras* pode ser usada como complemento do *TensorFlow*, foi decidido utilizar estas duas *frameworks*. Outro motivo que reforça a escolha do *framework TensorFlow*, é o maior número de registos do *TensorFlow* no *GitHub* em relação ao *PyTorch*, indicando que mais profissionais nesta área estão a utilizá-la.

2.6 Trabalhos Relacionados

Nesta secção, descreve-se um conjunto de projetos que estão diretamente ou indiretamente ligados à identificação e classificação de plantas e doenças em plantas, e que serviram de referência no desenvolvimento deste projeto.

2.6.1 Aprendizagem Profunda - Estudo e Aplicações

O trabalho publicado em [20] não está diretamente relacionado com o trabalho mas aborda o tema da Aprendizagem Profunda, estudando-o através da comparação de vários *frameworks* e utilizando um conjunto de dados composto por imagens de algarismos manuscritos. Os *frameworks* utilizadas para o estudo foram algumas das mais conhecidas, como o *Caffe*, *Theano* e *TensorFlow*, entre outras. Foi realizado também um estudo mais aprofundado da *Theano* com o *Keras*, *TensorFlow* com o *Keras* e *Microsoft CNTK*. Foi analisado o desempenho de algoritmos pertencentes a três paradigmas

da aprendizagem automática (supervisionada, semi-supervisionada e não supervisionada) através do conjunto de imagens de algarismos manuscritos. À data em que foi realizado este trabalho de investigação, constata-se que os métodos de aprendizagem profunda são significativamente melhores do que os métodos clássicos de aprendizagem automática, quando os dados são supervisionados ou semi-supervisionados. No que diz respeito ao trabalho com dados não supervisionados, conclui-se que o desenvolvimento ainda está numa fase embrionária. Com este tipo de dados, criam-se modelos que representem uma aproximação da realidade.

2.6.2 PlantAI

Daniel Bento [24] desenvolveu uma aplicação para a deteção e identificação de doenças em plantas utilizando Deep Learning, onde fez estudos das várias arquiteturas CNN e tecnologias. A aplicação foi desenvolvida com a ajuda da aplicação *Xamarin* [1], sendo o primeiro objetivo, implementar uma aplicação compatível com *Android*. Foram ainda criadas duas APIs, a *.Net Core API* e *Flask API*. A primeira *API* foi desenvolvida em *.NetCore* 2.1 e tem como principal função servir qualquer tipo de aplicação externa, ou seja, fazer a ligação entre os dados, aplicação *web*, aplicação móvel e ainda serve como intermediário de toda a parte *AI* do sistema. A segunda *API*, foi desenvolvida em *Python* e tem como principal objetivo servir a parte *AI* do sistema utilizando as seguintes funcionalidades: treino do modelo, a alteração do modelo em vigor e a classificação de imagem (Figura 2.19).

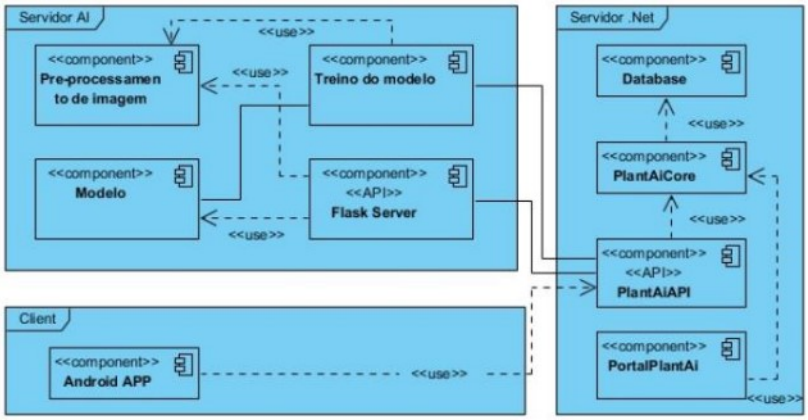


Figura 2.19: Arquitetura que Daniel Bento adotou para a implementação da aplicação.

[1] https://dotnet.microsoft.com/en-us/learn/xamarin/what-is-xamarin

A fonte de dados que serviu de suporte a esta dissertação, foi obtida através de uma organização, chamada *PlantVillage* [25]. Essa organização recolheu, agrupou e disponibilizou em várias plataformas esses dados para a consulta e utilização que os utilizadores achem conveniente. O conjunto de dados contém cerca de cinquenta e quatro mil imagens, distribuídas por trinta e oito classes, onde cada classe representa um problema fitossanitário (doença) distinto, pertencentes a catorze espécies de plantas agrícolas diferentes.

Para além de ter implementado uma aplicação móvel para a deteção e identificação das doenças foi ainda implementado uma aplicação web (Figura 2.20) usando a *API .NetCore*.

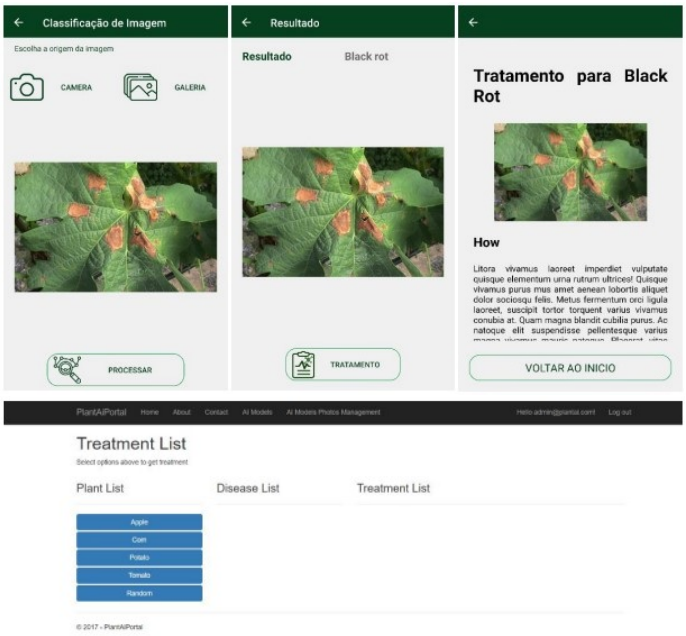


Figura 2.20: Aplicação mobile e web desenvolvido por Daniel Bento.

2.6.3 PlantDoc

No artigo [26], foi abordado o problema da deteção de folhas doentes/-saudáveis em imagens usando o estado de arte dos modelos de deteção de objetos. Uma das principais contribuições deste trabalho foi propor um conjunto de dados inteiramente novo para a deteção de doenças das plantas, chamado *PlantDoc*.

Figura 2.21: Amostras de várias classes do Dataset PlantDoc mostram a lacuna entre as imagens controladas pelo laboratório (PlantVillage [25]) e as da vida real.

As experiências feitas (Tabelas 2.2 e 2.3), onde fazem comparação a vários modelos, mostram a sua falta de eficácia em imagens do mundo real, quando foram treinados com conjunto de dados controlados, como o *PlantVillage* (*PVD*), mostrando assim a importância de conjuntos de dados do mundo real, como o *Cropped-PlantDoc* (*C-PD*).

Model	PreTrained Weights	Training Set (Set %)	Test Set (Set %)	Accuracy	F1-Score
VGG16	ImageNet	C-PD (80)	C-PD(20)	44.52	0.44
VGG16	ImageNet	PVD	C-PD (100)	19.73	0.18
VGG16	ImageNet+PVD	C-PD (80)	C-PD (20)	60.41	0.60
InceptionV3	ImageNet	C-PD (80)	C-PD (20)	46.67	0.46
InceptionV3	ImageNet	PVD	C-PD (100)	30.78	0.28
InceptionV3	ImageNet+PVD	C-PD (80)	C-PD (20)	62.06	0.61
InceptionResNet V2	ImageNet	C-PD (80)	C-PD (20)	49.04	0.49
InceptionResNet V2	ImageNet	PVD	C-PD (100)	39.87	0.38
InceptionResNet V2	ImageNet+PVD	C-PD (80)	C-PD (20)	**70.53**	**0.70**

Tabela 2.2: Treino de conjunto de dados controlados (PlantVillage - PVD) apresenta um fraco desempenho em imagens do mundo real. Desempenho sobre imagens do mundo real podem ser melhoradas através do treino em imagens reais

Model	PreTrained Weights	mAP (at 50% iou)
MobileNet	COCO	32.8
MobileNet	COCO+PVD	22.4
Faster-rcnn-inception-resnet	iNaturalist	36.1
Faster-rcnn-inception-resnet	COCO	**38.9**

Tabela 2.3: Deteção de folhas

Foi ainda usada a técnica da segmentação de imagem para extrair folhas das imagens que podem potencialmente melhorar a utilidade do conjunto de dados (Figura 2.22).

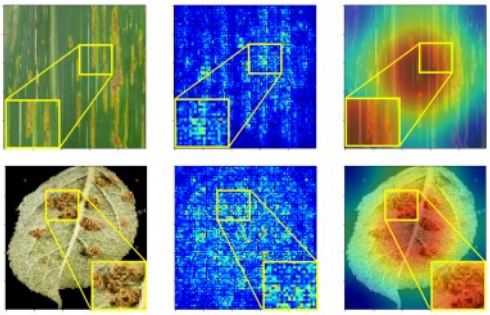


Figura 2.22: Saliência e Mapas de ativação mostra partes de uma doença numa folha.

Em baixo, na Figura 2.23, podemos ver o resultado final de uma aplicação móvel, que tem como funcionalidade de detetar com retângulos as folhas e identificar a sua doença.

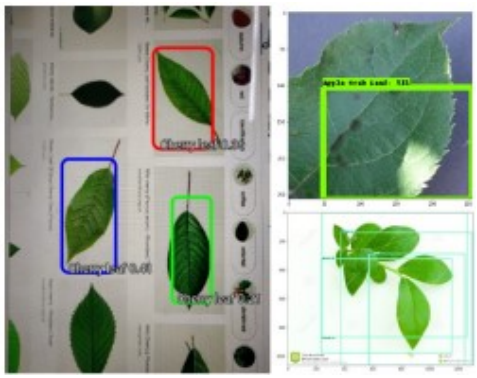


Figura 2.23: Resultados da deteção de folha na aplicação móvel

2.6.4 DiaMOS Plant

No artigo [27], é divulgado publicamente um conjunto de dados de campo recolhidos para diagnosticar e monitorizar os sintomas das plantas, chamado *DiaMOS Plant*, que consiste em 3505 imagens de frutos e folhas de pêra afetados por quatro doenças. Foi realizado uma análise comparativa aos conjuntos de dados da literatura existente, concebidos para a classificação e reconhecimento de doenças das folhas. Este estudo fornece orientações que serão úteis para este projeto de investigação na seleção e construção de dados. Foram usados métodos de classificação e reconhecimento baseados em modelos *CNN* e a análise que foi realizada neste artigo realça as boas práticas para a construção de conjuntos de dados, que têm impacto no conteúdo de

informação que os dados podem expressar, como a capacidade de descrever o ambiente ao redor no qual foram extraídos ou observados. Vários fatores foram tidos em consideração na construção do conjunto de dados proposto.

2.6.5 Ashley Poon - Detetar a saúde das plantas

Ashley Poon [28] desenvolveu um estudo para a construção de um conjunto de dados para a deteção e classificação de doenças em plantas usando modelos *CNNs*. Tem como objetivo identificar onde está a planta na imagem e aferir se esta se encontra saudável. O conjunto de dados foi construido com uma técnica chamada *Web Scraping* 4.3, que é uma técnica para recolher dados a partir de sites de forma automatizada. Para além de usar modelos *CNN* usou também o *Yolo* [29], que é um método de deteção de objetos de passagem única que utiliza uma rede neuronal convolucional como extrator de características.

2.6.6 iBlurDetect

Em [30], são discutidos vários métodos para a deteção de imagens com desfoque. Este estudo tem por objetivo fornecer uma comparação no desempenho dos estados de arte dos métodos disponibilizados.

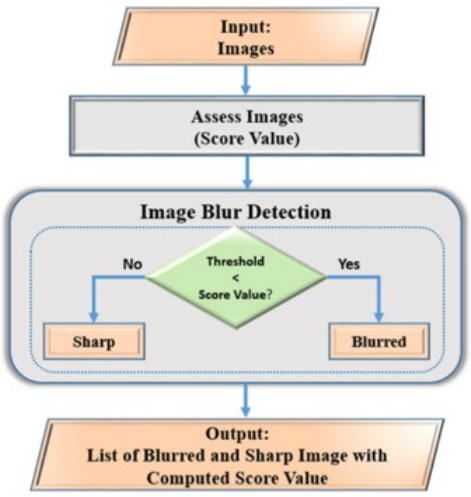


Figura 2.24: Fluxograma do processo de deteção de desfoque de uma imagem

Na Figura 2.24, podemos ver que para cada imagem é calculado um valor com o qual se determina se a imagem se encontra ou não desfocada.

2.6.7 Outras Contribuições

Nesta secção são apresentadas outras contribuições que serviram de referência, não no seu todo mas em particularidades como por exemplo, as propostas para resolver o problema das imagens não focadas.

No artigo [31] é apresentado uma solução para identificação de plantas utilizando *CNNs*. Neste trabalho não há o objetivo de detetar doenças nas plantas. A maior parte das propostas que detetam doenças são ligadas à agricultura, como por exemplo, o trabalho publicado em [32]. Estes trabalhos baseiam-se na deteção de plantas ou doenças no exterior. Na deteção de plantas em ambientes caseiros (interiores) a questão da iluminação pode ser um problema. Em [33, 34] são abordadas técnicas de deteção utilizando imagens com pouco luminosidade. Outro problema pode ser a falta de contraste, que pode ser consequência da pouca luminosidade. No trabalho proposto em [35] é apresentada uma técnica baseada na equalização de histograma adaptativo para identificar faces em imagens com pouco contraste.

São também apresentadas outras propostas para a deteção em imagens desfocadas [36, 37, 38, 39, 40, 41] que apresentam as mais diversas soluções para resolver este problema.

2.7 Aplicações Móveis e para a Web

Existem enumeras aplicações relativamente a esta área. Algumas dessas aplicações já têm anos de estudo, onde são usadas várias técnicas para a classificação e identificação de plantas ou doenças. Nesta secção, são apresentadas algumas dessas aplicações já existentes no mercado.

2.7.1 Pl@ntNet

Pl@ntNet [42] é uma aplicação (web e móvel) para recolher, anotar e pesquisar imagens para ajudar a identificar plantas, desenvolvida por um grupo que envolve cientistas da *CIRAD* [2], *INRAE* [3], *INRIA* [4], *IRD* [5] e da rede de Tela Botânica [6], num projeto financiado pela *Agropolis Fondation* [7]. Integra um sistema de ajuda para a identificação automática de plantas a partir de fotografias, em comparação com imagens de uma base de dados

[2]https://www.cirad.fr/
[3]https://www.inrae.fr/
[4]https://www.inria.fr/
[5]https://www.ird.fr/
[6]https://www.tela-botanica.org/
[7]https://www.agropolis-fondation.fr/

botânica. Os resultados tornam possível encontrar o nome botânico de uma planta se este estiver suficientemente ilustrado na base de dados de referência. Tanto o número de espécies processadas como o número de imagens utilizadas evoluíram com contribuições externas a este projeto.

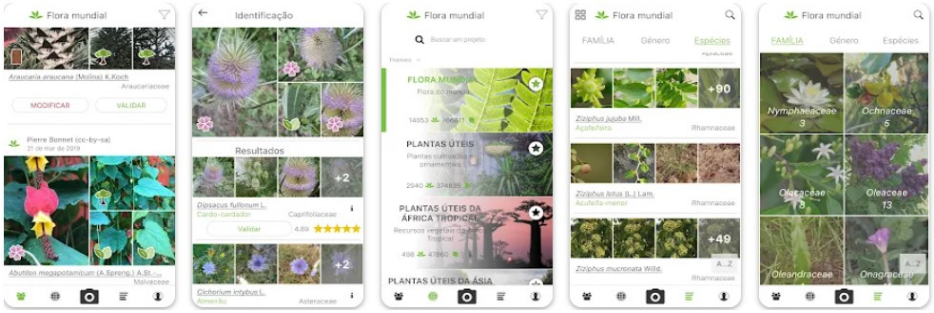


Figura 2.25: Aplicação mobile da Pl@ntNet

A Figura 2.25 ilustra a aplicação móvel. Em ambas as aplicações (web e móvel) não é possível a identificação de plantas ornamentais (plantas cultivadas pela sua beleza e usadas na arquitetura de interiores e no paisagismo de espaços externos) ou hortícolas (plantas de fácil cultivo que podem ser cultivadas em hortas e podem servir como alimento das pessoas). As aplicações funcionam melhor se as imagens submetidas incidirem sobre uma parte da planta bem definida. As imagens de folhas de árvores sobre um fundo uniforme proporcionam resultados mais relevantes.

2.7.2 Leafweb

Leafweb [43] é um projeto de classificação de doenças e deteção de saliência para o diagnóstico de plantas com base numa imagem. E que disponibiliza um protótipo em formato de um *website* [44] onde é possível submeter uma imagem e obter uma classificação para essa imagem.

Este projeto é desenvolvido com a ajuda da *framework* Pytorch, onde utilizam o conjunto de dados disponibilizados pela *PlantVillage* e onde aplicam a técnica de *Transfer Learning* usando a rede *ResNet101*.

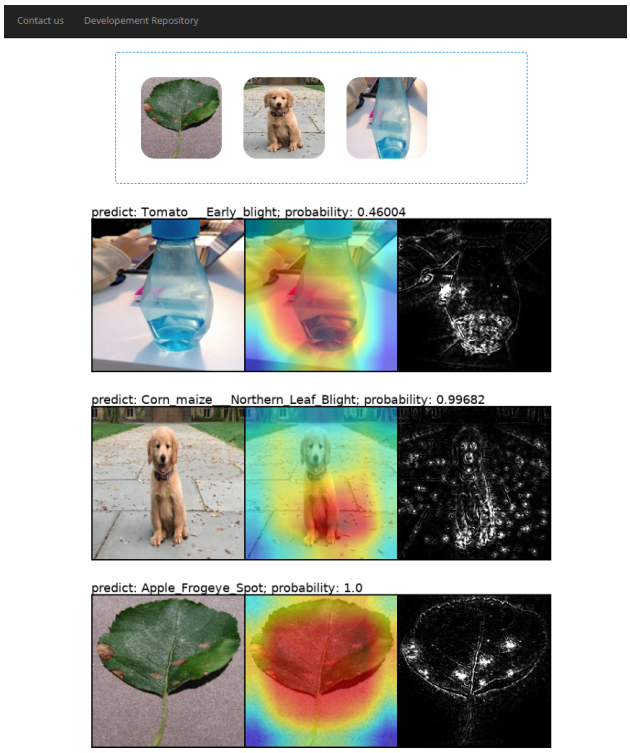


Figura 2.26: Aplicação web do projeto Leafweb

Como podemos ver na Figura 2.26, este protótipo apenas disponibiliza previsões para classes contidas no conjunto de dados *PlantVillage*, assim sendo, ao classificar uma imagem externa ao sistema, o resultado obtido irá ser também uma das categorias existentes no conjunto de dados.

2.7.3 Picture This

O *Picture This* é uma aplicação móvel que permite submeter uma fotografia para identificar plantas e diagnosticar doenças. Permite também falar com peritos na área de plantas, criar coleções e adicionar lembretes às suas plantas adicionadas. Para além de poder classificar plantas, ainda é possível classificar insetos, pássaros, ervas daninhas, anos das árvores pelos anéis de árvores e muito mais (Figura 2.27).

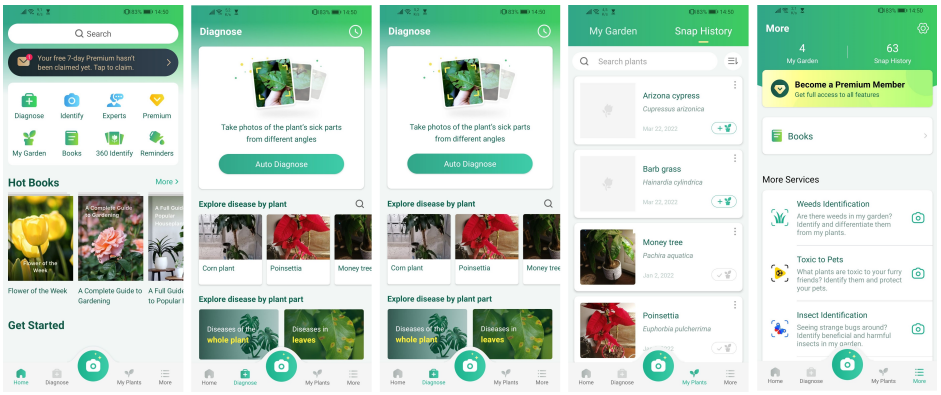


Figura 2.27: Aplicação mobile do Picture This

Esta aplicação contém inúmeros recursos, sendo a aplicação mais robusta das apresentadas. A aplicação tem um tempo limite até poder deixar de ser utilizado gratuitamente, depois disso, será necessário pagar para usufruir do resto da aplicação.

2.8 Sumário

Neste capítulo foram apresentados vários trabalhos relacionados com o trabalho proposto neste documento. Em primeiro lugar, foram apresentados várias propostas mais dedicadas aos algoritmos de deteção de plantas e de doenças em plantas. Nenhum dos trabalhos aborda as plantas domésticas. Os algoritmos são baseados em técnicas de *deep learning* e utilizam arquiteturas semelhantes à utilizada neste projeto. Depois são apresentadas várias aplicações que tiram partido dos algoritmos. A maioria apresenta uma interface utilizador que a partir de uma imagem de uma planta mostram informação relacionada com a planta.

Muitos dos problemas identificados no trabalho relacionado são devido à falta de qualidade das imagens capturadas ou com as condições dos locais onde foram obtidas. Neste trabalho é aplicado um bloco de pré-processamento para minorar estes problemas (ver Secção 4).

Capítulo 3

Análise do sistema

Este capitulo faz uma abordagem da visão do sistema, do ponto de vista estrutural, analisando as funcionalidades pretendidas da arquitetura do sistema que inclui um sistema de deteção e classificação de doenças em plantas.

3.1 Funcionalidades

No sistema existem 3 tipos de utilizadores, o utilizador comum, o utilizador registado e o gestor de conteúdos.

Utilizador comum

O utilizador comum não possui nenhuma conta e pode visualizar as plantas e doenças existentes, como também pode usar o classificador para identificar o tipo de planta e diagnosticar a doença.

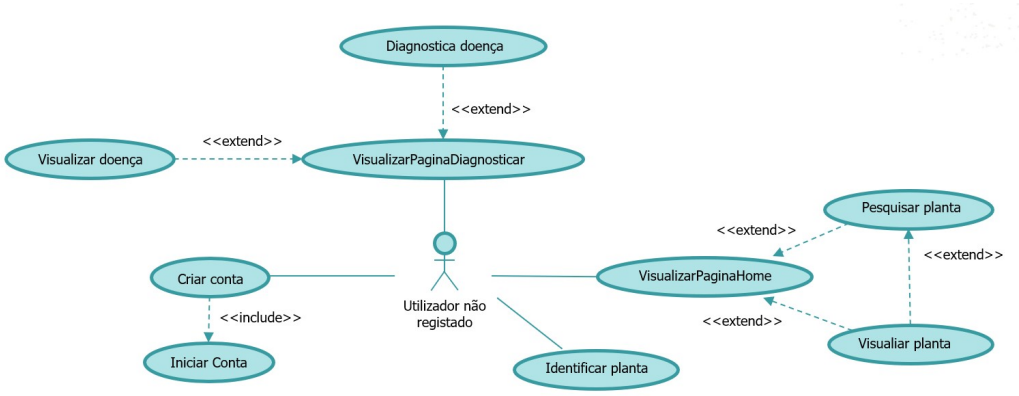


Figura 3.1: Funcionalidades do utilizador comum

Como podemos ver na Figura 3.1, o utilizador comum deverá conseguir visualizar as informações sobre plantas e doenças, como também poderá usar o sistema de classificação para identificar e diagnosticar. Sendo um utilizador comum, terá também a funcionalidade de criar conta.

Utilizador registado

Um utilizador registado, para além de incluir as mesmas funcionalidades do utilizador anterior, consegue ainda adicionar plantas ao seu perfil, visualizar o histórico referentes às plantas e doenças identificadas.

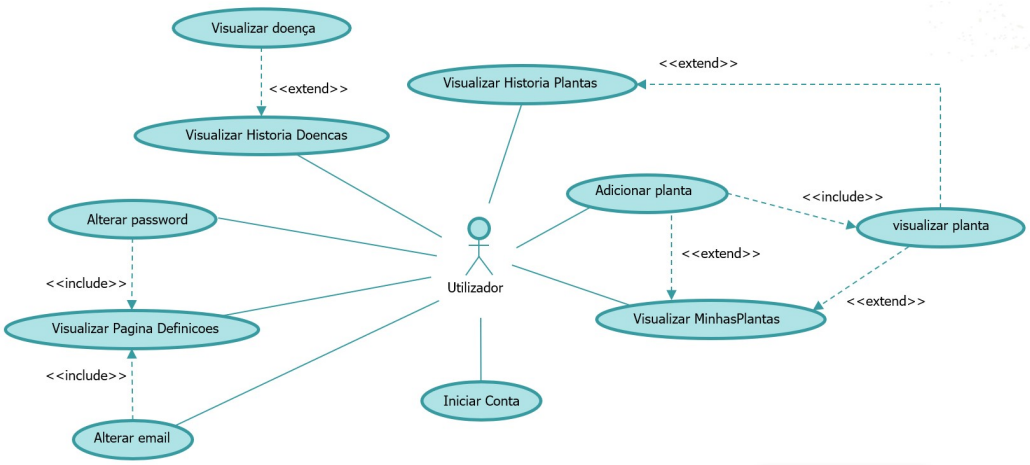


Figura 3.2: Funcionalidades do utilizador registado

Como podemos ver na Figura 3.2, sendo um utilizador registado, é possível alterar as credenciais da sua conta, adicionar plantas ao seu perfil e visualizar o histórico.

Gestor de conteúdos

Por fim, temos o gestor de conteúdos (Figura 3.3). Este tem como funcionalidade adicionar e editar as plantas e doenças que serão apresentadas aos utilizadores.

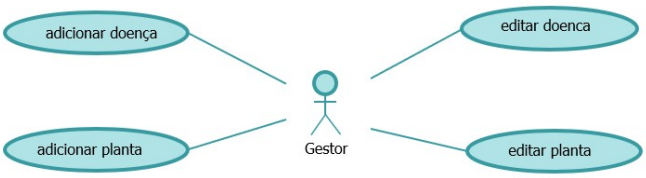


Figura 3.3: Funcionalidades do gestor

Este utilizador é o mais simples, mas é aquele que gere o que será visualizado no lado da aplicação. Com isto, a aplicação terá três tipos de utilizadores.

3.2 Arquitetura do sistema

A proposta de arquitetura para o sistema consiste em criar um sistema dividido em três áreas distintas. A primeira é dedicada à aplicação móvel, que é a área destinada aos utilizadores. A segunda é dedicada ao componente de *AI* do sistema e por último, a área que fornece toda a informação necessária à aplicação, como a autenticação ou a base de dados.

A área de *AI*, é um ambiente criado de forma a otimizar o desenvolvimento de *Python*. Deste modo, é desenhada uma API *Django*, que permite servir de ponto de comunicação do sistema *AI*. A *API* tem como funcionalidade classificar uma imagem, que envolve o pré-processamento de imagem e a utilização de modelos de classificação de imagem previamente treinados.

O terceiro componente é composto pelos serviços que permitem guardar, editar ou obter informação para a aplicação, tais como, a autenticação ou base de dados.

Por último, o componente da aplicação móvel, fica abstraída de qualquer tipo de processamento, ficando apenas dependente de *Internet* para ligar ao servidor AI ou a serviços de base de dados.

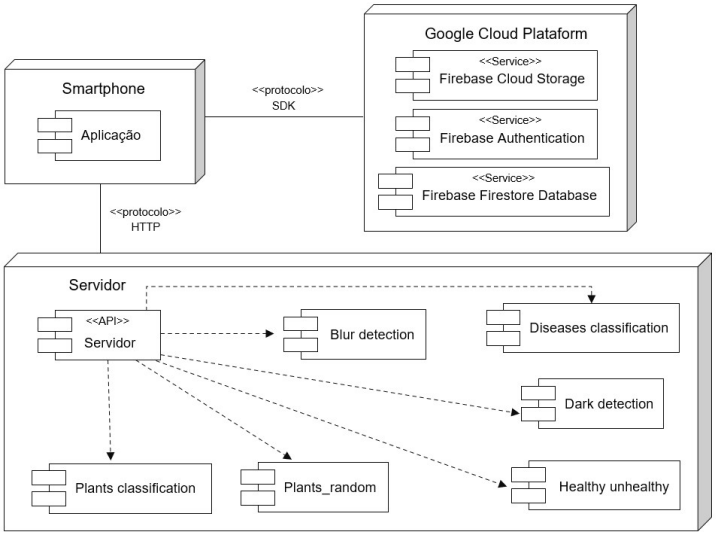


Figura 3.4: Arquitetura do sistema

A vantagem de separar o componente *AI*, é a possibilidade de se poder utilizar o componente *AI* separadamente de outras aplicações ou outros serviços. Na Figura 3.4, podemos ver que o componente Servidor (componente *AI*) é constituído por vários módulos que ajudam no processo de classificação, tanto para a planta como para a doença (processo explicado na secção seguinte). No componente que envolve serviços, optou-se por utilizar os serviços da *Google* e como foi dito, qualquer serviço poderá ser usado aqui, sendo uma arquitetura adaptável. O uso destes serviços serão abordados na Secção 6.2. Assim, sempre que é preciso classificar uma imagem, a aplicação envia a imagem para o servidor *AI*, o servidor *AI* retorna uma resposta para a aplicação e a aplicação pede a informação da planta ou doença classificada ao servidor da base de dados.

3.3 Sistema de classificação

Como foi visto anteriormente, na Figura 3.4, o componente *AI* possui vários módulos. Esses módulos irão ajudar no processo de classificação de uma imagem. Na Figura 3.5, é possível ver o diagrama do sistema de classificação com as diferentes fases.

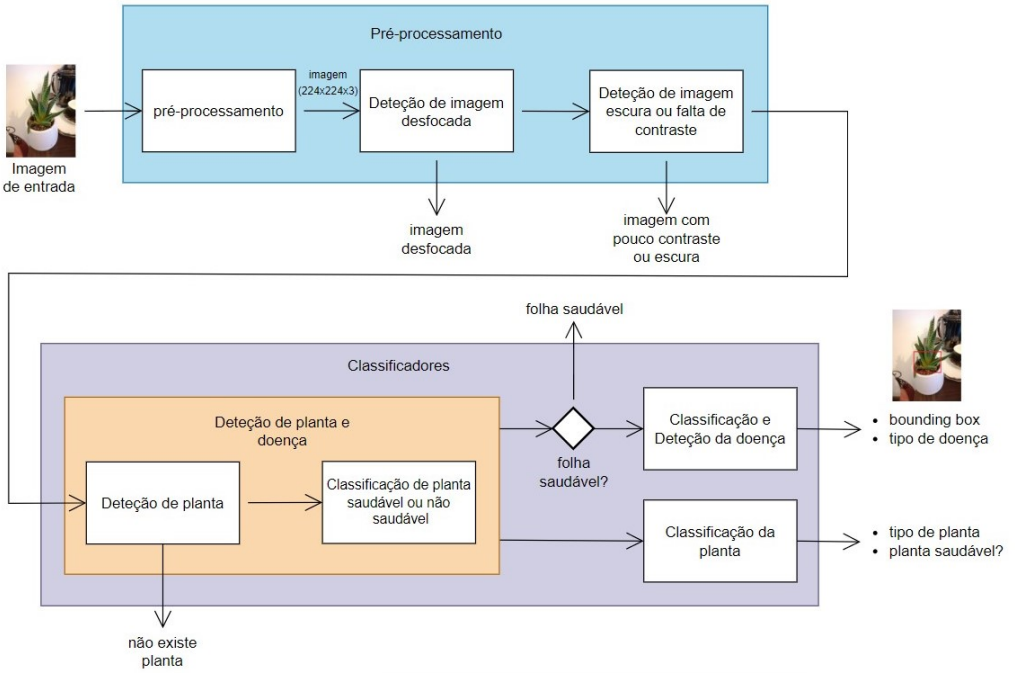


Figura 3.5: Diagrama da arquitetura do sistema de classificação

Podemos ver na Figura 3.5, temos uma imagem de entrada, onde será feito, primeiramente, um pré-processamento e onde é analisado se a imagem está em condições para ser classificada. Neste pré-processamento, a imagem irá sofrer várias modificações para depois ser tratada e melhorada para continuar para as próximas etapas. No bloco dos classificadores, é onde a imagem de entrada, depois de ser analisada, será classificada. Podemos ver, que é possível escolher um de dois caminhos, um para classificar a planta e outro para classificar a doença. Sendo que, quando é para classificar a planta, o sistema devolve o tipo de planta e se a planta está saudável ou não. Quando é para classificar e detetar a doença, o sistema devolve, o tipo de doença e uma caixa delimitadora. Podemos ainda ver que, só entrará no bloco de classificar a doença, caso a folha ou planta a analisar esteja não saudável. Sendo um bloco de classificadores, serão usados classificadores que usam arquiteturas *CNNs*. Todos os métodos que serão utilizados para a implementação deste sistema serão abordados na Secção 4, como por exemplo, o uso das arquiteturas *CNNs*.

Capítulo 4

Métodos

Este capítulo aborda em pormenor as arquiteturas *CNN*, os seus hiperparâmetros e os diferentes métodos usados no pré-processamento. As métricas de desempenho dos modelos *CNN* são mostrados e discutidos na Secção 7.

4.1 Redes Neuronais Convolucionais

Como visto na Secção 2.6, existem várias propostas para a deteção e classificação de doenças em folhas. A maior parte dos estudos, propuseram o uso de arquiteturas baseadas em *CNN*. Nas Secções 2.3.1 e 2.3.2, foi feito um estudo das arquiteturas existentes e das suas vantagens. É ainda usado o algoritmo *Grad-CAM* [45] para detetar a doença na imagem com uma caixa delimitadora.

As arquiteturas *AlexNet*, *MobileNet* e ResNet aceitam imagens *RGB* como entrada e consistem numa sequência de camadas convolucionais, normalizações, ativações *ReLU* e *max pooling*. A última camada *fully connected* com 1000 classes treinadas com a *ImageNet* é substituída para cada um dos problemas. Assim sendo, ambas as arquiteturas foram modificadas para resolver cada um dos problemas neste trabalho, tais como:

- Detetar planta;
- Detetar planta saudável;
- Classificar planta;
- Detetar e classificar doenças.

Detetar planta

Para detetar se existe uma planta na imagem utiliza-se um classificador binário. Foi utilizada a arquitetura *MobileNetV2*. A última camada *fully connected* é substituída por duas camadas densas. A primeira tem 16 unidades com uma função de ativação $ReLU$ e a segunda tem 1 unidade com uma função de ativação *sigmoid*. É também usado, como otimizador, o método Adam.

Detetar planta saudável

Para detetar se a planta é saudável ou se tem algum tipo de doença utiliza-se novamente um classificador binário. Foi utilizada a arquitetura MobileNetV2. A última camada *fully connected* é substituída por uma camada onde será feito um *pooling* médio global, seguido por duas camadas. A primeira camada tem 1024 unidades com uma função de ativação $ReLU$ e a segunda 1 unidade. Sendo este, um problema de classificação binária é usada para a última camada uma função de ativação *sigmoid*. É ainda usado como otimizador o Adam.

Classificar planta

No problema para determinar qual o tipo de planta, é também usada uma arquitetura *MobileNetV2* e a última camada *fully connected* é substituída por uma camada densa com 256 unidades com uma função de ativação $ReLU$, um *droupout* com uma probabilidade 0.25 e uma camada final densa com o número de unidades correspondentes ao número de plantas a classificar. Sendo este, um problema de classificação multi-classe, iremos usar a função de ativação *softmax* com 11 classes (Secção 5).

Deteção e classificação de doença

Para classificar uma doença, é usada a arquitetura *InceptionResNetV2*. A última camada *fully connected* é substituída por uma camada de normalização com *momentum* de 0.99 e *epsilon* de 0.001. Foi adicionado uma camada densa com 256 unidades com uma função de ativação $ReLU$ e regularizadores no *bias* e nos pesos. Foi ainda adicionado um *dropout* com probabilidade de 0.45 e, por fim, uma camada final densa com uma função de ativação *softmax* e com o número de unidades de saída igual ao número de doenças a classificar, que são 5 no total (Secção 5).

Para detetar a doença, incluindo a zona da planta afetada (caixa retangular a limitar a região), é usado o algoritmo *Grad-CAM* (*Gradient-weighted*

Class Activation Mapping) [45, 46, 47, 48]. *Grad-CAM* é uma técnica popular para visualizar onde uma *CNN* está a procurar, que neste caso, queremos procurar a zona da planta afetada. *Grad-CAM* é específico de cada classe, o que significa que pode produzir uma visualização separada para cada classe presente na imagem. O algoritmo utiliza gradientes para produzir um *heatmap*, realçando as regiões importantes na imagem para prever a classificação.

Para produzir um *heatmap*, necessitamos de escolher, primeiramente, uma camada convolucional que iremos analisar. Para isso, a última camada convolucional da rede foi escolhida, a *conv_7b*, como podemos ver na Figura 4.1, onde é apresentado a parte final da rede.

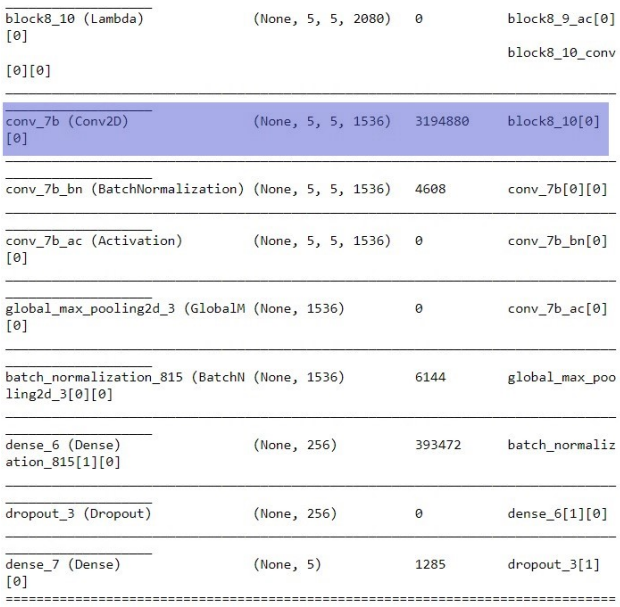


Figura 4.1: Últimas camadas da arquitetura da InceptionResNetV2

A camada denominada *conv_7b* foi selecionada por ser a última etapa no processo de convolução. Consequentemente, todas as modificações realizadas nas camadas precedentes serão refletidas nesta camada. Em outras palavras, essa camada representa o ajuste final no processo de convolução. Tendo a camada escolhida, devemos remover a função de ativação da última camada, ou seja, remover o *softmax*. De seguida, criamos um modelo que mapeia a imagem de entrada para as ativações da última camada convolucional, bem como as predições.

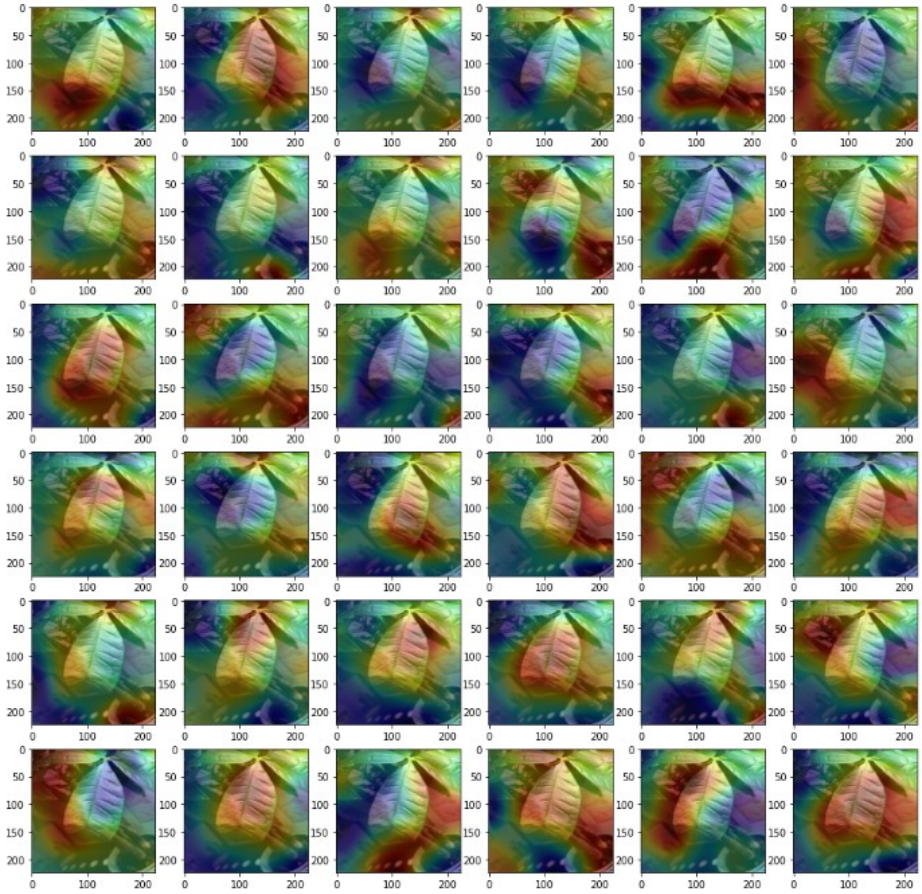


Figura 4.2: Mapas de ativação

Na Figura 4.2, podemos ver 36 mapas de ativação das 1536 que são calculadas na camada *conv_7b*, juntamente com a imagem de entrada, ou seja, em cada *plot* é mostrado uma imagem de entrada e um mapa de ativação. Depois, calculamos o gradiente da classe prevista para a imagem de entrada em relação às ativações da última camada convolucional. Com os gradientes calculados, calculamos a sua média, multiplicamos o canal e somamos tudo.

Figura 4.3: Heatmap gerado com o algortimo Grad-CAM na camada convolucional conv_7b e a sobreposição do heatmap com a imagem de entrada

Na Figura 4.3, podemos ver o *heatmap* calculado com o algoritmo *Grad-CAM*. Na imagem de entrada, a região que mais realça é a zona da doença da planta. A caixa que irá delimitar a área com a doença será obtida a partir desta *heatmap*. É feito um reescalamento do *heatmap* para o tamanho da imagem de entrada e uma binarização por limiar. De seguida, são extraídos os conjuntos conexos, em outras palavras, a região branca. Por fim, calcula-se o retângulo à volta dessa mesma região.

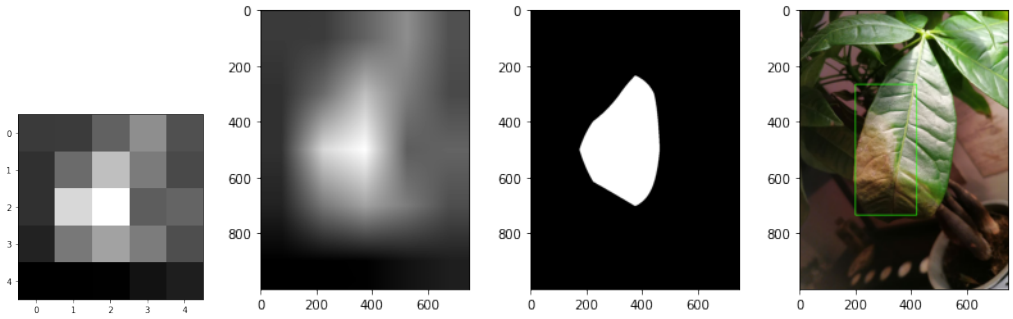


Figura 4.4: Geração de caixa delimitadora

Como podemos visualizar, na Figura 4.4, são precisas 4 fases para a criação da caixa delimitadora. Gerar o *heatmap* com o algoritmo *Grad-CAM*, reescalar o *heatmap* gerado para o mesmo tamanho que a imagem

de entrada, binarizar com um limiar e calcular a caixa do conjunto conexo encontrado. No processo da binarização foi usado um limiar de 170 numa escala de [0,256].

4.2 Data Augmentation

Data augmentation é um pré-processamento convencional usado para aumentar o conjunto de treino adicionando cópias ligeiramente modificadas de dados já existentes. Num contexto do mundo real, pode existir uma variedade de condições, tais como, diferentes orientações, iluminações, etc. Estas condições são resolvidas através da adição de dados modificados sintaticamente para o treino da Rede Neuronal. *Data augmentation* atua como um regularizador e ajuda a reduzir o *overfitting* quando se treina um modelo de aprendizagem automática, que está relacionado com *oversampling* na análise de dados.

No processo de Data augmentation aplicam-se aos dados um conjunto de transformações geométricas para gerar mais dados: inversão, rotação, translação, recorte, manipulações do espaço de cor, entre outras. A operação de inverter uma imagem consiste em espelhar uma imagem ao longo do eixo horizontal ou vertical. A inversão é mais comum ao longo do eixo horizontal do que ao longo do eixo vertical. A translação consiste em deslocar a imagem ao longo do eixo vertical ou eixo horizontal, onde pode ajudar a evitar o *bias* do posicionamento. A operação de recorte é um método onde a imagem é cortada num local aleatório e também proporciona um efeito semelhante às translações. Estes métodos de *data augmentations* [1] podem ser encontradas na *framework Tensorflow* como camadas de pré-processamento e como métodos, nas bibliotecas *tensorflow.keras.layers* e *tf.image*, respetivamente. Nem todos os métodos de data augmentation podem ser aplicados cegamente ao conjunto de dados, sendo que é recomendado a utilização de métodos que se adequam ao problema. Por exemplo, rodar e inverter imagens, geralmente são seguros num conjunto de dados de cães e gatos [2], mas não são seguros num conjunto de dados para tarefas de reconhecimento de dígitos [3], tais como o 6 e o 9. Os métodos têm de ser selecionados cuidadosamente após a compreensão do problema e do conjunto de dados. Nesta tese, serão usados métodos de *data augmentation*, tais como: espelhamento no eixo vertical e horizontal, escalamento, translação, corte aleatório, corte

[1] https://www.tensorflow.org/tutorials/images/data_augmentation
[2] https://www.tensorflow.org/datasets/catalog/cats_vs_dogs
[3] https://www.tensorflow.org/datasets/catalog/mnist

central, manipulação da luminosidade, alterações de contraste, mudanças na saturação e alterações na componente *hue*. Todos estes métodos foram utilizados com a ajuda do ImageDataGenerator [4], que pertence à biblioteca do *Keras*. Este método, permite gerar ou selecionar várias imagens a partir de um conjunto de dados originais, através da aplicação dos vários métodos de *data augmentation*.

4.3 Web Scraping

O *Web scraping* é o processo de recolha de dados a partir de sites de uma forma automatizada. Para o treino dos modelos é necessário uma grande quantidade de dados. Por isso, utilizou-se esta técnica que de forma automática procura encontrar as imagens pretendidas.

4.4 Normalização e redimensionamento dos dados

A normalização é outro pré-processamento convencional e é frequentemente aplicado como parte da preparação de dados para os modelos de aprendizagem automática. Entre as melhores práticas para o treino de uma CNN em que os dados são imagens, é aconselhável dividir todos os valores dos pixeis por 255 (*8-bit RGB*), para que o intervalo de intensidade do pixel seja delimitado no intervalo entre 0 e 1.

Para além dos métodos de pré-processamento vistos acima, ainda foi feito um redimensionamento a todas as imagens. A razão deste redimensionamento deve-se ao facto de os modelos só poderem aceitar tamanhos de imagens iguais, tanto no treino como no teste.

4.5 Melhoria de imagem

Muitas das imagens que são capturadas podem sofrer de vários fatores ambientais que alteram a sua qualidade e podem piorar o desempenho dos modelos, tais como, baixa iluminação, *blur*, etc. Estes fatores têm de ser abordados e, por conseguinte, alguns métodos foram estudados para resolver os problemas.

As *CNNs* alcançaram enormes sucessos em várias áreas, nomeadamente, reconhecimento de imagem, análise de vídeo, processamento de linguagem

[4]https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

natural e muito mais. O desempenho é atribuído a diferentes fatores da arquitetura da rede, tais como: a profundidade das camadas de convolução, as funções de ativação, as camadas de *pooling* e muito mais. Os trabalhos apresentados para deteção e classificação de plantas e doenças não dão muito atenção à qualidade dos dados de entrada que são fornecidos às arquiteturas das *CNNs*. Nesta secção são apresentados os diferentes tipos de pré-processamento utilizados e o seu efeito no desempenho do modelo. Técnicas como o aumento e normalização de dados, como discutido nas Secções 4.2 e 4.4, são conhecidos como pré-processamento convencionais. As técnicas que fazem parte de pré-processamento não convencionais, são: *Gaussian Blur*, correção de iluminação, melhoramento do contraste, conversão de espaço de cores e muito mais.

4.5.1 Método de ajustamento de intensidade

Vários estudos sobre aplicações que usam a câmara foram realizados, mas o desempenho e a capacidade de generalização desses métodos em condições reais permanecem insatisfatórios [33], sendo que, estudos relacionados a plantas não foram encontrados. Entre as condições reais, as variações de iluminação têm um papel essencial no desempenho de aplicações que usam a câmara. Existe um estudo que se baseia em *CNNs* [34] para melhorar a intensidade de baixa luminosidade, mas os estudos baseados em CNN requerem conjuntos de dados contendo pares de imagens de baixa luminosidade, o que é bastante difícil. Por conseguinte, tal como referido em [33], a investigação e análise do método baseado na intensidade poderia ser uma alternativa razoável no uso de *CNN*.

O nível de intensidade é definido como sendo o valor médio do plano de intensidade da imagem, relativo à componente de valor (V) no espaço de cor *HSV*. Após isso, é determinado um limiar a partir da média dos valores das intensidades das imagens de um conjunto de dados, e se a média calculada para uma imagem de entrada estiver a baixo do limiar é considerado como baixa luminosidade. O ajuste dessa intensidade é realizado por:

$$g(x) = f(x) + \beta, \tag{4.1}$$

onde $f(x)$ é a imagem de entrada, $g(x)$ a imagem de saida e β o parâmetro *bias* que é usado para controlar a intensidade. A desvantagem deste método é que o valor β é determinado manualmente.

4.5.2 Equalização de Histograma Adaptativo de Contraste Limitado

Um outro problema para o reconhecimento de imagens não é ter uma imagem escura, mas sim, que tenha um baixo contraste. Pelo que, uma imagem com valor médio de intensidade alta pode produzir os mesmos resultados. Ou seja, uma imagem de baixo contraste tem pouca informação. Existe duas técnicas muito usadas no processamento de imagens, a equalização do histograma e *CLAHE*.

A equalização do histograma [49, 50, 51] é uma técnica que ajusta o contraste de uma imagem [52] através da atualização da distribuição da intensidade de pixeis do histograma de imagem. Esta técnica é adequada quando o histograma da imagem está confinado a uma região em particular, mas pode falhar em situações em que há grandes variações de intensidade. Esta equalização do histograma considera o contraste global da imagem. Em muitos casos, não é uma boa ideia.

Para resolver este problema, é usado a *CLAHE* (Equalização de Histograma Adaptativo de Contraste Limitado). Neste método, a imagem é dividida em blocos. Em cada bloco é feito uma equalização do histograma. Assim, numa pequena área, o histograma limitar-se-à a uma pequena região (a menos que haja ruído). Se o ruído estiver presente nestas regiões, será amplificado. Para resolver este problema, é necessário aplicar uma limitação do contraste. Se qualquer valor no histograma estiver acima do limite de contraste especificado, esses pixeis são cortados e distribuídos uniformemente a outras unidades antes de se aplicar a equalização do histograma. Embora o método *CLAHE* seja mais expendioso a nível computacional, pode produzir melhores resultados do que a equalização do histograma, uma vez que as hipóteses de aumentar o ruído é baixa. Uma das desvantagens, é que os parâmetros para este método, como o limiar e o tamanho dos blocos têm de ser manualmente definidos. Na Figura 4.5, podemos ver a diferença do desempenho da equalização do histograma e *CLAHE*.

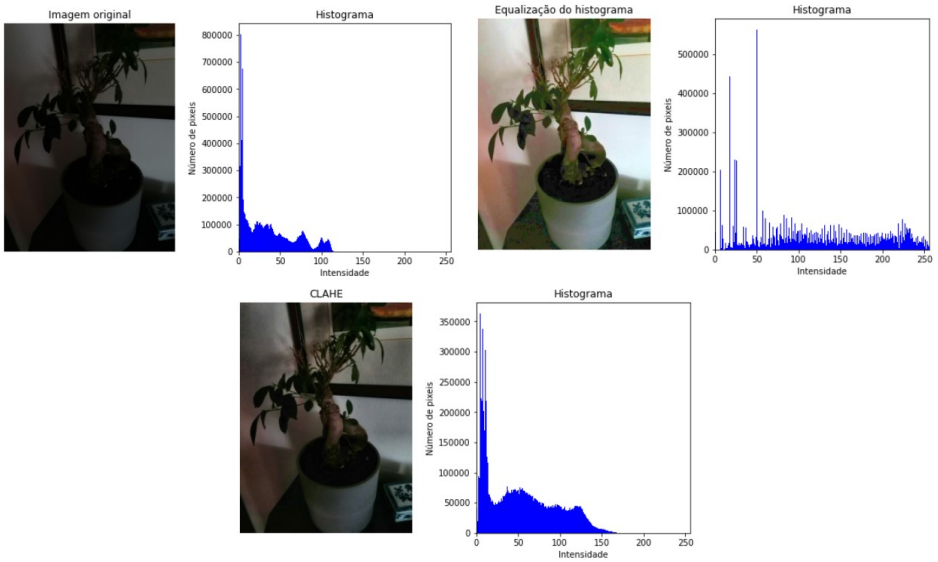


Figura 4.5: Equalização do histograma e CLAHE e os seus respetivos histogramas

Como podemos ver na Figura 4.5, o histograma da imagem de entrada está confinado para o lado escuro, considerando uma imagem de baixo contraste. Não existem valores acima do valor 110, como observado no histograma. O método de equalização distribui a intensidade dos pixeis pela imagem, mas é suscetível em aumentar o ruído e, eventualmente perdendo informação ou detalhe na imagem. Enquanto que, no caso do método *CLAHE*, é feito uma limitação ao contraste.

4.5.3 Deteção de imagens desfocadas

Transformada de Fourier

A Transformada de *Fourier* será usada para acedermos a características geométricas de uma imagem de domínio espacial, para que possamos ver se essa mesma imagem está desfocada.

A Transformada de *Fourier* produz uma imagem de saída com valores de números complexos que pode ser visualizada como duas imagens, seja com a parte real e imaginária ou com a magnitude e a fase . No processamento de imagens, muitas vezes apenas a magnitude da Transformada de *Fourier* é mostrada, pois contém a maior parte das informações da imagem no domínio espacial.

Pode-se aferir se a imagem está desfocada ao calcular a média da imagem

após remover as suas componentes de baixa frequência. O processo de filtrar a imagem por um filtro passa-baixo consiste em calcular a magnitude do espectro da Transformada Discreta de Fourier (DFT), remover as baixas frequências e aplicar Transformada Inversa de Fourier (IDFT) (ver Figura 4.6). Na Figura 4.7, podemos ver a diferença de uma imagem desfocada com uma imagem não desfocada.

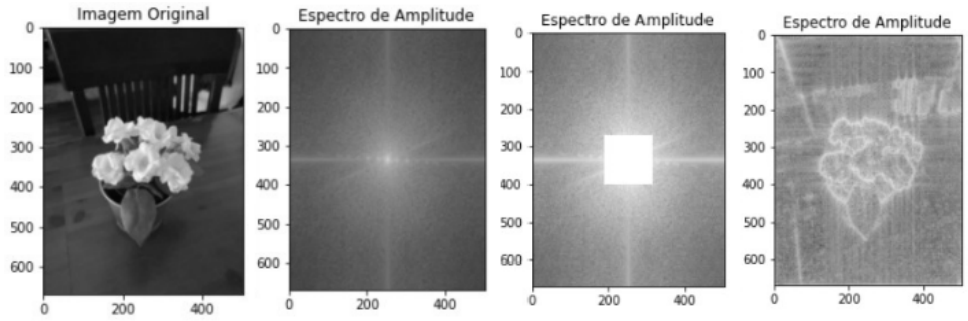


Figura 4.6: Imagem original em tons de cinzento. Espetro da amplitude usando uma transformação logarítmica. Remoção das baixas frequências e imagem final

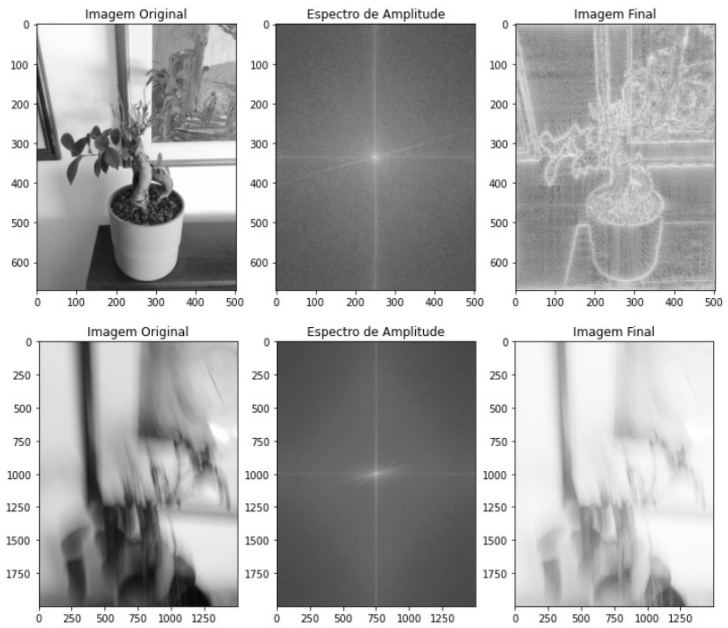


Figura 4.7: Exemplo da utilização da FFT em imagens desfocadas e não desfocadas

Nos espetros de amplitude amostrados na Figura 4.7, conseguimos perceber qual das duas poderá ser uma imagem desfocada, dependendo do seu limiar. No espetro de amplitude da imagem não desfocada existe uma maior variedade de frequências. Nas imagens finais, ao aplicar a *IFFT*, vemos que na imagem não desfocada há um maior contraste do que na imagem desfocada.

Variância de Laplace

Outra abordagem para analisar se a imagem está desfocada, é usar o método da variância de Laplace. Com este método, é possível determinar apenas um único valor para representar o desfoque da imagem. Este método, aplica uma convolução à imagem de entrada com o operador Laplace e calcula a sua variância. Se a variância tiver a baixo de um limiar pré-definido, a imagem é classificada como desfocada.

Existem vários métodos populares para remover o desfoque de imagens, como o Richardson Lucy [53] ou Wiener [54], que utilizam um algoritmo de desconvolução. No entanto, estes métodos exigem a PSF (Point Spread Function) [55, 56], a qual nem sempre está disponível. Existem outros métodos, como a *blind deconvolution* [57], que permite remover o desfoque mesmo quando a PSF não pode ser medida. No entanto, estes métodos não são triviais, a investigação nesta área ainda está em curso e não é garantido que forneçam melhores resultados. Além disso, a *blind deconvolution* é mais complexa e demorada do que a desconvolução não cega, pois estima a PSF após cada iteração. A maioria desses algoritmos trata apenas de um tipo de desfoque, ou seja, é igual em toda a imagem, enquanto as imagens reais têm desfocagens diferentes em várias áreas. Outro método possível é o método Burst, porém, este requer um conjunto de imagens semelhantes à imagem desfocada, o que nem sempre é possível.

Capítulo 5

Conjuntos de dados

Neste projeto foram usados vários conjuntos de dados, incluindo os dados obtidos com a técnica de *Web Scraping* (Secção 4.3), para o treino dos vários métodos estudados, Secção 4. Neste capitulo, serão abordados os diferentes conjuntos de dados usados ao longo de trabalho.

Detetar planta

A identificação de uma planta em uma imagem é um problema de classificação binária. Para isso, usámos os conjunto de treino da *Ashley Poon*, juntamente, com o conjunto de dados das 10 plantas. Estes conjuntos servirão para classificar se existe planta. O conjunto que foi usado para classificar que não existe planta, foi o conjunto de dados *Caltech101* [58]. Este conjunto consiste em 101 classes e cada classe contém, aproximadamente, entre 40 e 800 imagens de diferentes objetos. Esse conjunto antes de ser usado, teve de ser tratado, pois continha imagens com plantas com ruído. Por isso, todos os dados que continham plantas (excluindo, relva, arbustos, árvores,..) foram passados para a classe de plantas. Foram ainda retiradas algumas das classes que não faziam sentido neste problema, tais como, prédios, pneus, aviões de grande porte, entre outros. Esses dados que foram retirados, para além de contribuírem para que o modelo treine mais rápido, também ajudam a evitar que o modelo entre em sobre-aprendizagem. Assim sendo, ficamos com um conjunto onde a classe das plantas contém 7939 imagens e na classe de outros 11691 imagens.

Ainda foi usado o conjunto de dados *256_ObjectCategories* [59] para testar os métodos que serão usados para este tipo de classificação. Este conjunto de dados, é uma extensão dos dados *Caltech101*, contendo num total de 30.607 dados de 256 classes. Sendo usado para esta classificação, este conjunto teve

de ser tratado, dividindo o que é e o que não é planta, ou seja, dividindo em duas classes.

Classificação de plantas

No que se refere ao problema de classificar o tipo de planta, foi usado a técnica *Web Scraping* para construir um conjunto de dados de plantas pela a inexistência desses mesmos conjuntos. Foram escolhidas 10 tipos de plantas domésticas mais comuns:

ficus bonsai : 551

hedera helix : 540

helianthus annuus : 570

hyacinthus : 623

yucca elephantipes : 451

peperomia obtusifolia : 610

opuntia rubescens : 457

haworthia : 735

hippeastrum : 535

roses : 480

others : 678

Total : 6230 imagens

Foi ainda usado como teste extra um conjunto de 50 imagens tiradas com a câmara telemóvel.

Detetar planta saudável

No problema de determinar se a planta é saudável, foi usado o conjunto de dados feito pela *Ashley Poon* [60]. Este conjunto de dados contém várias imagens de vários conjuntos, contendo num total de 1955 imagens, 903 imagens de plantas/folhas saudáveis e 1052 de plantas/folhas não saudáveis. Para teste extra, foi usado 300 imagens tiradas com o telemóvel e imagens da *Internet*.

Detetar e classificar a doença

Como este trabalho está focado em plantas domésticas, e sendo este um problema de detetar e classificar a doença, foi construído um conjunto de dados com dados de doenças respetivas a essas plantas domésticas. No entanto não existe nenhum conjunto de dados que contenha informação de doenças de plantas domésticas para o treino de um modelo *CNN*. Sendo assim, foram usados vários conjuntos de dados para que se pudesse ter um conjunto de dados final com n doenças comuns a essas plantas. Como existem poucos conjuntos de dados que contém doenças comuns que possam incluir nas plantas domésticas, apenas 5 são apresentadas: *Scorch*, *Scab*, *Rust*, *Powdery* e *Spot*.

Figura 5.1: Scorch, Scab, Rust, Powdery, Spot

Para a primeira doença, *scorch*, foi usado o conjunto de dados *Plant Leaf Diseases* (PLD) [61], que usa dados do *PlantVillage* [62]. O conjunto de dados PLD contém 39 classes diferentes de folhas de plantas e imagens de fundo. O conjunto de dados contém num total 61.486 imagens. Mas não foi usado o conjunto todo, foram usados dados que correspondem à classe *Corn_northern_leaf_blight* (985 imagens), por ser, a que mais se assemelha à doença de plantas domésticas, referente à doença *scorch*. Foi ainda usada a técnica de *Web Scraping* 4.3, para enriquecer mais esta classe, acrescentado um total de 227 imagens. Para a doença *scab*, foi usado o conjunto de dados *Plant Pathology* [63], que contém 4 classes, mas apenas será usada para esta classe (592 imagens), a classe correspondente à doença *scab*. Na doença *rust*, foram usados dois conjuntos de dados, o *Plant Pathology* e o *Plant Diseases Recognition* (PDR) [64] (504 e 227 imagens para a doença *rust*, respetivamente). O conjunto de dados PDR, contém 3 classes com um conjunto de 1530 imagens no total. A seguir temos, a doença *powdery*, onde usamos também conjunto de dados *Plant diseases recognition* (500 imagens) e a técnica *Web Scraping* para enriquecer esta classe (164 imagens). E por fim, temos a doença *spot*, usando o conjunto de dados *Pear DiaMOS* [65]. Este é um conjunto de dados para diagnóstico e monitorização de doenças de plantas, recolhidos no terreno, contendo num total de 3505 imagens, mas

só 884 imagens serão usadas para a doença *spot*. Em baixo, podemos ver o número total final de cada classe do conjunto de dados construido:

Scorch	985 (PLD) + 227 (WS) = 1212
Scab	592 (PP)
Rust	504 (PDR) + 227 (622) = 1126
Powdery	500 (PDR) + 164 (WS) = 665
Spot	884 (PDM)

Ainda foi usado mais 52 imagens como teste extra usando a câmara de um telemóvel.

Deteção de imagens desfocadas

Para avaliar os métodos no que diz respeito à deteção de imagens desfocadas, foi usado o conjunto de dados *CERTH* de imagens desfocadas [66]. O conjunto de dados de imagem *CERTH* consiste em 2480 imagens digitais, das quais 1850 são fotografias tiradas por vários modelos de câmaras em diferentes condições que não foram alteradas após a sua captura. As restantes, são imagens desfocadas artificialmente. Para produzir estas imagens foram selecionadas aleatoriamente 60 imagens não distorcidas e depois foram aplicados vários tipos de filtros gaussianos, de movimento e de média circular.

Capítulo 6

Desenvolvimento da aplicação

Nesta secção serão abordadas todas as etapa do processo de desenvolvimento da aplicação *HousePlant*, nomeadamente, o hardware, as bibliotecas, a base de dados e a implementação da aplicação. Será ainda desenvolvida uma API para interagir com os modelos implementados.

6.1 Hardware

Para a realização deste projeto, foi utilizado um computador cujo o *CPU* é um *Intel Core i5-6300HQ @ 2.30GHz*, a *GPU* é uma *GeForce GTX 950M* e a a *RAM* de 8,00 GB.

6.2 Base de dados

É usada uma base de dados que tem como papel o armazenamento de informação, relativamente às plantas e doenças, bem como, nome da planta, nome da doença, tratamentos a proceder a uma determinada doença, entre outros. Ainda é possível armazenar imagens das respetivas plantas e doenças e armazenar a informação dos utilizadores. É utilizada a base de dados *NoSQL* disponibilizado pelo *Firebase*. O *Firebase* é uma plataforma de desenvolvimento de aplicações que ajuda a criar, melhorar e expandir a aplicação. Podemos dizer que é um *Backend-as-a-Service* (*Baas*), ou seja, é um modelo que fornece aos desenvolvedores de aplicações de Web e mobile uma maneira de ligar as suas aplicações à *cloud* de armazenamento e *APIs* expostas por aplicações de *backend*, além de fornecer recursos como gestão de utilizadores, notificações *push* e integração com serviços de redes sociais. Existe uma variedade de tecnologias do lado do servidor disponíveis no mercado para desenvolvedores testarem novas possibilidades de uso e o *Firebase*

é uma das que mais tem atraído a atenção no mundo dos dispositivos móveis. Além disso, o *Firebase* é versátil e os tipos de aplicações que podem ser desenvolvidos com *Firebase* são: *Android*, *iOS* e *Web*.

Toda a sua base é construída na infraestrutura da *Google*, sendo categorizada como um programa de base de dados *NoSQL*, que armazena dados em documentos do tipo *JSON*.

Uma das vantagens de usar o *Firebase* neste projeto, é que a plataforma oferece um grande número de serviços que não é preciso implementar, tais como:

- Segurança na comunicação e transferência de dados;
- Limitações de infraestrutura;
- Compatibilidade com vários tipos de dispositivos diferentes;
- Autenticação.

Todos estas situações são possíveis de serem solucionadas por meio dos serviços oferecidos pelo *Firebase*. Além disso, o *Firebase* é gratuito. Os serviços que iremos usar, são:

- Firestore Database;
- Cloud Storage;
- Autenticação.

A *Firestore Database* é uma base de dados flexível e escalável para desenvolvimento focado em dispositivos móveis, Web e servidores. A *Cloud Storage* oferece uma forma prática de guardar arquivos e imagens. O serviço tem o seu próprio sistema de regras de segurança para proteger os arquivos para os diferentes tipos de utilizadores. A *Cloud Storage* foi concebida para oferecer desempenho ideal, confiabilidade, escalonamento automático e usabilidade de referencia. A autenticação é suportado pelo o uso de senhas, números de telefone, perfil do *Google*, entre outros.

6.2.1 Modelo de dados

O modelo de dados desenvolvido cumpre com os requisitos propostos neste projeto, sendo composto por sete tabelas: *diseases*, *managers*, *histPlant*, *histDiseases*, *myPlants*, *plants*, *registeredUser*.

Modelo Entidade-Associação

Na Figura 6.1 é apresentado um modelo entidade-associação, que se trata de uma técnica para modelar os dados de um sistema de informação, sendo esse, um modelo conceptual independente da tecnologia. Tendo como vantagens, questionar regras da organização, salientar novas necessidades de informação e revelar incoerências atuais.

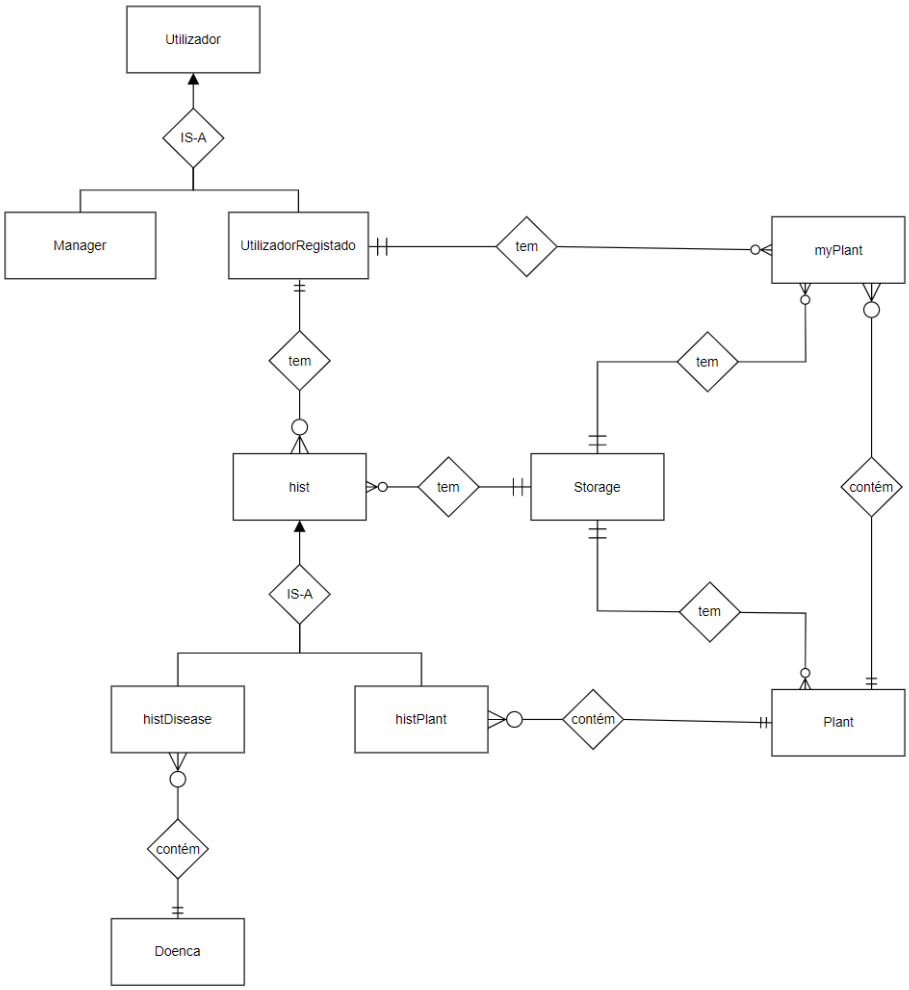


Figura 6.1: Modelo Entidade-Associação

Podemos ver, no modelo da Figura 6.1, é representado por um conjunto de blocos, sendo esses, apresentados como entidades. As entidades são uma abstração para a descrição de objetos ou conceitos que possuam um conjunto de características comuns. Podemos ainda ver que, tanto como a entidade utilizador, como a entidade *hist* são generalizações de outras duas entidades cada uma. Generalizações são uma representação de entidades que possam conter atributos iguais. Atributos são características comuns aos objetos ou conceitos que a Entidade retrata. A baixo, serão apresentados as diferentes entidades com os respetivos atributos.

Utilizadores

Em baixo são apresentados as entidades e os seus atributos referentes aos utilizadores. Estas entidades servirão para saber que utilizadores estão registados e quais são os seus privilégios. Os atributos, apresentados são: id, email e password.

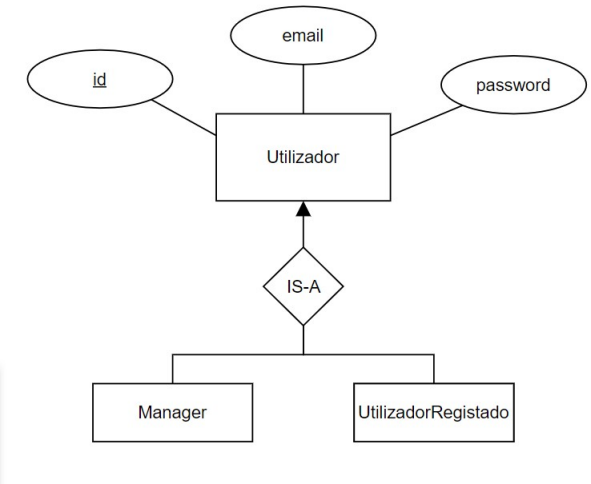


Figura 6.2: Entidades do utilizador

Onde, o id é um identificador, o email e a password são as credenciais que o utilizador, seja gestor ou utilizador registado, para se autenticar.

Storage

Podemos ver que a entidade *storage* contém um identificador e uma imagem associados. Esta entidade terá como objetivo armazenar todas as imagens que serão adicionadas.

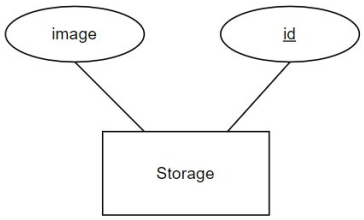


Figura 6.3: Entidade Storage

Plantas

Em baixo são apresentados os atributos da entidade planta. Esta entidade têm como objetivo guardar a informação de diferentes plantas que serão apresentadas aos utilizadores.

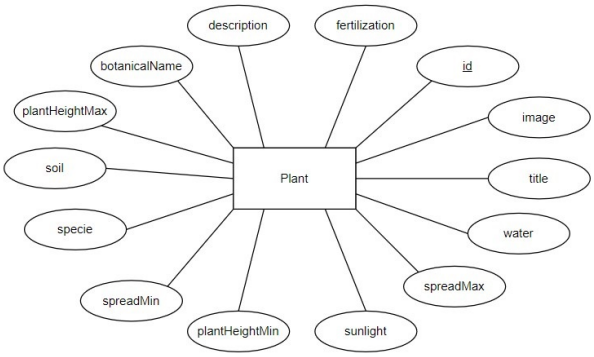


Figura 6.4: Entidade Plant

Cada planta terá um identificador associado, juntamente com uma descrição, um título, um nome botânico e o nome da espécie. Ainda foram adicionados atributos para indicar qual o tamanho mínimo ou máximo da planta, tanto em comprimento como em largura. Por fim, temos atributos que associam ao tratamento de como manter uma planta doméstica.

Doenças

Em baixo são apresentados os atributos da entidade doença. Esta entidade tem como objetivo guardar a informação de diferentes doenças que serão apresentadas aos utilizadores.

Cada doença terá um identificador associado, juntamente com várias descrições que ajudarão de como tratar essa doença. Ainda tem como atributos

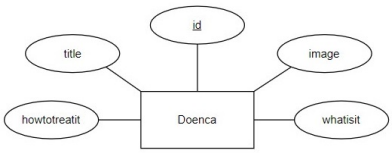


Figura 6.5: Entidade Disease

uma imagem e um título.

As minhas plantas

Para esta entidade, temos a entidade *myPlant*, que terá a informação das plantas domésticas adicionadas pelos utilizadores registados, organizadas pelas localizações da casa (e.g. casa de banho, sala de jantar, cozinha).

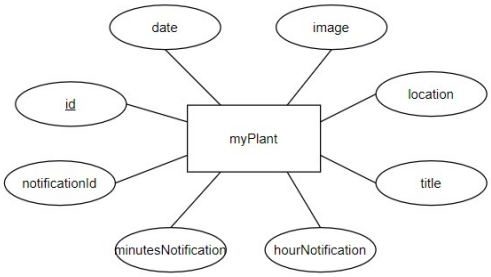


Figura 6.6: Entidade myPlant

Para isso, foi adicionado como atributos, um identificador, um título e uma *imagem* para poder descrever a planta adicionada. Um atributo *localização* para adicionar a localização da planta na casa. E uma data que indicará quando é que a planta foi adicionada. Os utilizadores poderão ser notificados para saber que têm de tratar das plantas. Assim, foram adicionados três atributos para descrever as notificações: *notificationId*, *minutesNotification* e *hourNotification*. O *notificationId* será utilizado para mostrar se o utilizador que ser notificado ou não. Caso seja vazio esse atributo, a notificação não será apresentada. Os outros dois, servem para indicar para quando é que o utilizador deverá ser notificado (horas e minutos da notificação). Por fim, temos um atributo associado à localização de um espaço da casa.

Histórico

Finalmente, temos as entidades referentes ao histórico. Tal como acontece nas entidades utilizadores, existe uma generalização para duas entidades, o

histórico de plantas e o histórico de doenças. Estas entidades terão como objetivo guardar as plantas e doenças identificadas pelo utilizador.

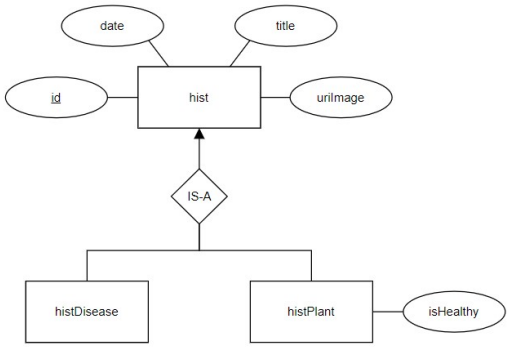


Figura 6.7: Entidades do histórico

Com isto, temos um identificador, um título e uma data de quando foi identificado.

Modelo não relacional

Como mencionado, será usado o *Firebase* como base de dados, sendo que algumas das entidades não serão precisas de implementar, já que a *Firebase* apresenta algumas dessas funcionalidades. A entidade *Storage*, será removida e não será implementada, a *Firebase* já oferece essa funcionalidade de armazenar imagens na base de dados (*cloud storage*). Assim sendo, todas as outras entidades que possuem imagens, terão de ter um atributo como identificador da imagem a usar da *storage*.

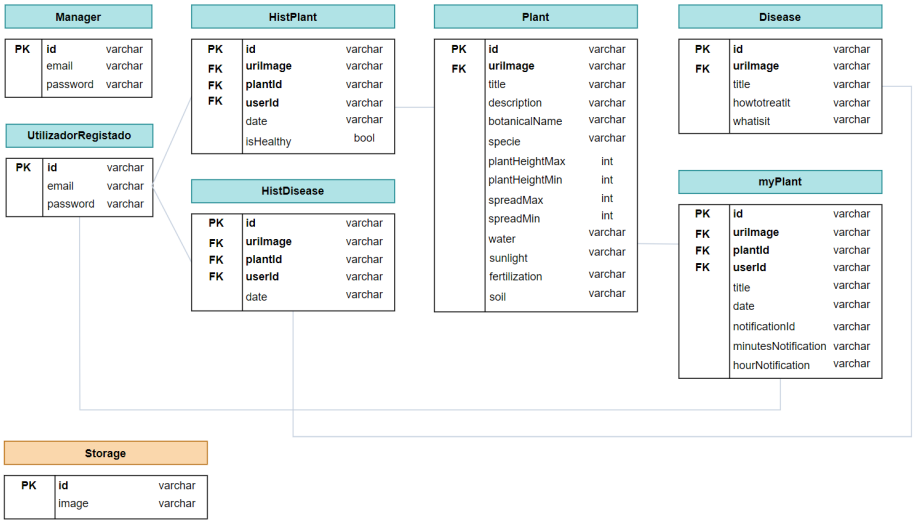


Figura 6.8: Modelo não relacional

A base de dados é constituída por 7 tabelas: *Manager*, *userRegistered*, *HistPlant*, *HistDisease*, *Plant*, *Disease* e *myPlant*. Todas as tabelas serão implementadas usando a interface da *Firebase*.

6.3 API desenvolvida

O desenvolvimento de uma *API* é um dos pontos essenciais de toda a arquitetura do sistema desenvolvido. Tem como objetivo interagir com toda a parte *AI* desenvolvida em *Python*. Foi desenvolvida em *Python* usando a biblioteca *Django* e as principais funcionalidades desta *API* é classificar o tipo de planta e o tipo de doença, oferecendo as respetivas informações de cada classificação. O desenvolvimento das funcionalidades distribuído sobre a forma de *API* permite uma vez mais, efetuar a ligação entre os ambientes, servidor *AI* e aplicação móvel, utilizando o protocolo *HTTP*. O servidor *AI* foi implementado numa máquina local. E visto que, a arquitetura do sistema é flexível, o servidor *AI* poderá ser adicionado a um servidor público (usando por exemplo uma máquina virtual) para que possa ser acedido publicamente.

Esta API terá três métodos (três endpoints):

- Um método para verificar se a *API* tem ligação;
- Um método para classificar uma imagem identificando a planta;

- E por fim, um método para classificar uma imagem identificando e detetando a doença.

O primeiro *endpoint* (Tabela 6.1), serve para saber se existe alguma conexão entre os ambientes, caso não exista, será possível avisar o utilizador.

Serviço	Plant
Endpoint	/api/index
Método	POST ou GET
Descrição	Este método tem como objetivo perceber se existe ligação ao servidor
Parâmetros	--

Resposta	{
	'type' : string,
	'date' : string
	}

Tabela 6.1: Endpoint para o método que vê se existe ligação com o servidor AI

No segundo (Tabela 6.2), temos um método que serve para classificar uma imagem identificando a planta, que inclui uma imagem como parâmetro e retorna a predição da classificação da planta, o nome da planta e se a planta está saudável.

Endpoint	/api/plant
Método	POST
Descrição	Este método devolve a classificação do tipo de planta numa imagem

Parâmetros	{
	"image": string
	}

Resposta	{
	"msgtype" : string,
	"pred" : string,
	"name_plant" : string,
	"msgHealthy" : string
	}

Tabela 6.2: Endpoint para o método que classifica uma planta a partir de uma imagem

Por fim, temos um terceiro método (Tabela 6.3), que serve para classificar e detetar a doença numa imagem que inclui também como parâmetro uma imagem e retorna a sua predição, o nome da doença e os pontos de referência

(x, y, comprimento e altura) de uma caixa delimitadora onde a doença foi detetada. Como iremos enviar dados que podem pesar e não poderão ser enviados simplesmente pelo o texto do *url* como parâmetro, mas sim, pelo corpo, iremos usar como método *HTTP*, o *POST*.

Endpoint	/api/disease
Método	POST
Descrição	Este método devolve a classificação do tipo de doença numa imagem

Parâmetros	{
	"image": string
	}

Resposta	{
	"msgtype" : string,
	"pred" : string,
	"name_disease" : string,
	"boundingbox" : {
	"x": x,
	"y": y,
	"w": w,
	"h": h
	}
	}

Tabela 6.3: Endpoint para o método que classifica uma doença a partir de uma imagem

6.4 Aplicação HousePlants

Um dos objetivos deste trabalho é implementar uma aplicação móvel que comunica com o servidor AI e os serviços da Google. Para isso, iremos usar o *framework React Native* para implementar a aplicação. *React Native* é um *framework open-source* de *JavaScript*, concebido para construir aplicações em múltiplas plataformas, nomeadamente, *iOS*, *Android* e também *Web*, utilizando a mesma base de código. *React Native* é baseado em *React* que foi introduzido pela *Meta*.

Relativamente às funcionalidades que foram mencionadas, temos três tipos de utilizadores: o utilizador comum, o utilizador registado e o gestor. E todos esses utilizadores conseguem fazer *login*, entrando pela página de inicio da aplicação. Não é possível criar conta para um gestor mas é possível criar conta para um utilizador. Para criar uma conta gestor terá de ser a partir da interface *Firebase*.

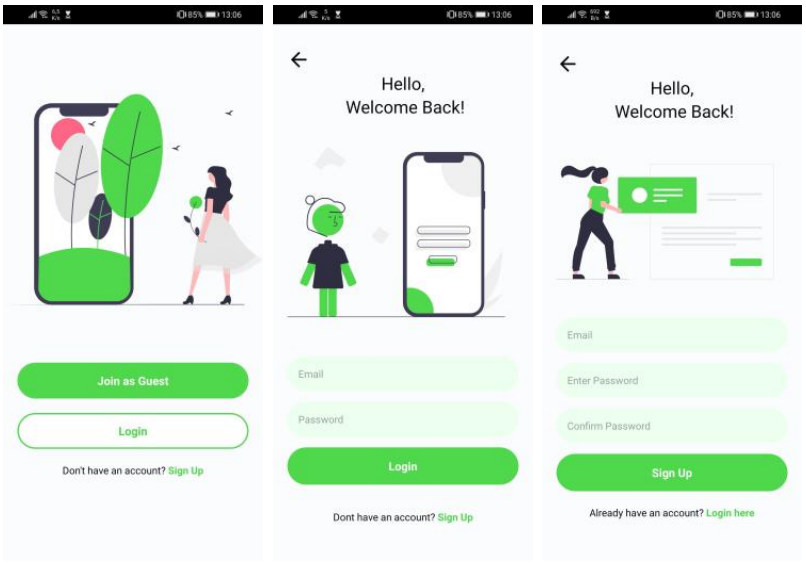


Figura 6.9: Página de inicio, login e registo

O utilizador para se registar terá de introduzir o seu email e uma palavra-passe (Figura 6.9). Não será preciso inserir o username ou o nome, porque não será partilhável o perfil.

Tanto o utilizador comum como o utilizador registado podem visualizar as plantas e doenças que estão disponíveis na aplicação (Figura 6.10).

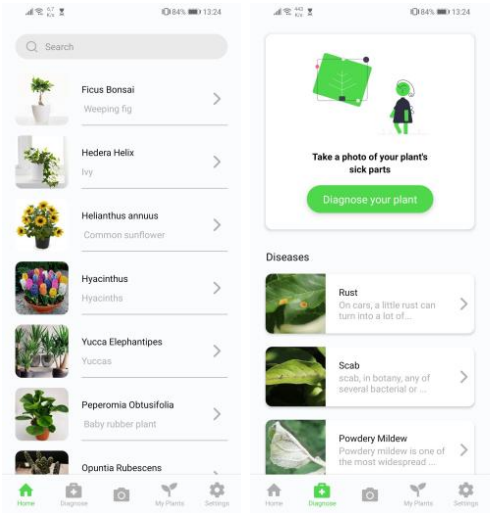


Figura 6.10: Página de principal e página de doenças

Ambos os utilizadores conseguem aceder à página principal, que é a

página que contém uma lista de plantas e, à página de doenças, que contém as doenças e onde será possível diagnosticar uma planta clicando no botão verde (Figura 6.10). Para visualizar a página de detalhe de uma planta ou de uma doença o utilizador tem de clicar em um elemento das listas apresentadas.

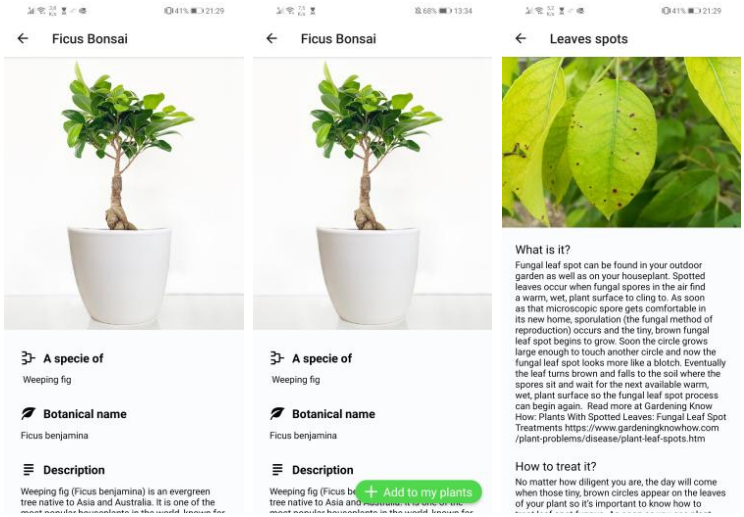


Figura 6.11: Página de detalhe da planta e doença

Um utilizador registado, quando entrar numa página de detalhe de uma planta (Figura 6.11), pode querer adicionar essa planta ao perfil de plantas. Assim sendo, foi adicionado um botão no canto inferior direito, para quando o utilizador esteja registado, possa adicionar a planta ao seu perfil, como é mostrado na Figura 6.11 na segunda imagem.

Para detetar uma planta e obter as suas informações, é utilizado o ícone da câmara do menu de navegação (Figura 6.10). Esse menu de navegação terá cinco páginas diferentes, nomeadamente, a página principal, a página de doenças, a câmara para a identificação de uma planta, as minhas plantas e as definições da conta. Para o uso da câmara, será necessário o uso do servidor *AI*, já que usará os métodos fornecidos pela *API* para classificar a imagem. Quando a aplicação não tiver conexão ao servidor, será apresentado uma mensagem ao utilizador (Figura 6.12), dizendo que o servidor poderá estar em baixo e para tentar novamente mais tarde.

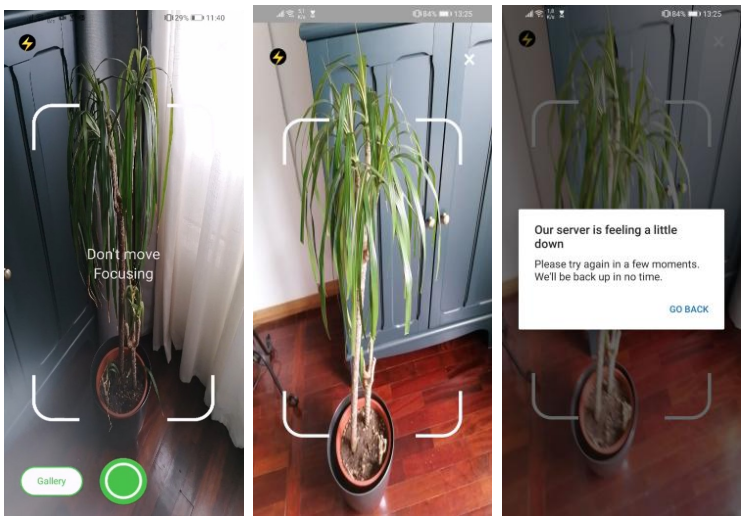


Figura 6.12: Deteção com a câmara e mensagem de aviso do servidor em baixo

No processo de diagnosticar ou identificar uma planta poderá ser apresentada uma mensagem ao utilizador caso a imagem não esteja em condições para ser analisada (Figura 6.13).

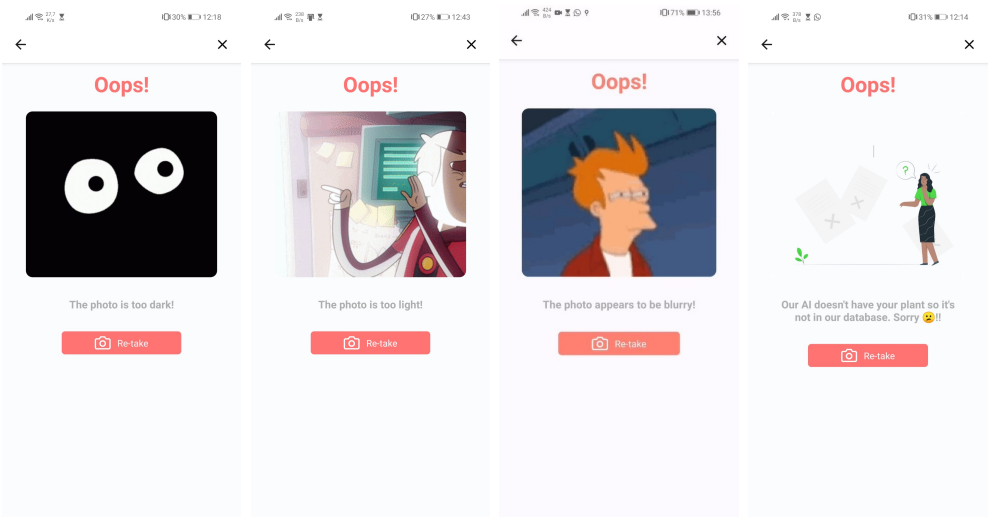


Figura 6.13: Avisos para imagens tiradas sem condições

No caso de diagnosticar uma planta com uma imagem em condições, será apresentado um ecrã a dizer se a planta está ou não saudável (imagem de 1 a 3 da Figura 6.14) . No caso de identificar uma planta, o utilizador será

encaminhado para a página de detalhe da planta. Nessa página poderá aparecer um aviso (botão) vermelho caso a planta não esteja saudável (imagem 4 da Figura 6.14). Esse botão encaminhara para a câmara.

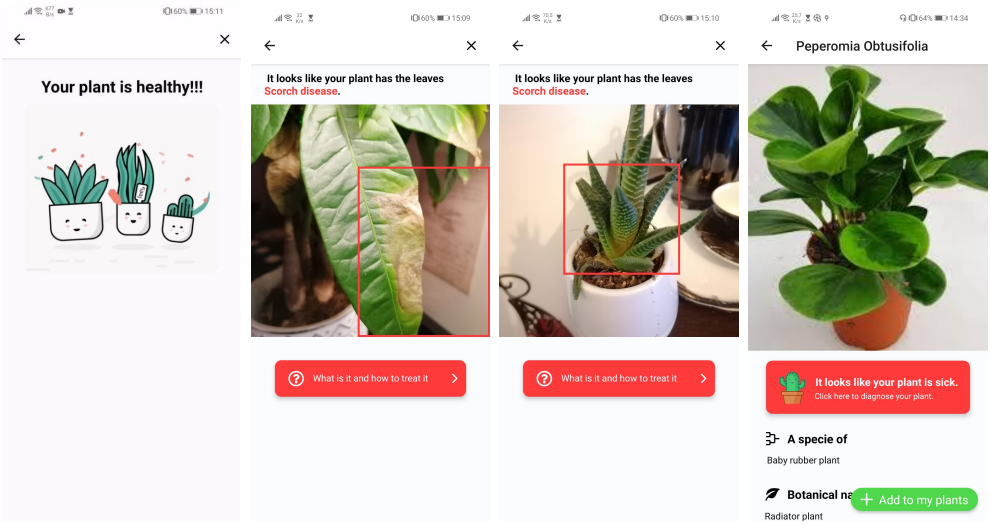


Figura 6.14: Diagnóstico e deteção da planta

Para as últimas duas funcionalidades, só o utilizador com conta terá permissão para aceder, por isso, será apresentado uma página de aviso caso o utilizador não esteja registado (Figura 6.15).

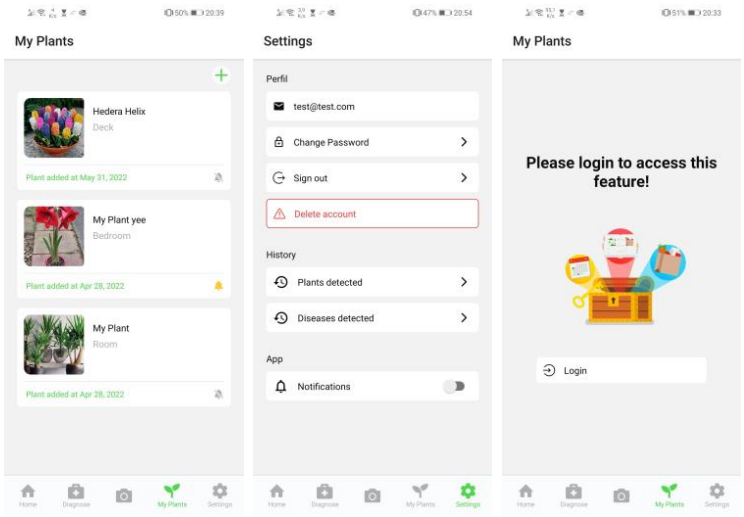


Figura 6.15: Página de perfil das plantas, página de definições e página de acesso negado

Na Figura 6.15, é possível visualizar duas páginas: as minhas plantas e as definições. A página das plantas é onde o utilizador pode adicionar plantas consoante o tipo de planta e o local da casa onde a planta se situa, clicando no botão branco com um ícone verde no canto superior direito (Figura 6.15). Ainda é possível adicionar um lembrete para avisar o utilizador de quando deve tratar da planta. É possível visualizar essas funcionalidades na Figura 6.17. Na página de definições, é onde o utilizador pode editar a sua conta, visualizar o histórico das deteções feitas e ativar as notificações.

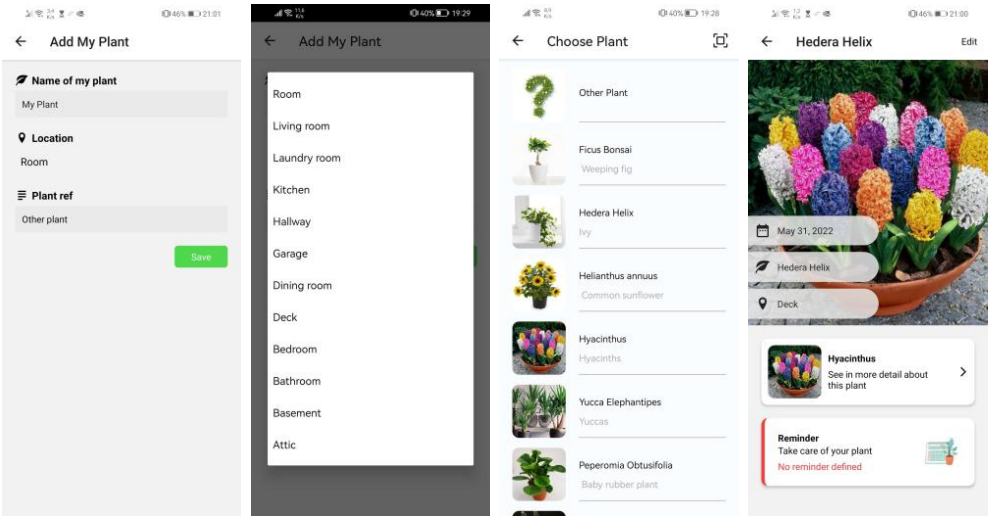


Figura 6.16: Inserir uma planta no perfil

Podemos ver que, na Figura 6.16, a planta que é adicionada é referenciada a um tipo de planta existente para conseguir aceder às informações da respetiva planta. Caso a planta não exista na aplicação, essa planta fica sem referência e não terá nenhuma informação associada. Poderá ainda pôr o local onde a planta se situa na casa.

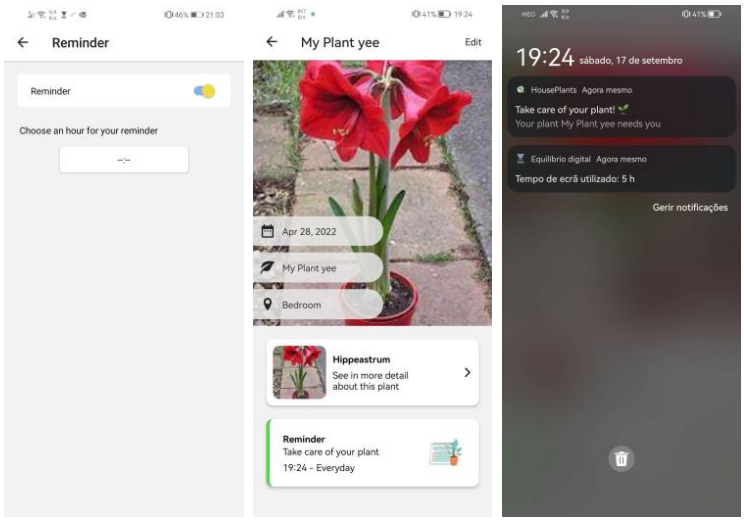


Figura 6.17: Adicionar um alerta e receber uma notificação

Ao adicionar uma planta, em baixo aparece uma caixa destinada à notificação da planta. A notificação servirá para notificar ao utilizador de não esquecer de tratar da planta a uma determinada hora do dia. Quando adicionada, essa caixa aparece a vermelho porque não tem nenhum alarme ativo. Ao selecionar essa caixa, poderá ativar o alarme escolhendo a hora a que pretende receber a notificação. Na Figura 6.17, podemos ver que foi adicionado um alarme às 19:24 e recebeu precisamente à mesma hora uma notificação.

Por fim, temos as páginas para o gestor (Figura 6.18), onde o gestor pode adicionar e editar plantas e doenças à aplicação.

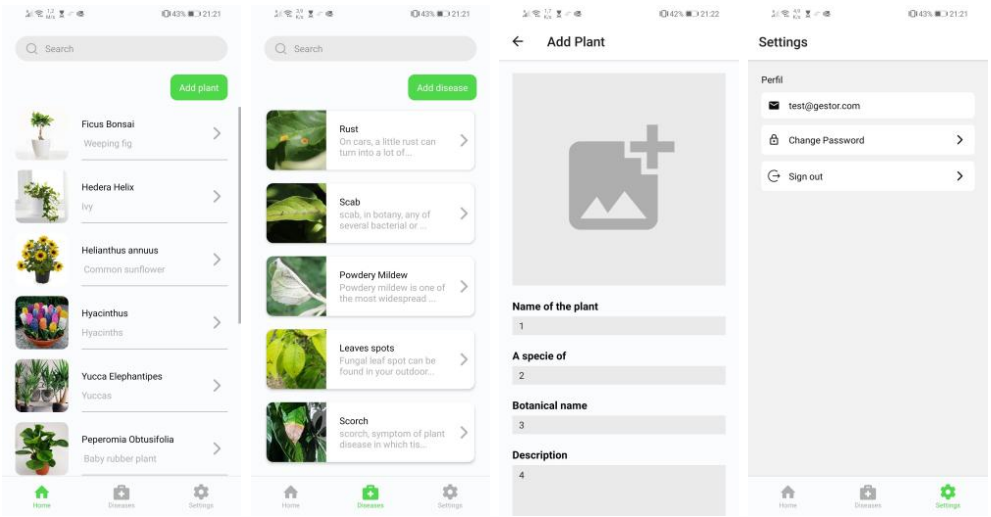


Figura 6.18: Páginas do gestor

É possível visualizar, na Figura 6.18, as diferentes páginas a que o gestor tem acesso.

Capítulo 7

Avaliação dos modelos

Esta secção introduz o conjunto de dados usado neste projeto e discute os resultados obtidos através de várias experiências realizadas com os métodos discutidos na Secção 4.

Para a criação de um modelo de classificação é necessário obter dados e estabelecer os parâmetros adequados. Esses parâmetros são fundamentais para ajustar o modelo, e sua variação pode impactar significativamente nos resultados obtidos, influenciando tanto no tempo de treino quanto na taxa de acerto. Por isso, é fundamental determinar quais parâmetros são relevantes para o problema em questão e selecioná-los de forma cuidadosa, garantindo que o modelo seja capaz de generalizar bem para novas amostras. É importante destacar que o ajuste de parâmetros é um processo iterativo que pode ser realizado com o auxílio de técnicas de otimização, como o uso de algoritmos de busca de hiperparâmetros ou a validação cruzada.

7.1 Metodologia de treino e teste

Relativamente aos dados introduzidos no processo de treino e tal como foi abordado na Secção 6.2, os dados que irão ser utilizados foram obtidos por várias fontes. Uma parte foi retirada de vários *datasets* disponíveis. A outra parte foi gerada utilizando métodos para aumentar os dados.

No entanto, para efetuar o processo de treino é comum fazer uma divisão dos dados, sendo que existem duas partes, a parte utilizada durante a fase de treino do modelo e a parte utilizada para testar o modelo. Esta última parte é importante, pois é com os dados desta fração que se vai analisar os resultados.

Existem dois métodos muito utilizados para a divisão dos dados, o método *Hold-out* e o método *Cross Validation*. O primeiro método consiste em dividir, por exemplo, $\frac{2}{3}$ dos dados para treino e o $\frac{1}{3}$ para teste. O segundo método consiste em dividir os dados em vários subconjuntos. O método *Cross Validation* pode ser o mais eficiente, no entanto o método *Hold-out* é mais simples. Como o *Cross Validation* usa vários conjuntos de treino e teste, requer mais tempo e poder computacional. Deste modo, para o trabalho deste projeto e de acordo com as componentes hardware disponíveis, será utilizado o mecanismo de *Hold-out*, devido à utilização de menos recursos e simplicidade do mecanismo.

	Nome do método	Treino (%)	Teste (%)	Dados de treino	Dados de teste	Dados Totais	Número de classes
Modelos CNN	Detetar planta	75	25	14722	4908	19630	2
	Detetar planta saudável	75	25	1466	489	1955	2
	Classificar planta	75	25	4672	1558	6230	11
	Detetar e classificar doença	90	10	4032	446	4479	5
Outros métodos	Deteção de imagens escuras	-	100	-	81	81	2
	Deteção de imagens desfocadas	-	100	-	2480	2480	2

Tabela 7.1: Metodologias de treino e teste

Nos modelos *CNN* serão usados o método *Hold-out*, mas para outros métodos, como a deteção de imagens desfocadas ou deteção de falta de intensidade, não será preciso de dividir os dados, pois, não haverá nenhum treino de modelo. Por isso, para esses métodos serão usados os dados por inteiro.

7.2 Métricas de avaliação

Neste trabalho, iremos ter dois tipos de classificações, binárias e multiclasse, sejam métodos que usam modelos *CNN* ou métodos mais simples. Por isso, nas classificações binárias serão usadas as seguintes métricas: a matriz de confusão e probabilidade de acertos. Adicionalmente, para cada avaliação será ainda avaliado o tempo de treino e o tempo que demora a prever os resultados. Para além destas, serão ainda usadas outras métricas que serão mostradas mais abaixo.

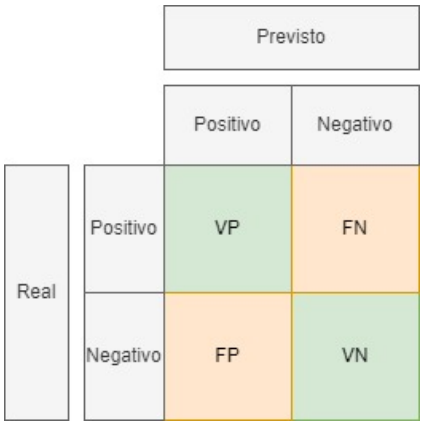


Figura 7.1: Matriz de confusão para classificação binária

A matriz de confusão é uma matriz que permite, para cada classe do modelo, visualizar as frequências de classificação:

- Verdadeiros positivos (VP), que ocorrem quando no conjunto de dados, a classe que se está a analisar é a classe obtida do resultado da previsão.
- Falsos positivos (FP), que ocorrem quando no conjunto de dados, a classe que se está a analisar não foi a classe obtida do resultado da previsão
- Falso verdadeiro (FV), que ocorrem quando no conjunto de dados, a classe que não se está a analisar foi a classe obtida do resultado da previsão.
- Falso negativo (FN), que ocorrem quando no conjunto de dados, a classe que não se está a analisar, não foi a classe obtida do resultado da previsão.

Com estes dados, é possível obter as seguintes métricas:

- Accuracy – define a percentagem de acerto do modelo,

$$\text{accuracy} = \frac{VP + FV}{VP + FP + FV + FN}; \tag{7.1}$$

- Recall ou TPR – define a proporção de positivos que foram identificados corretamente (taxa dos valores verdadeiros-positivos),

$$\text{recall} = \frac{VP}{VP + FN} \tag{7.2}$$

- Precision – define a proporção de identificações positivas que são corretas,

$$\text{precision } = \frac{VP}{VP + FP} \tag{7.3}$$

- F1-Score – demonstra o balanço entre a precision e o recall,

$$\text{f1-score} = 2 * \frac{\text{precision } * \text{ recall}}{\text{precision } + \text{ recall}} \tag{7.4}$$

- FPR – define a taxa dos valores falsos-negativos,

$$\text{F P R} = \frac{\text{FP}}{\text{FP } + \text{ VN}} \tag{7.5}$$

Para além das métricas mencionadas acima, será usado também a curva de *ROC* e a curva PR. A curva de *ROC* é uma curva que visualiza os valores entre *TPR* no eixo do y e *FPR* no eixo do x. Basicamente, é calculado o *TPR* e o *FPR* para diferentes limiares à saída de um classificador binário. Quando maior o *TPR* e menor o *FPR*, melhor é o limiar. A curva PR, similar à curva de *ROC*, é uma curva dos vários valores de *precision* e *recall* para vários valores de limiar.

Para a classificação multi-classe, será usado também a matriz de confusão e a probabilidade de acertos. A única diferença é que a matriz de confusão terá mais de duas entradas, sendo que a *accuracy* pode ser demonstrada a partir da diagonal da matriz de confusão.

7.3 Parâmetros a avaliar

Nesta secção serão apresentados para os vários algoritmos implementados várias experiências e em cada experiência serão avaliados vários parâmetros. É importante aferir os resultados obtidos com a manipulação dos parâmetros, de modo a verificar, quais as melhores opções. Como a maior parte dos métodos utilizam modelos *CNN*, serão apresentados alguns do parâmetros que foram utilizados:

- Taxa de aprendizagem (learning rate);
- Números de épocas;
- Tamanho do batch;

- Otimizador;
- Método do erro;
- Camadas.

A taxa de aprendizagem é um parâmetro que permite definir a velocidade a que uma rede ao treinar altera os seus pesos, sendo que quanto maior for essa taxa maior é a variação dos pesos. O número de épocas é o número de iterações que a rede demorá a aprender. Sendo que o número de épocas faz variar o tempo do treino do modelo. O tamanho do *batch* define a quantidade de dados de um conjunto de dados que são utilizados em cada ciclo de iteração de uma época. Ou seja, uma época é uma iteração, composta por vários ciclos, onde cada ciclo tem um tamanho *batch*. O otimizador é um algoritmo que ajuda a calcular novos pesos. Um dos mais usados é o Adam [67] devido à sua eficiência e facilidade de configuração. Durante o processo de otimização, o método do erro é utilizado para calcular o erro no final de cada ciclo ou iteração, o qual é então utilizado pelos otimizadores para ajustar os pesos do modelo.

7.4 Resultados obtidos nos métodos

Para a criação de um modelo de classificação ou um outro tipo de método, serão necessários dados e parâmetros. Esses parâmetros permitiram ajustar o modelo, provocando variações nos resultados, influenciando quer em tempo de treino, quer no valor da taxa de acerto. Por isso, para esta fase, iremos fazer várias avaliações, tanto aos parâmetros de entrada, como aos dados e aos métodos utilizados.

7.4.1 Detetar planta

Para este tipo de classificação, onde o objetivo é determinar se uma imagem contém algum tipo de planta doméstica, iremos usar uma arquitetura CNN, nomeadamente, a *MobileNetV2*. Para esta arquitetura iremos descartar a parte do classificador e incluir uma nova, usando a técnica de *transfer learning*. E iremos iniciar os pesos com os pesos da *Imagenet*. Como classificador será usado uma rede neuronal, com 16 entradas e 1 saída. A entrada do modelo terá um tamanho de 224 por 224, sendo que as imagens terão de ser redimensionadas caso não o tenham. Sendo uma classificação binária, terá inicialmente uma saída. Na Figura 7.2, podemos ver o resumo do modelo que servirá como primeira experiência para este tipo de classificação.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_224 (Functi (None, 7, 7, 1280)        2257984
_________________________________________________________________
flatten (Flatten)            (None, 62720)             0
_________________________________________________________________
dense (Dense)                (None, 16)                1003536
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 17
=================================================================
Total params: 3,261,537
Trainable params: 1,003,553
Non-trainable params: 2,257,984
_________________________________________________________________
```

Figura 7.2: Resumo do modelo para a classificação de planta

Podemos ver que, na Figura 7.2, o classificador possui três camadas, a camada de entrada (*flatten*), uma camada escondida (*dense*) e uma camada de saída (*dense_1*). Na camada escondida é usado uma função de ativação *ReLu* e na camada de saída uma *Sigmoid*.

No treino, foi utilizado o otimizador Adam e a métrica de avaliação para calcular o erro foi o MSE (Mean Squared Error), que é o critério preferido para problemas de regressão. Foi ainda usado para o tamanho do *batch*, 64 e para o número de épocas, 10.

Foi obtido uma taxa de acertos 98.5% no teste e 99.9% no treino. E um erro final no teste de 0.045. Para uma melhor visualização do treino do modelo, temos a seguinte Figura 7.3, que nos mostra a taxa de acertos e o erro ao longo das épocas.

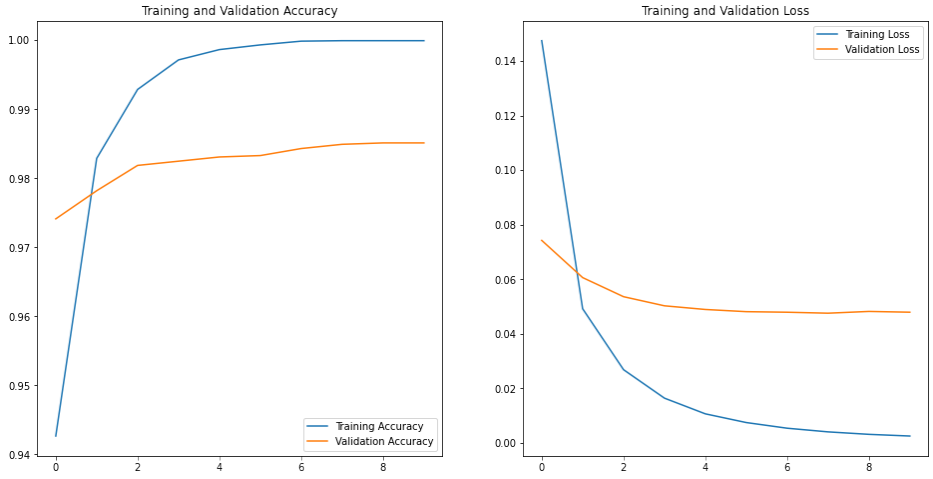


Figura 7.3: Evolução do modelo para a classificação de planta

Tanto a taxa de acertos como o erro vão diminuindo ao longo das épocas. E o facto de parecer que ambos dos gráficos de treino e de teste ficaram separados é porque o treino começou com uma elevada taxa de acertos. Para este teste, obtivemos num total de 4908 dados de teste apenas 73 errados.

Após treinar um modelo, é preciso usar certas métricas para avaliar a qualidade do modelo. O problema é, sendo uma classificação binária com uma função de *sigmoid* à saída, os valores que saem no final são valores que estão entre [0,1]. Ou seja, é necessário ter um limiar, para decidir qual das classes será a escolhida. Neste classificador foi usado um limiar de 0.5, onde se obteve um melhor resultado com um total de 73 errados, um *recall* de 0.985, *precision* de 0.99 e *f-score* de 0.99.

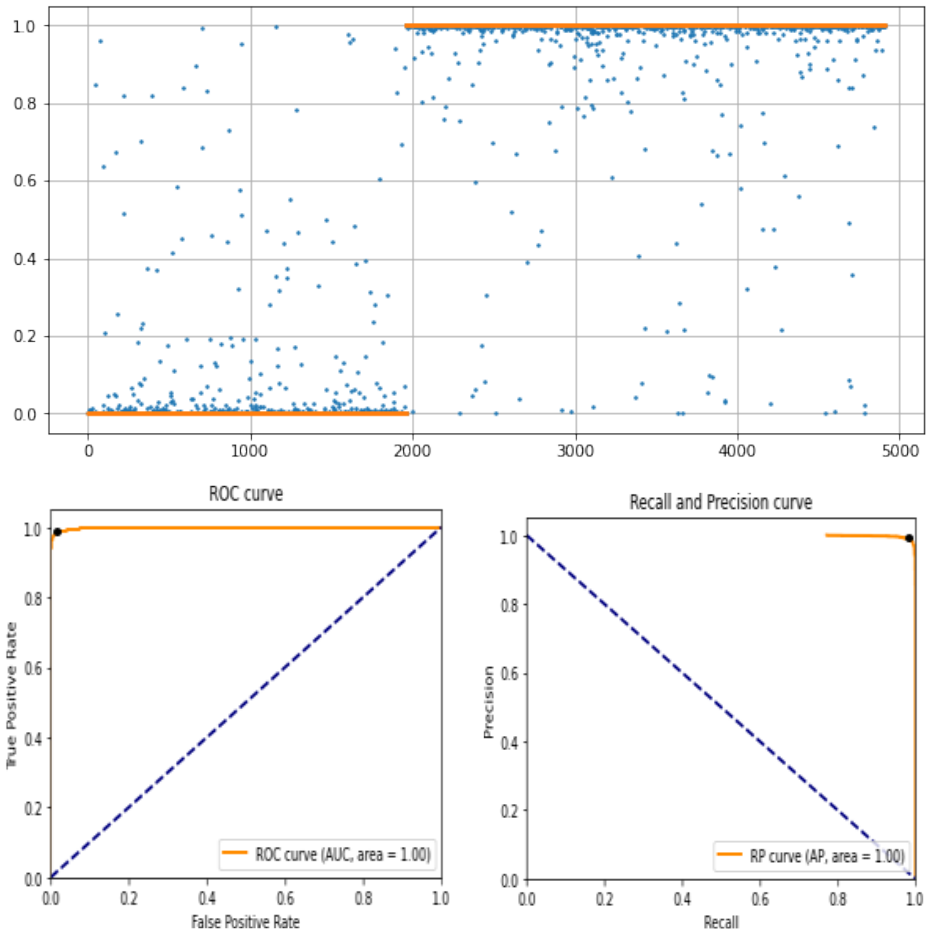


Figura 7.4: Curva de ROC e PR do modelo para a classificação de planta

Na primeira imagem da Figura 7.4, podemos ver o comportamento da

função *sigmoid*, onde os pontos azuis são os valores preditos e os laranjas os valores reais. Na segunda imagem, é apresentado um gráfico da curva *ROC*. A área abaixo da curva *ROC*, que varia entre 0 e 1, é uma medida do desempenho do modelo, sendo que valores mais próximos de 1 indicam uma maior capacidade de discriminação entre as classes. Nesse caso, a área da curva *ROC* é igual a 1. Na terceira imagem, é apresentada a curva *RP* que relaciona a precisão com o *recall* do modelo. A área abaixo da curva *RP*, que também varia entre 0 e 1, é uma medida da qualidade da classificação realizada, sendo que valores mais próximos de 1 indicam uma melhor capacidade do modelo em encontrar exemplos positivos enquanto minimiza os falsos positivos. Nesse caso, a área da curva *RP* também é igual a 1.

Foi usado o conjunto de dados *256_ObjectCategories* para testar o modelo com outras novas imagens. Com este dados obtivemos uma taxa de acerto de 95% e uma taxa de erro de 0.498.

A taxa de falsos negativos foi de 4,7%, indicando que 4,7% das amostras que pertenciam à classe negativa foram incorretamente classificadas como positivas. É possível perceber que houve um número considerável de falsos positivos. Isso pode ser atribuído a imagens que continham uma planta, mesmo pertencendo a outra classe, o que acabou afetando a classificação correta dessas amostras. Na Figura 7.5 podemos ver alguns desses resultados.

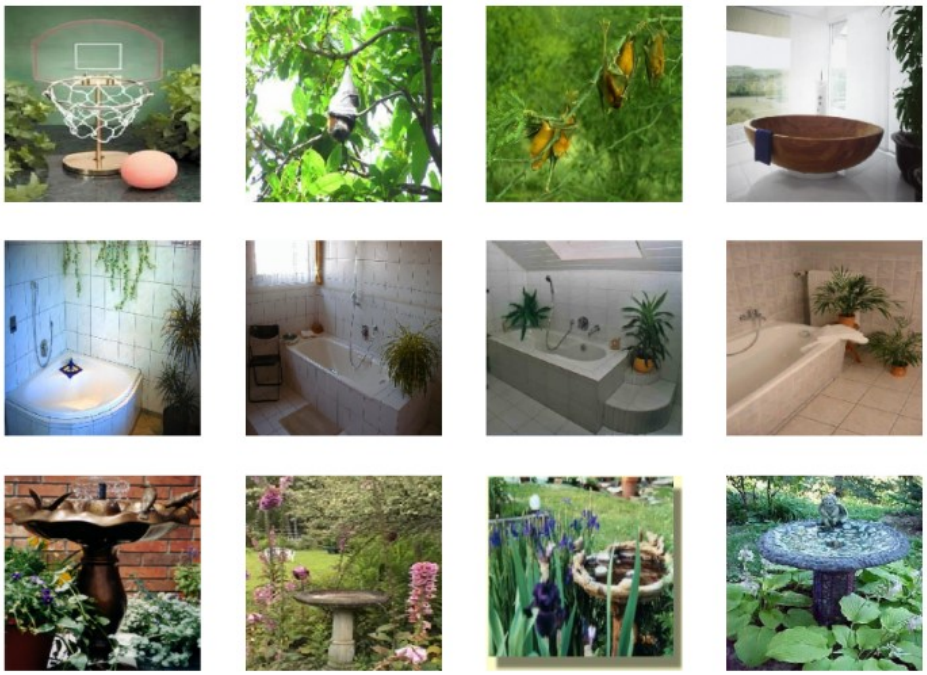


Figura 7.5: Imagens de teste do conjunto de dados 256_ObjectCategories

Com isto, podemos dizer que, para este classificador obtivemos um bom modelo para detetar se uma imagem contém ou não uma planta doméstica.

7.4.2 Detetar planta saudável

Para este tipo de classificação, onde o objetivo é determinar se a planta/-folha detetada é saudável ou não, foram feitas 7 experiências. Nas primeiras 4, foram utilizadas arquiteturas feitas de raiz e nas outras 3 foram utilizadas arquiteturas que usam a técnica de *transfer learning*, usando a arquitetura *MobileNetV2*. Na Tabela 7.2 podemos ver os resultados obtidos com as 4 primeiras arquiteturas.

Arquitetura	normalização	padding	data augmentation	Accuracy
1	não	não	não	0.717
	sim	não	não	0.762
	sim	sim	não	0.804
	sim	sim	sim	0.808
2	não	não	não	0.705
	sim	não	não	0.766
	sim	sim	não	0.796
3	não	não	não	0.676
	sim	não	não	0.747
	sim	sim	não	0.794
3	não	não	não	0.686
	sim	não	não	0.721
	sim	sim	não	0.806
	sim	sim	sim	0.803

Tabela 7.2: Quatro arquiteturas feitas de raiz para detetar se a planta é saudável.

Na Tabela 7.2, podemos ver as arquiteturas que foram feitas de raiz. Foram feitos 4 testes diferentes para cada arquitetura. No primeiro, apenas foi feito um redimensionamento aos dados de entrada. No segundo, foi feito uma normalização. No terceiro, foi feito a normalização e ainda foi feito um *padding* aos dados. O *padding*, é um tipo de redimensionamento dos dados, onde a imagem a redimensionar não será desformatada, acrescentado bordas pretas (pixeis pretos) de lado. E por fim, no quarto teste, foi usada a técnica de *data augmentation*.

Na primeira destas 4 arquiteturas são utilizadas 6 camadas. A primeira

camada tem uma entrada de 256 por 256 e a última camada contém apenas 1 nó. Na Figura 7.6, podemos ver com mais detalhe essa arquitetura. O melhor resultado para esta experiência foi obtido com normalização, *padding* e *data augmentation*. A taxa de acertos obtida foi de 80.4%.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 254, 254, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 127, 127, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 125, 125, 16)      2320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 62, 62, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 60, 60, 16)        2320
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 16)        2320
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 16)        0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense (Dense)                (None, 16)                50192
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 17
=================================================================
Total params: 57,617
Trainable params: 57,617
Non-trainable params: 0
_________________________________________________________________
```

Figura 7.6: Primeira arquitetura para o problema de deteção de planta saudável.

Na segunda experiência, com uma mudança na arquitetura, foram realizadas apenas 3 testes diferentes. Dado que a taxa de acertos nesta experiência baixa, foi decidido não utilizar *data augmentation* porque não iria aumentar a taxa de acerto para valores acima da experiência anterior. A alteração que foi feita da arquitetura 1 para a arquitetura 2 foi mudar na camada *dense* o número de nós, de 16 para 64. Com isto, foi obtida uma taxa de acertos de 79.6%.

Na terceira, foi mudado o número de nós da camada *dense* de 64 para 128. Com isto, obtivemos uma taxa de acertos de 79.4%.

Na quarta experiência e última das arquiteturas testadas, foi retirado a camada *conv2d_3* com o *max pooling* respetivo (*max_pooling2d_3*) à arquitetura

representada na Figura 7.6, ficando apenas com 3 camadas convolucionais. Ainda foi alterado o número de nós em todas as camadas, passando de 16 para 32. Mas apenas obtivemos uma taxa de acertos de 80.3%, pelo que foi mais baixo que a primeira experiência.

Nestas primeiras 4 experiências das arquiteturas construídas, a que teve melhor taxa de acertos foi a primeira arquitetura, com normalização, *padding* e *data augmentation*.

Para as arquiteturas que usam *transfer learning*, usámos a arquitetura MobileNetV2. Foram criadas 3 arquiteturas diferentes. Em cada uma foi usado a técnica *fine-tuning*, que tem como objetivo congelar camadas, isto é, essas camadas congeladas ao treinar, os pesos não se alteram. Assim, haverá menos processamento a nível computacional. E já que o objetivo é usar a arquitetura *MobileNetV2* como base e não criar uma nova, então não precisaremos de re-treinar todas as camadas. A entrada das redes convolucionais será de 224 por 224 e foi usado *data augmentation*.

Na primeira arquitetura, usamos como classificador uma rede neuronal, com apenas uma camada densa de 32 nós. A camada densa tem uma função do tipo *ReLu* e a camada de saída *sigmoid*.

Na segunda foi adicionado mais uma camada densa com 32 nós e foi ainda adicionado dois *dropouts* com uma probabilidade de 0.2. As duas camadas densas têm uma função de ativação do tipo *ReLu* e a camada de saída do tipo *sigmoid*.

E por fim, a terceira arquitetura, com apenas uma camada densa de 1024 nós. Foi ainda adicionado uma *pooling* de média global na entrada da rede. As camadas têm as mesmas funções de ativações.

Podemos ver com mais detalhe a composição das arquiteturas usadas na Figura 7.7 e o seu comportamento ao longo das épocas. As redes foram treinadas para 30 épocas com um tamanho do *batch* de 64, com o otimizador Adam e a métrica de avaliação usada para o erro foi o *MSE*.

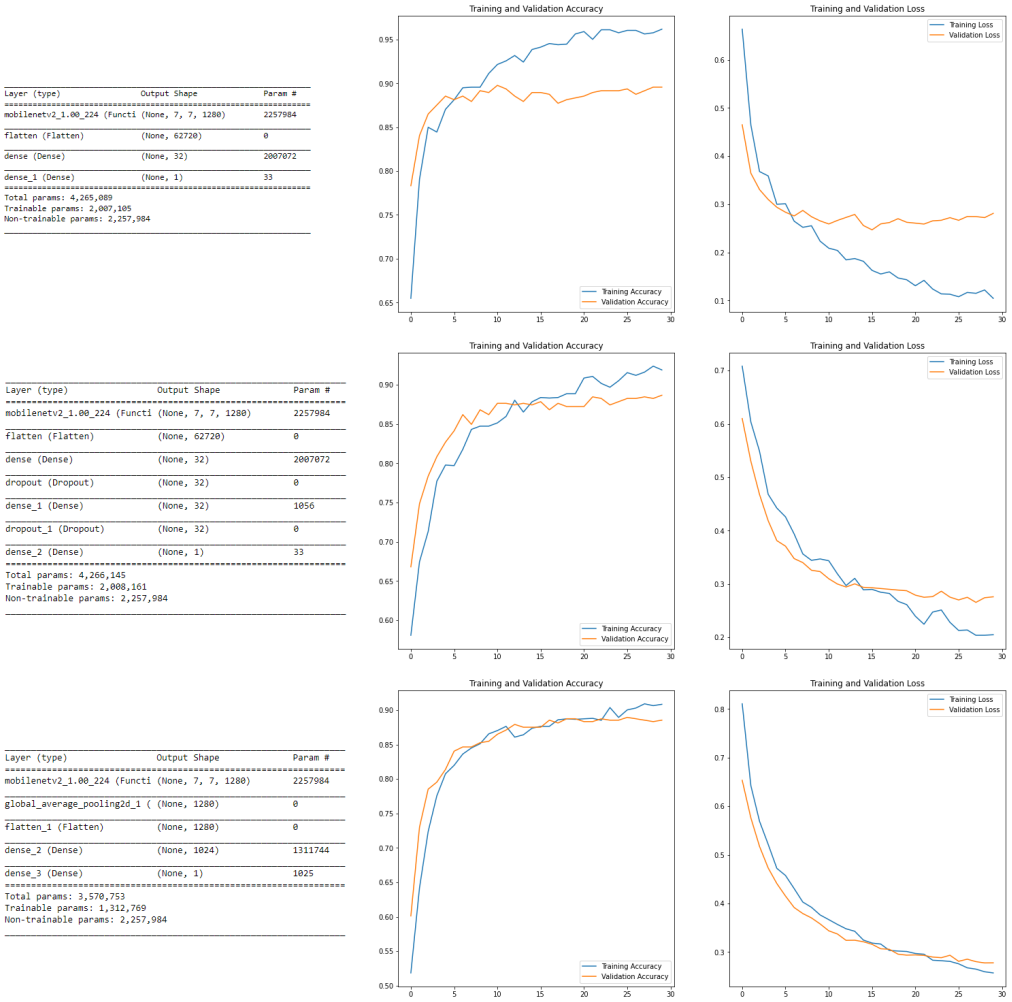


Figura 7.7: Evolução das arquiteturas com *transfer learning* para o problema de determinar se a planta está saudável

Na Figura 7.7, podemos ver que a terceira rede foi a que comportou-se melhor no que se refere à diferença das taxas de acertos e dos erros calculados ao longo das épocas nos dados de treino e de teste, ou seja, as curvas de treino e de teste desta arquitetura são as que mais se assemelham. Na primeira arquitetura temos como taxa de acerto uma percentagem de 89.6%, na segunda de 88.6% e na terceira 88.5%. Apesar de a terceira ter uma perceção de melhores resultados visto no gráfico, teve uma taxa de acertos menor que a primeira.

Como vimos, os melhores resultados foram obtidos com as arquiteturas

de *transfer learning*. Por isso, foi feito um outro teste com cada umas dessas arquiteturas com o conjunto de dados de várias imagens de plantas com e sem doenças tiradas com o telemóvel, que envolvia um conjunto de 300 imagens. Para a primeira arquitetura obtivemos uma taxa de acertos de 92%, na segunda de 91.6% e na terceira 94.6%. Assim sendo, para este classificador foi escolhido a terceira arquitetura. Observou-se que ao incluir imagens capturadas por dispositivos móveis no conjunto de dados os resultados melhoraram.

7.4.3 Classificar planta

Para este problema, temos como objetivo identificar qual o tipo de planta que está presente numa imagem. No treino foi usada a arquitetura *MobileNetV2* e o conjunto de dados das plantas. É de realçar que o conjunto de dados contém 10 classes de plantas com mais 1 classe extra. Essa classe extra será para classificar todas as outras plantas que não estejam dentro das características das 10 classes de plantas. Tendo mais de 2 classes por classificar, este problema é abordado com um classificador multi-classe.

A arquitetura usada terá um classificador de uma camada densa de 256 nós e uma função de ativação *ReLu*. A camada de saída terá uma função de ativação *softmax* com 11 nós no total. Ainda será usado um *dropout* de probabilidade 0.25. Será inicializado com os pesos da *Imagenet* e terá como entrada um tamanho de 224 por 224. É usado também a técnica de *fine-tuning* para treinar parte das camadas convolucionais.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_224 (Functi (None, 7, 7, 1280)        2257984
_________________________________________________________________
flatten_2 (Flatten)          (None, 62720)             0
_________________________________________________________________
dense_4 (Dense)              (None, 256)               16056576
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_5 (Dense)              (None, 11)                2827
=================================================================
Total params: 18,317,387
Trainable params: 16,059,403
Non-trainable params: 2,257,984
_________________________________________________________________
```

Figura 7.8: Resumo do modelo para o problema de classificar a planta

Para o treino, foi usado o otimizador Adam e a métrica de avaliação usada para o erro foi o *MSE*. O número de épocas foi 20 e o tamanho do *batch* de

64. Em baixo podemos ver a evolução da rede ao ser treinada ao longo das épocas.

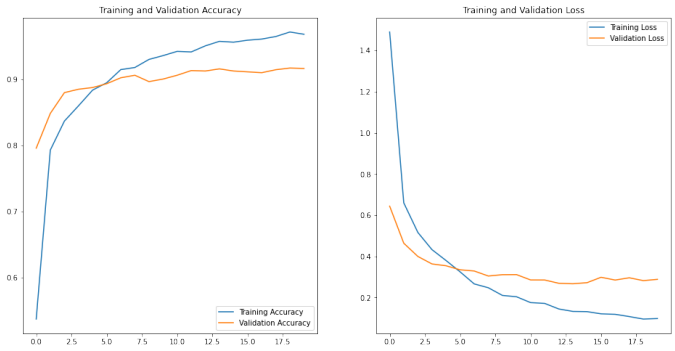


Figura 7.9: Evolução do modelo para o problema de determinar se o tipo de planta

Como podemos ver a rede estabilizou por volta dos 94% de taxa de acertos e de 0.28 de erro nos dados de teste. Em baixo podemos ver a matriz de confusão com os dados de teste.

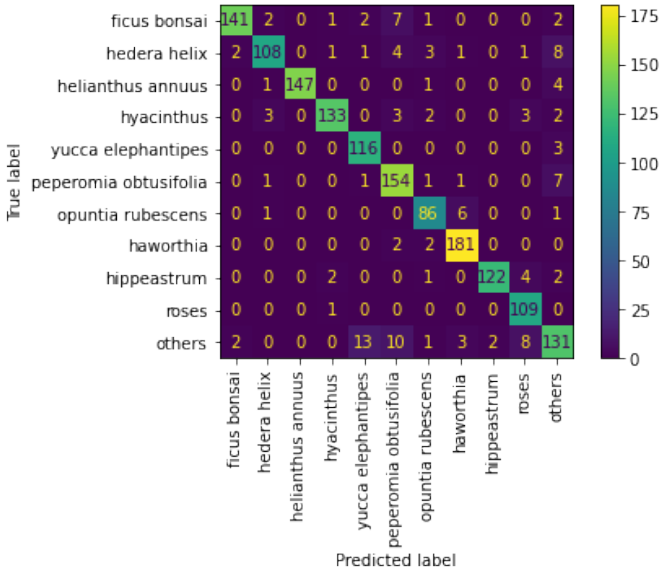


Figura 7.10: Matriz de confusão para o problema de classificar a planta

Podemos ver que na matriz de confusão temos duas classes (*yucca elephantipes* e *opuntia rubescens*) que se confundem com a classe *others*, isto deve-se

ao facto de haver poucos dados. Os restantes resultados obtidos estão dentro do esperado para que o classificador possa ser utilizado na aplicação sem prejudicar a experiência do utilizador.

Como segundo teste, foram tiradas 50 fotografias com um telemóvel de acordo com as classes e testou-se essas imagens com a rede treinada (Figura 7.11).

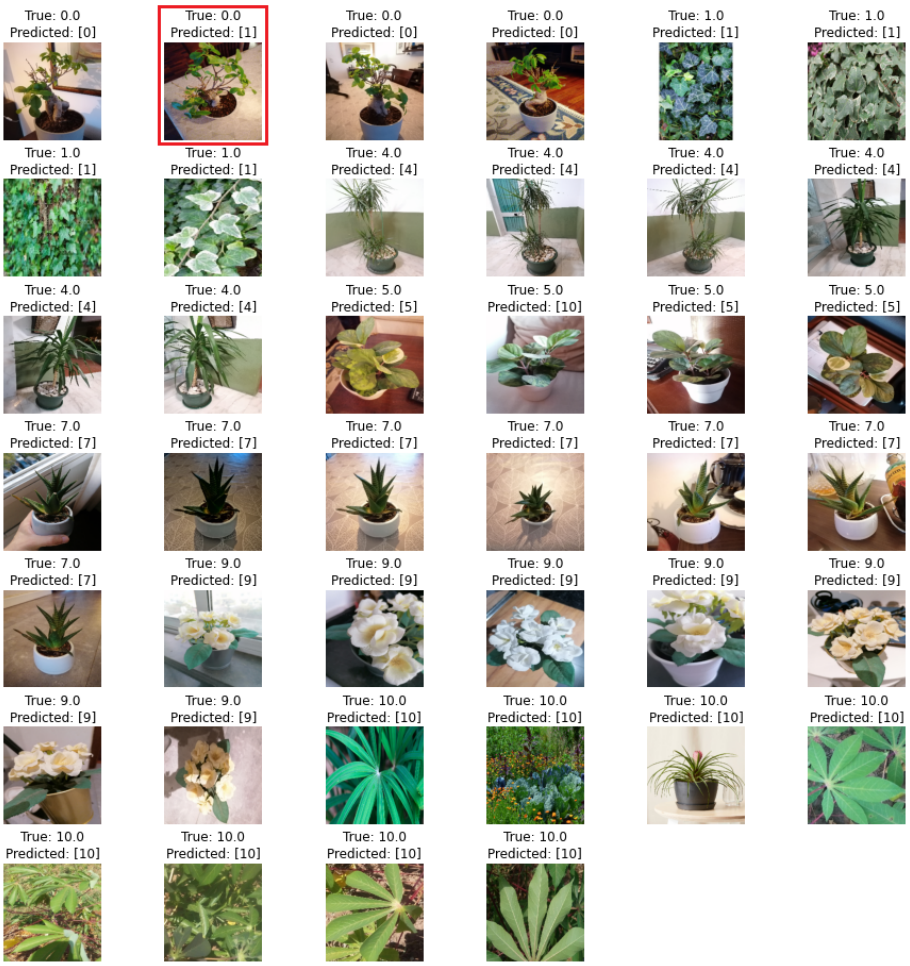


Figura 7.11: Teste com fotografias tiradas com o telemóvel para o problema de determinar se o tipo de planta

Na Figura 7.11, das imagens que são mostradas, apenas uma está errada. Por isso, podemos dizer que a rede está treinada de forma adequada para os dados que temos.

7.4.4 Detetar e classificar doença

Para este bloco, o objetivo é classificar qual o tipo de doença da planta detetada. Para isso, foi usado o conjunto de dados com 5 doenças para o treino de um classificador. Para detetar a doença é necessário extrair bem as caraterísticas das imagens a classificar. Sendo que pode haver doenças muito semelhantes é necessário escolher a arquitetura adequada.

Com o estudo que foi feito sobre as arquiteturas 2.3.1, vimos que as mais usadas para problemas semelhantes eram a *MobileNet* e a *ResNet*. Existem vários variantes provenientes à *ResNet*, tal como a *ResNet50/V2*, *ResNet101/V2*, *ResNet152/V2*, *InceptionResNetV2*, entre outras. Para isso, foram escolhidas duas redes pré-treinadas, nomeadamente, a *MobileNetV2* e *InceptionResNetV2*.

As redes usam data augmentation e são inicializados com os pesos da *ImageNet*. Não será incluído a parte do classificador, já que será gerado e retreinado com um classificador diferente para o problema em questão usando a técnica de *transfer learning*. Na Figura 7.12, podemos ver a arquitetura do classificador usada nos modelos.

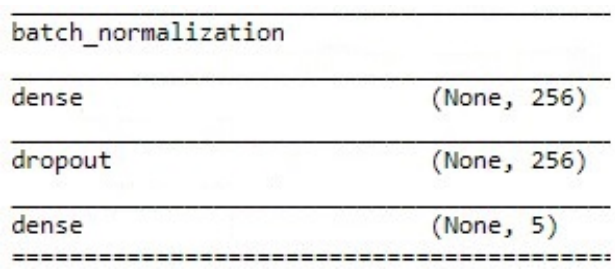


Figura 7.12: Resumo da parte do classificador dos modelos

Para este estudo, foram utilizadas duas arquiteturas para o classificador dos modelos. Em ambas as arquiteturas, é aplicada uma técnica denominada *batch normalization*, bem como um *dropout* após a primeira camada escondida, com o objetivo de melhorar a generalização do modelo, juntamente com regularizadores para o *kernel* e o *bias*. Conforme é apresentado na Figura 7.12.

Na primeira arquitetura, é usada uma camada escondida de 32 nós com uma função *ReLu* e uma camada de saída com uma função de ativação *softmax*. Na segunda arquitetura, a camada escondida possui um total de 256

nós. Na Tabela 7.3 são apresentados os valores usados no *batch normalization* e *dropout*.

Batch normalization	
momentum	epsilon
0.99	0.001

Dropout (probabilidade)	
arquitetura 1	arquitetura 2
0.2	0.45

Tabela 7.3: Valores usados no *batch normalization* e *dropout*

Para o treino, testou-se com dois otimizadores, o Adam e o Adamax. Foi usado um *callback* que reduzisse a taxa de aprendizagem quando a métrica a monitorizar (neste caso o erro) parasse de melhorar. Ou seja, reduzir a taxa de aprendizagem a partir de um fator quando nenhuma melhoria for observada no erro para um número de épocas de *patience*. O número de épocas usado para treinar o modelo foi de 15 com 10 *steps* por época. Podemos observar na Tabela 7.4, os valores usados nos otimizadores e callback.

Otimizador		
Adam		Adamax
0.001	0.00001	0.001

Callback		
fator	patience	mínima taxa de aprendizagem
0.3	3	0.000001

Tabela 7.4: Valores para os parâmetros de treino

Na primeira arquitetura, com 32 nós, as redes não apresentaram evolução na aprendizagem em ambos os otimizadores. Na segunda arquitetura, durante o teste com o otimizador Adam, os modelos não demonstraram melhorias significativas na aprendizagem para taxas de acerto superiores a 70%, tanto para a *MobileNetV2* quanto para a *InceptionResNetV2*. Entretanto, utilizando o otimizador *Adamax*, conseguimos atingir uma taxa de acerto de apenas 86% com a *MobileNetV2*, mas obtivemos uma taxa de acerto de 92.9% com a *InceptionResNetV2*. Na Figura 7.13, é possível observar a evolução do modelo utilizando a *InceptionResNetV2*.

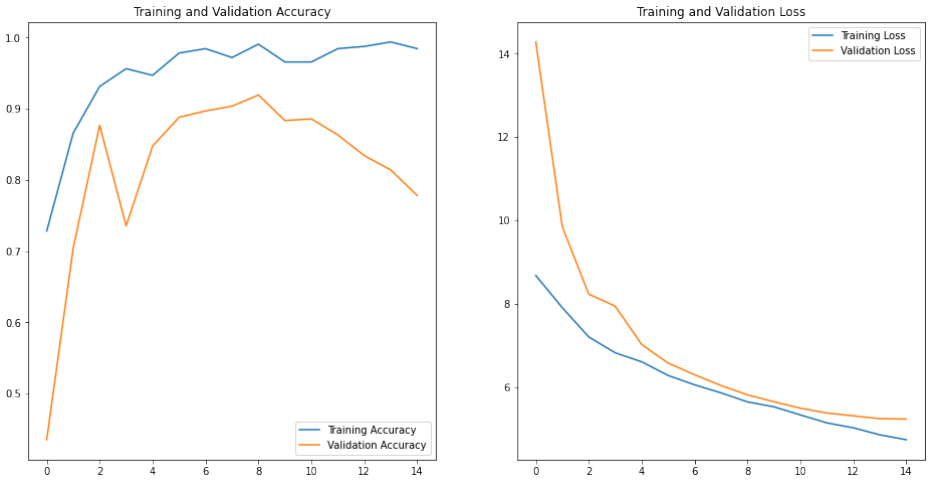


Figura 7.13: Evolução do modelo InceptionResNetV2 para o problema de determinar se o tipo de doença

Podemos ver que a probabilidade de acertos do modelo começa a decair quando chega aos 93%. Por isso, é usado um outro *callback* para guardar o melhor modelo de acordo com a taxa de acertos nos dados de teste, com isto obtivemos 92.9% na taxa de acertos. Na Figura 7.15, podemos visualizar a sua matriz de confusão

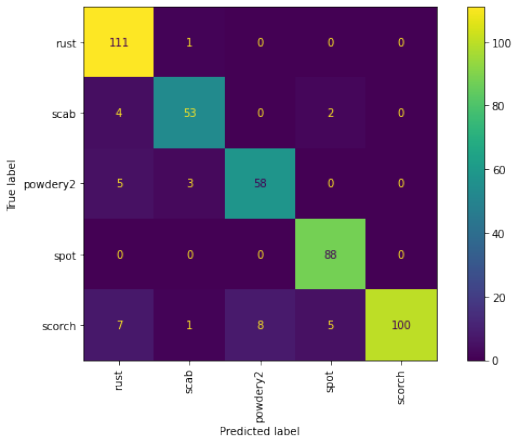


Figura 7.14: Matriz de confusão com o modelo *InceptionResNetV2* para o problema de determinar se o tipo de doença

Para outro teste, foram adquiridas várias fotografias com um telemóvel a várias plantas com doenças, mas como podemos perceber algumas das imagens foram classificadas mal por haver falta de dados. Algumas das imagens

contém doenças que contém manchas amarelas, e manchas amarelas são umas das descrições da doença *Rust*, só que em alguns casos nem sempre é assim. E pela falta de dados, a rede classifica como *Rust*.

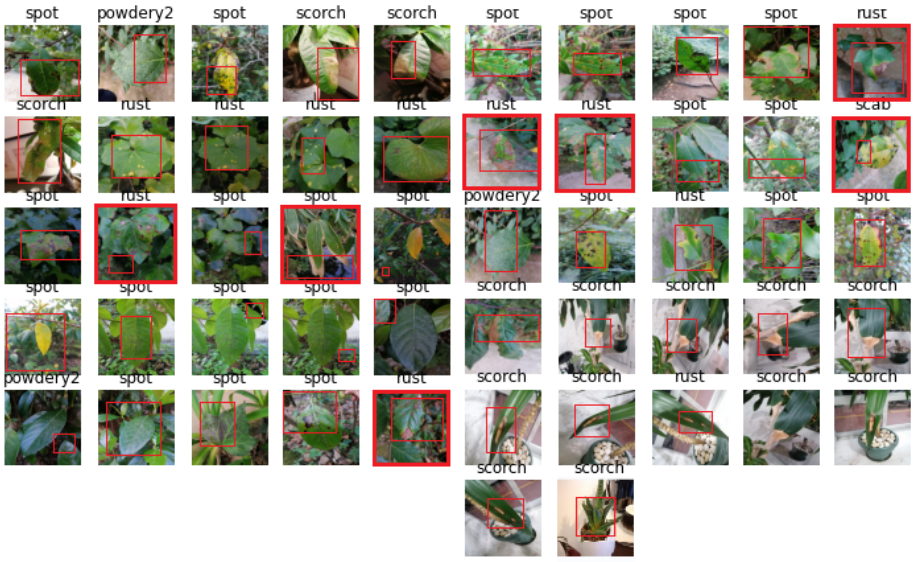


Figura 7.15: Teste com fotografias - determinar se o tipo de doença

Como o estudo das doenças é uma área vasta, seria necessário pessoas especializadas para a recolha de dados de plantas com doenças. E como os dados que foram obtidos a partir de várias fontes, alguns desses dados poderão não estar bem classificadas, podendo enganar a rede ao treinar. Tendo poucos dados e imagens possivelmente mal classificadas para treinar o modelo, os resultados serão muito diferentes ao classificar imagens no real. Na Figura 7.15, é possível observar que as imagens classificadas incorretamente apresentam bordas vermelhas ao redor das bordas. Além disso, pode-se notar a presença de outras bordas menos intensas dentro das imagens, que são geradas pelo algoritmo *Grad-CAM*. É importante salientar que algumas dessas bordas geradas pelo algoritmo nem sempre indicam corretamente a doença, devidamente à falta de dados.

7.4.5 Deteção de imagens escuras

Para ajudar o utilizador e as redes, foi adicionado um bloco para detetar se as imagens estão escuras. Caso estivessem escuras era pedido ao utilizador para tirar uma fotografia nova. Ao treinar as redes, é usado *data augmentation* que altera a luminosidade das imagens ao treinar. Por isso, aqui iremos

descartar e alertar o utilizador das imagens que estão muito escuras para as redes. Para isso, foram adquiridas 81 imagens (telemóvel e *google*) para detetar se a imagem tem baixa luminosidade usando o método que foi proposto na Secção 4.5.1 .

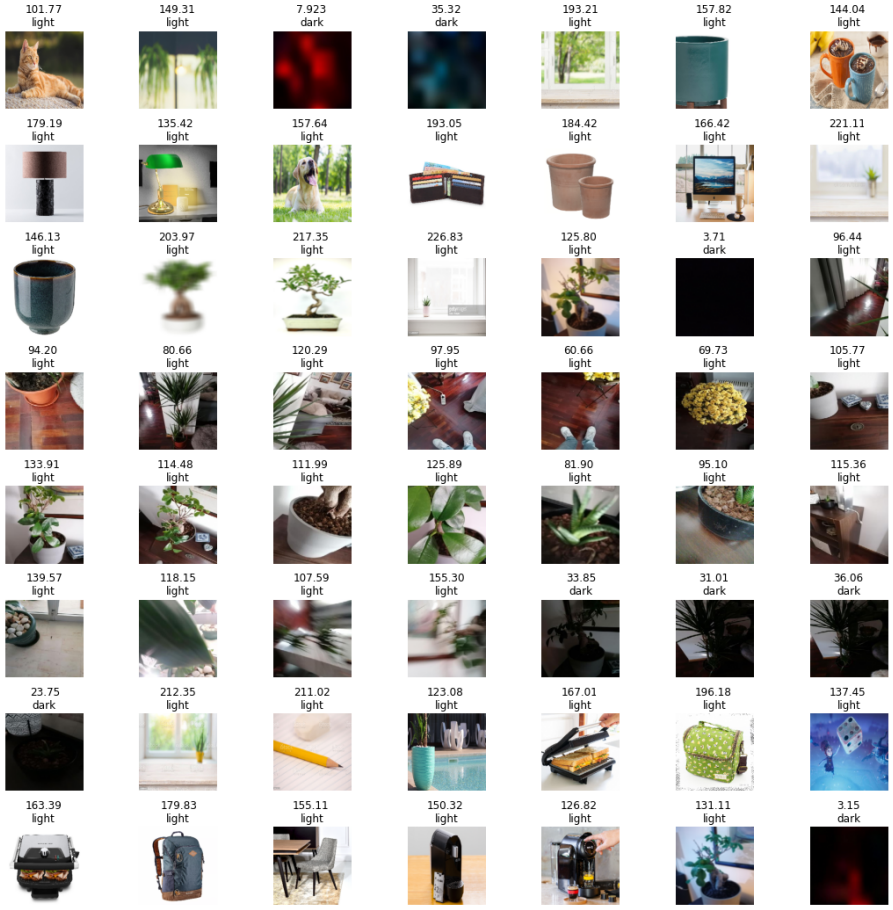


Figura 7.16: Teste com fotografias e imagens da Google para o problema de deteção de pouca luminosidade

Podemos ver na Figura 7.16, que as imagens apresentadas têm um número em cima. Esse número é a média dos pixeis de cada imagem. E com esse número é possível perceber se a imagem tem baixa luminosidade. Para isso, é necessário ter um limiar. Para este teste foi usado um limiar de 50 e como podemos ver não houve nenhum erro.

7.4.6 Deteção de imagens desfocadas

Com o mesmo objetivo de ajudar o utilizador e as redes foi proposto a deteção de imagens desfocadas. Para este classificador foram testados 2 métodos (Secção 4.5.3), a *FFT* e a variância de Laplace. Para estes testes, usou-se o conjunto de dados *CERTH*.

Com método *FFT*, obteu-se uma taxa de acertos de 77.30% e com método da variância de Laplace obteu-se 84.39%. Em baixo podemos ver um exemplo de uma imagem tirada com um telemóvel.

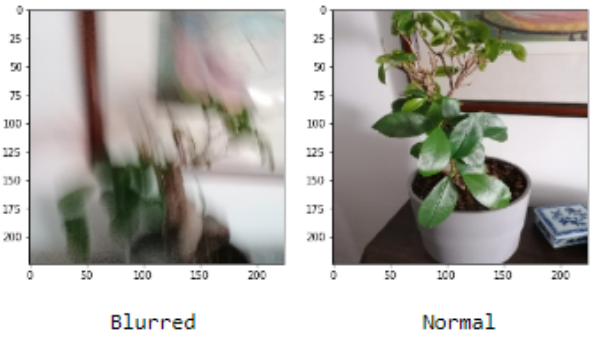


Figura 7.17: Deteção de desfoque

Em baixo, na Tabela 7.5, temos um resumo dos métodos que foram escolhidos com as respetivas taxas de acertos.

Detetar planta	MobileNetV2	98.5%
Detetar planta saudável	MobileNetV2	94.6%
Classificar planta	MobileNetV2	94%
Detetar e classificar doença	InceptionResNetV2	92.9%
Deteção de imagens escuras	Conversão para HSV, média e limiar	99%
Deteção de imagens desfocadas	Variância de Laplace	84.39%

Tabela 7.5: Resultados dos métodos escolhidos

Foi possível concluir que pelas experiências feitas aos métodos, os resultados são adequados para que a experiência do utilizador na aplicação não seja alterada.

7.5 Resultados obtidos na aplicação

Com os métodos testados o próximo passo é testar esses mesmos métodos com a aplicação e o servidor a funcionar. Para isso, foram adquiridas várias fotografias a vários objetos, plantas, folhas, etc. E foi avaliado o seu tempo de processamento de acordo com o que foi visto nas Secções 3.2 e 3.3. Na Figura 7.18 podemos ver os vários testes que fizemos usando a aplicação, juntamente com o servidor *AI* e a base de dados.

Deteção de imagens escuras
Deteção de imagens desfocadas
Deteção de plantas

predito	real	tempo
dark	dark	28 ms
dark	dark	23 ms
dark	dark	18 ms
dark	dark	24 ms
blurred	blurred	18 ms
blurred	blurred	22 ms
blurred	blurred	34 ms
random	random	788 ms
random	random	658 ms
random	random	598 ms
random	random	549 ms
plant	random	762 ms
random	random	725 ms
random	random	615 ms
random	random	505 ms
random	random	724 ms
random	random	745 ms
dark	dark	20 ms
plant	random	581 ms
plant	random	624 ms
blurred	blurred	19 ms
plant	random	599 ms
plant	random	638 ms

Deteção do tipo de doença*

predito	real	tempo
rust	rust	3.80 s
rust	rust	3.63 s
powdery	powdery	4.22 s
healthy	healthy	1210 ms
powdery	powdery	3.89 s
healthy	healthy	1109 ms
healthy	healthy	1103 ms
scorch	scorch	3.85 s
scorch	scorch	3.52 s
scorch	scorch	3.54 s
rust	rust	3.98 s
scorch	scorch	3.47 s
scorch	scorch	4.36 s
spot	spot	4.37 s
spot	spot	3.79 s
spot	spot	3.53 s
healthy	healthy	939 ms
powdery	healthy	3.69 s
scorch	scorch	3.89 s
scab	scab	3.72 s
scab	scab	4.26 s

Deteção do tipo de planta*

predito	real	tempo
NDPLANT/healthy	NDPLANT/healthy	2.40 s
NDPLANT/healthy	HEDERA/healthy	1647 ms
NDPLANT/healthy	NDPLANT/healthy	1687 ms
haworthia/unhealthy	haworthia/unhealthy	1631 ms
haworthia/unhealthy	haworthia/unhealthy	1631 ms
NDPLANT/healthy	NDPLANT/healthy	1767 ms
NDPLANT/healthy	NDPLANT/healthy	1578 ms
hyacinthus/unhealthy	NDPLANT/healthy	1725 ms
NDPLANT/healthy	NDPLANT/healthy	1703 ms
NDPLANT/healthy	NDPLANT/healthy	1435 ms
NDPLANT/unhealthy	rose/	1636 ms
roses/unhealthy	roses/unhealthy	1517 ms
roses/unhealthy	roses/unhealthy	1565 ms
roses/unhealthy	roses/unhealthy	1508 ms
NDPLANT/unhealthy	haworthia/unhealthy	1669 ms
peperomia/unhealthy	peperomia/unhealthy	1845 ms
peperomia/unhealthy	peperomia/unhealthy	1722 ms
peperomia/unhealthy	peperomia/unhealthy	1784 ms
bonsai/healthy	bonsai/healthy	1781 ms
yucca/unhealthy	yucca/unhealthy	1675 ms
bonsai/unhealthy	bonsai/unhealthy	1571 ms
bonsai/healthy	bonsai/healthy	1851 ms
peperomia/healthy	bonsai/healthy	1573 ms
yucca/unhealthy	yucca/unhealthy	1565 ms
yucca/unhealthy	yucca/unhealthy	1763 ms

Legenda
classificação errada

*utiliza também os métodos da primeira tabela

Figura 7.18: Teste à aplicação e servidor AI

Podemos ver que os classificadores, em geral, são relativamente rápidos a disponibilizar informação para ser apresentada na aplicação. Os testes estão divididos em três partes. Numa primeira parte, temos os testes das deteções das imagens escuras, desfocadas ou se a imagem contém uma planta ou não. No segundo teste, temos a deteção do tipo de doença. E terceiro, a deteção do tipo de planta. Para o segundo e terceiro teste é necessário realizar os métodos do primeiro teste, juntamente com a deteção se a planta é saudável. Através destes resultados, podemos perceber que é possível obter uma classificação de uma planta ou doença em menos de 5 segundos.

Capítulo 8

Conclusões

O desenvolvimento deste trabalho permitiu contribuir para a área das plantas e doenças, usando técnicas de *Deep Learning* para o reconhecimento de imagens, nomeadamente, modelos *CNN*. Foi ainda feito um estudo de vários métodos de pré-processamento que pudessem melhorar no desempenho dos modelos, tais como, a deteção e melhoria de imagens de baixa luminosidade, imagens desfocadas, *data augmentation*, normalização e redimensionamento.

Através das experiências e dos resultados dos vários métodos, foi possível criar uma aplicação, capaz de detetar e classificar plantas e doenças. Onde o utilizador, ao usar a câmara da aplicação *HousePlants*, conseguia classificar as plantas domésticas e detetar e classificar a doença presente, apresentado depois, uma séria de informações da planta ou doença. Foi possível concluir que pelas experiências feitas aos métodos, os resultados foram superiores ao esperado, mesmo tendo conjuntos de dados pequenos, quer isto dizer, pouco enriquecidos em termos de informação. Foi possível obter as seguintes taxas de acertos nos métodos estudados:

Detetar planta	MobileNetV2	98.5%
Detetar planta saudável	MobileNetV2	94.6%
Classificar planta	MobileNetV2	94%
Detetar e classificar doença	InceptionResNetV2	92.9%
Deteção de imagens escuras	Conversão para HSV, média e limiar	99%
Deteção de imagens desfocadas	Variância de Laplace	84.39%

Tabela 8.1: Resultados dos métodos utilizados para a implementação da aplicação HousePlants

Os quatro primeiros métodos, são métodos que usam modelos CNN, os outros dois são métodos mais convencionais, usados antes dos modelos. As experiências feitas à aplicação, ou seja, testar a aplicação juntamente com os métodos, foi possível obter uma classificação de uma planta ou doença em menos de 5 segundos.

A aplicação móvel é constituída também por uma *API* e um servidor *Firebase*. A *API*, é o servidor que contém toda a parte da aprendizagem automática, no qual ajudará a aplicação a fazer o reconhecimento das imagens. Já o servidor *Firebase*, servidor disponibilizado pela *Google*, ajudará na segurança do registo de utilizadores na aplicação e será o repositório onde será guardado as informações necessárias a todas as plantas e doenças que serão apresentadas na aplicação.

Contudo, para a elaboração deste projeto tivemos algumas limitações, no que diz respeito aos conjuntos de dados e ao hardware disponibilizados. Os conjuntos de dados que recolhemos de várias fontes não foram suficientes, e muitos desses dados não foram tratados por especialistas, o que significa que muitos dos dados podem estar incorretamente etiquetados. Alguns desses dados foram usados para testes, mas quando foram testados em condições reais, os resultados foram diferentes, dificultando nos testes dos métodos utilizados.

O hardware utilizado também não ajudou, o que levou a fazer vários testes e onde consumiu imenso tempo, mesmo com o uso da *GPU*. E com o tempo despendido na maior parte nos estudos dos modelos, não foi possível fazer um estudo mais aprofundado no que concerne ao melhoramento de imagens.

No entanto, este trabalho pode ser alvo de futuros desenvolvimentos, como melhorar os métodos de pré-processamento estudados neste trabalho. Aumentar os conjuntos de dados, tornando-os mais ricos, é outro aspeto importante para o futuro, para que se possa obter melhores resultados, não só nos modelos a treinar como no real. Além disso, a segmentação de imagem pode ser explorada para melhorar a identificação da planta, permitindo extrair a região de interesse de forma mais precisa e eficiente. A integração de novas técnicas para a deteção e reconhecimento de imagens, ou a utilização de sistemas de deteção de imagem como *SSD* [68] ou *Yolo* [29], também são questões que deverão ser abordadas em trabalhos futuros. Outro aspeto a melhorar no futuro, consiste em ter novas funcionalidades que se demonstrem úteis aos utilizadores, tais como, secção de apoio, ter um *backoffice* no qual os gestores possam gerir a aplicação e incluir um site para que os utilizadores

possam visualizar as informações das plantas e doenças existentes. Finalmente, visto que a arquitetura do sistema é flexível, será necessário adicionar o servidor *AI* a um servidor público para que possa ser acedido publicamente.

Bibliografia

[1] C. Levy, "Why are houseplant prices so high," Available at https://anaturalcuriosity.org/why-are-houseplant-prices-so-high-in-2020/ (2020/09/03).

[2] C. Akin, "Why millennials are leading the houseplant trend," Disponivel em https://houseplantresourcecenter.com/2020/01/the-plant-crazy-generation-why-millennials-are-leading-the-houseplant-trend/ (2020/01/27).

[3] "Instead of houses, young people have houseplants," Disponivel em https://houseplantresourcecenter.com/2020/01/the-plant-crazy-generation-why-millennials-are-leading-the-houseplant-trend/ (2018/08/06), housePlant Resource Center.

[4] "Global indoor plants market," Disponivel em https://www.databridgemarketresearch.com/reports/global-indoor-plants-market, 08 2022, data Bridge Market Research.

[5] "Nonlinear function," Disponivel em https://www.cuemath.com/calculus/nonlinear-functions/, cuemath.

[6] A. Patterson, "Foundations of ml: Parameterized functions," Disponivel em https://towardsdatascience.com/foundations-of-ml-parameterized-functions-d2951a62272e (2019/07/27).

[7] M. Nielsen, data Science Academy. Deep Learning Book, 2022. Disponível em https://www.deeplearningbook.com.br/funcao-de-ativacao/.

[8] S. Polamuri, softmax Function Vs Sigmoid Function. Disponível em https://dataaspirant.com/difference-between-softmax-function-and-sigmoid-function/, (2017/03/07).

[9] P. Ratan, what is the Convolutional Neural Network Architecture?. Disponível em https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/, (2020/10/28).

[10] Deep Learning, Reconhecimento de Imagens. Disponível em https://lapix.ufsc.br/ensino/visao/visao-computacionaldeep-learning/deep-learningreconhecimento-de-imagens/.

[11] A. S. Swami Sivasubramanian, Peter DeSantis and A. Jassy, "Dive into deep learning, convolutional neural networks (lenet)," Disponivel em https://d2l.ai/chapter_convolutional-neural-networks/lenet.html, 2021.

[12] Y. LeCun, "Heroes of deep learning: Yann lecun," Disponivel em https://www.deeplearning.ai/blog/hodl-yann-lecun/.

[13] Vanishing gradient problem, wikipedia. Disponível em https://en.wikipedia.org/wiki/Vanishing_gradient_problem.

[14] Galoosh33, Google Inception model:why there is multiple softmax, stackExchange. Disponível em https://stats.stackexchange.com/questions/274286/google-inception-modelwhy-there-is-multiple-softmax.

[15] C. F. Wang, a Basic Introduction to Separable Convolutions. Disponível em https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728, (2018/08/14).

[16] C.-F. Wang, "Benchmark analysis of representative deep neural network architectures," Department of Informatics, Systems and Communication, University of Milano-Bicocca, 20126 Milan, Italy. IEEE Access, 10 2018, pp. 1–8, disponível em https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8506339.

[17] O. Russakovsky et al., "ImageNet large scale visual recognition challenge," Int. J. Comput. Vis., vol. 115, no. 3, pp. 211–252, Dec. 2015.

[18] P. Sharma, "Understanding transfer learning for deep learning," Data Science Blogathon. Analytics Vidhya, 09 2021, disponível em https://www.analyticsvidhya.com/blog/2021/10/understanding-transfer-learning-for-deep-learning/.

[19] A. Mari, "Understanding transfer learning for deep learning." PennyLane dev team, 01 2021, disponível em https://pennylane.ai/qml/demos/tutorial_quantum_transfer_learning.html.

[20] J. M. N. G. Pires, "Aprendizagem profunda: Estudo e aplicações." Universidade de Évora, Escola de ciências e tecnologia, departamento de informática, 2017, disponível em https://dspace.uevora.pt/rdpc/handle/10174/23224.

[21] J. Hale, "Comparação entre frameworks, dlframeworks," 2018, disponível em https://gist.github.com/ogyalcin/7392a9eb6addb57244b818b14903cee8#file-dlframeworks-tsv.

[22] ——, "Deep learning framework power scores 2018." TowardsDataScience, 09 2018, disponível em https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a.

[23] ——, "Deep learning frameworks google sheet." TowardsDataScience, 09 2018, disponível em https://docs.google.com/spreadsheets/d/1mYfHMZfuXGpZ0ggBVDot3SJMU-VsCsEGceEL8xd1QBo/edit#gid=0.

[24] D. C. Bento, "Plantai - deteção e identificação de doenças em plantas utilizando deep learning." Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Aplicações, 2019, disponível em https://arxiv.org/pdf/1911.10317.pdf.

[25] PlantVillage, Microsoft. Disponível em https://plantvillage.psu.edu/.

[26] P. J. P. K. S. K. Davinder Singh, Naman Jain and N. Batra, "Plantdoc: A dataset for visual plant disease detection." Indian Institute of Technology Gandhinagar, Gujarat, India 382 355, 11 2019, disponível em https://arxiv.org/pdf/1911.10317.pdf.

[27] . F. M. M. Gianni Fenu, "Diamos plant dataset: A dataset for diagnosis and monitoring plant disease," Agronomy, University of Cagliari. MDPI Open Access Journals, 11 2021, pp. 1–14, 2107, Disponivel em https://doi.org/10.3390/agronomy11112107.

[28] A. Poon, Detecting Plant Health with a YOLOv4 Model, Medium, 2021. Disponível em https://www.agropolis-fondation.fr/.

[29] Redmon, Joseph and Farhadi, Ali, YOLOv3: An Incremental Improvement, arXiv, 2018. Disponível em https://pjreddie.com/darknet/yolo/.

[30] R. A. Pagaduan, M. C. R. Aragon, and R. P. Medina, "iblur-detect: Image blur detection techniques assessment and evaluation

study," in *Proceedings of the International Conference on Culture Heritage, Education, Sustainable Tourism, and Innovation Technologies - Volume 1: CESIT,*, INSTICC. SciTePress, 2020, pp. 286–291, 10.5220/0010307700003051, 978-989-758-501-2, Disponivel em https://www.scitepress.org/Papers/2020/103077/103077.pdf.

[31] Kody G. Dangtongdee, Computer Engineering Department, College of Engineering, Plant Identification Using Tensorflow, 2019. Disponível em https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1271&context=cpesp.

[32] Jansirani Sankar, Sri Ramakrishna Engineering College, AN OPEN CV BASED AUTOMATIC LEAF DISEASE IDENTIFICATION AND FERTILIZED AGROBOT USING IOT, 2019. Disponível em https://www.researchgate.net/publication/335813450_AN_OPEN_CV_BASED_AUTOMATIC_LEAF_DISEASE_IDENTIFICATION_AND_FERTILIZED_AGROBOT_USING_IOT.

[33] Jung-Hwa Kim and Jin-Woo Jeong. Gaze in the dark: Gaze estimation in a lowlight environment with generative adversarial networks. Sensors, 20(17):4935, 2020. Disponível em https://www.researchgate.net/publication/344000432_Gaze_in_the_Dark_Gaze_Estimation_in_a_Low-Light_Environment_with_Generative_Adversarial_Networks.

[34] C. Chen, Q. Chen, J. Xu, and V. Koltun. Learning to see in the dark. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018. Disponível em https://arxiv.org/pdf/1805.01934.pdf.

[35] Gibran Benitez-Garcia, Jesus Olivares-Mercado, Gualberto Aguilar-Torres, Gabriel Sanchez-Perez and Hector Perez-Meana, Face Identification Based on Contrast Limited Adaptive Histogram Equalization (CLAHE), Mechanical and Electrical Engineering School of National Polytechnic Institute of Mexico. Mexico, Mexico D.F, 2012. Disponível em https://www.researchgate.net/publication/268434391_Face_Identification_Based_on_Contrast_Limited_Adaptive_Histogram_Equalization_CLAHE.

[36] Anwesa Roy, Pooja Aher, Krushna Kalaskar,Priya Agarwal, Computer Engineering Department,3Students, Dept. of Computer Engineering, Blur Classification and Deblurring of Images , 2017. Disponível em https://www.irjet.net/archives/V4/i4/IRJET-V4I4460.pdf.

[37] M.Kalpana Devi, R.Ashwini, Department of CSE, Jansons Institute of Technology, Karumathampatti, An Analysis on Implementation of various Deblurring Techniques in Image Processing, 2016. Disponível em https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.695.8358&rep=rep1&type=pdf.

[38] Fagun Vankawala, Amit Ganatra, Amit Patel, Department of Computer Engineering Charusat, A Survey on different Image Deblurring Techniques, 2015. Disponível em https://www.irjet.net/archives/V3/i12/IRJET-V3I12241.pdf.

[39] Miika Aittala and Frédo Durand, Massachusetts Institute of Technology, Cambridge MA 02139, USA, Burst Image Deblurring Using Permutation Invariant Convolutional Neural Networks, 2018. Disponível em https://openaccess.thecvf.com/content_ECCV_2018/papers/Miika_Aittala_Burst_Image_Deblurring_ECCV_2018_paper.pdf.

[40] Shengyang Dai and Ying Wu, EECS Department, Northwestern University, Evanston, IL 60208, USA, Removing Partial Blur in a Single Image. Disponível em http://users.ece.northwestern.edu/~sda690/PartialBlur_CVPR09.pdf.

[41] Oliver Whyte, Josef Sivic and Andrew Zisserman, Dept. of Engineering Science University of Oxford, Deblurring Shaken and Partially Saturated Images. Disponível em https://www.di.ens.fr/willow/research/saturation/whyte11.pdf.

[42] Pl@ntNet. Disponível em https://plantnet.org/en/.

[43] H. Xu and C. G. Zhao, "Softer loss transfer distilling and saliency detection for image-based plant diseases diagnosis," 2018.

[44] Xu, H and Zhao, C G, leafweb, 2018. Disponível em https://www.agropolis-fondation.fr/.

[45] Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization, Cornel University. Disponível em https://arxiv.org/abs/1610.02391.

[46] Fchollet, Grad-CAM class activation visualization, 2021. Disponível em https://keras.io/examples/vision/grad_cam/.

[47] Daniel Reiff, Understand your Algorithm with Grad-CAM, Jul 21, 2021. Disponível em https://towardsdatascience.com/understand-your-algorithm-with-grad-cam-d3b62fce353.

[48] R. Draelos, MD, PhD, Grad-CAM: Visual Explanations from Deep Networks, May 29, 2020. Disponível em https://glassboxmedicine.com/2020/05/29/grad-cam-visual-explanations-from-deep-networks/.

[49] OpenCV, Google, Histogram Equalization, 2022. Disponível em https://docs.opencv.org/4.x/d5/daf/tutorial_py_histogram_equalization.html.

[50] G. Benitez-Garcia, J. Olivares Mercado, G. Aguilar-Torres, G. Sanchez-Perez, and H. Perez-Meana, "Face identification based on contrast limited adaptive histogram equalization (clahe)," vol. 1, 04 2012.

[51] R. C. Gonzalez and R. E. Woods, Digital Image Processing, Third Edition, 2008. Disponível em https://www.math.uci.edu/icamp/courses/math77c/demos/hist_eq.pdf.

[52] How can I adjust contrast, 2019. Disponível em https://stackoverflow.com/questions/10549245/how-can-i-adjust-contrast-in-opencv-in-c.

[53] Richardson–Lucy deconvolution, Wikipedia. Disponível em https://en.wikipedia.org/wiki/Richardson-Lucy_deconvolution.

[54] Wiener deconvolution, Wikipedia. Disponível em https://en.wikipedia.org/wiki/Wiener_deconvolution.

[55] Rudi Rottenfusser - Zeiss Microscopy Consultant, 46 Landfall, Falmouth, Massachusetts, 02540. Erin E. Wilson and Michael W. Davidson - National High Magnetic Field Laboratory, 1800 East Paul Dirac Dr., The Florida State University, Tallahassee, Florida, 32310., The ideal point spread function, 2023. Disponível em https://zeiss-campus.magnet.fsu.edu/articles/basics/psf.html.

[56] Huygens Imaging Academy, Scientific Volume Imaging Point Spread Function, 2023. Disponível em https://svi.nl/Huygens-Imaging-Academy.

[57] Blind deconvolution, Wikipedia. Disponível em https://en.wikipedia.org/wiki/Blind_deconvolution.

[58] Li Fei-Fei and Rob Fergus and Pietro Perona, Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories, FeiFei2004LearningGV, Computer Vision and Pattern Recognition Workshop, 2004. Disponível em https://www.tensorflow.org/datasets/catalog/caltech101.

[59] Griffin, G. Holub, AD. Perona, P, Caltech 256 Image Dataset, Caltech Technical Report. Disponível em https://www.kaggle.com/datasets/jessicali9530/caltech256.

[60] Ashley Poon, Detecting Plant Health dataset, 2021. Disponível em https://drive.google.com/drive/folders/14EO3jQPyI2OeHs5hLdtDXpN3hMY5izXA.

[61] J, A. Pandian; Gopal, Geetharamani (2019), "Data for: Identification of Plant Leaf Diseases Using a 9-layer Deep Convolutional Neural Network", Mendeley Data, V1, doi: 10.17632/tywbtsjrjv.1. Disponível em https://data.mendeley.com/datasets/tywbtsjrjv/1.

[62] David P. Hughes and Marcel Salath, "An open access repository of images on plant health to enable the development of mobile disease diagnostics through machine learning and crowdsourcing", CoRR, abs/1511.08060, 2015, arXiv, 1511.08060, 2018. Disponível em https://www.tensorflow.org/datasets/catalog/plant_village.

[63] Thapa, Ranjita; Zhang, Kai; Snavely, Noah; Belongie, Serge; Khan, Awais. The Plant Pathology Challenge 2020 data set to classify foliar disease of apples. Applications in Plant Sciences, 8 (9), 2020. Disponível em https://www.kaggle.com/competitions/plant-pathology-2020-fgvc7/overview/description.

[64] R. Rahman. Plant disease recognition dataset, 2021. Disponível em https://www.kaggle.com/datasets/rashikrahmanpritom/plant-disease-recognition-dataset.

[65] Gianni Fenu, Francesca Maridina Malloci. (2021). DiaMOS Plant Dataset: A Dataset for Diagnosis and Monitoring Plant Disease (Version 1) [Data set]. Zenodo. Disponível em https://zenodo.org/record/5557313#.Yt2sGnbMJPb.

[66] E. Mavridaki, V. Mezaris, "No-Reference blur assessment in natural images using Fourier transform and spatial pyramids", Proc. IEEE International Conference on Image Processing (ICIP 2014), Paris, France, October 2014. Disponível em https://mklab.iti.gr/results/certh-image-blur-dataset/.

[67] Prakharroy, Intuition of Adam Optimizer. Disponível em https://acervolima.com/intuicao-de-adam-otimizador/.

[68] Detecção Single Shot Multibox (SSD), Dive Into Deep Learning. Disponível em https://pt.d2l.ai/chapter_computer-vision/ssd.html.