DEPARTAMENTO DE ENGENHARIA MECÂNICA E INDUSTRIAL

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DANIEL CASTRO SIMÕES

Licenciado em Ciências da Engenharia Mecânica

SISTEMA DE VISÃO TERMOGRÁFICO BASEADO EM APRENDIZAGEM PROFUNDA PARA DETEÇÃO DE DEFEITOS INTERNOS EM MATERIAIS COMPÓSITOS DE MATRIZ POLIMÉRICA

DISSERTAÇÃO PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA MECÂNICA

MESTRADO EM ENGENHARIA MECÂNICA

Universidade NOVA de Lisboa

setembro, 2023

DEPARTAMENTO DE ENGENHARIA MECÂNICA E INDUSTRIAL

SISTEMA DE VISÃO TERMOGRÁFICO BASEADO EM APRENDIZAGEM PROFUNDA PARA DETEÇÃO DE DEFEITOS INTERNOS EM MATERIAIS COMPÓSITOS DE MATRIZ POLIMÉRICA

DISSERTAÇÃO PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA MECÂNICA

DANIEL CASTRO SIMÕES

Licenciado em Ciências de Engenharia Mecânica

Orientador: Nuno Alberto Marques Mendes
Professor Auxiliar, Faculdade de Ciências e Tecnologia da Universidade NOVA de Lisboa

Coorientador: Miguel Araújo Machado
Professor Auxiliar, Faculdade de Ciências e Tecnologia da Universidade NOVA de Lisboa

Júri:

Presidente: Doutora Catarina Isabel Silva Vidal,
Professora Auxiliar da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Arguentes: Doutor Pedro Mariano Simões Neto,
Professor Associado com Agregação da Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Orientador: Doutor Nuno Alberto Marques Mendes,
Professor Auxiliar da Faculdade de Ciências e Tecnologia da Universidade NOVA de Lisboa

MESTRADO EM ENGENHARIA MECÂNICA

Universidade NOVA de Lisboa
setembro, 2023

SISTEMA DE VISÃO TERMOGRÁFICO BASEADO EM APRENDIZAGEM PROFUNDA PARA DETEÇÃO DE DEFEITOS INTERNOS EM MATERIAIS COMPÓSITOS DE MATRIZ POLIMÉRICA

Dedico esta dissertação aos meus pais e ao meu irmão.

.

AGRADECIMENTOS

Gostaria de começar por agradecer ao meu orientador desta dissertação, o Professor Nuno Mendes, por todo o apoio técnico e disponibilidade que apresentou durante a realização deste trabalho. Agradecer também ao meu coorientador, o Professor Miguel Machado, por todas as sugestões relacionadas com a parte mais experimental.

Agradecer à FCT e, em especial, aos docentes responsáveis pela lecionação do curso de Engenharia Mecânica, pelo conhecimento e experiências passadas ao longo destes 5 anos, que, sem dúvida, fizeram de mim uma pessoa mais evoluída a nível profissional e pessoal.

Agradecer a todos as pessoas que tive oportunidade de conhecer e que estiveram no meu caminho ao longo destes 5 anos. Em especial, agradecer às grandes amizades que contruí e que com certeza ficarão para a vida e passo a enumerar: Catarina Matos, João Barbeiro, Miguel Matos, Pedro Tavares, Mafalda Palma, João Filipe, David Caçador e Marcos Balau. Sem dúvida que todos os bons momentos construídos em redor deste grupo de amigos tornaram esta caminhada muito mais fácil.

Queria agradecer também a uma das pessoas mais importantes que apareceu no meu caminho, a minha namorada, Inês Carvalho e que foi uma das maiores surpresas que a faculdade me reservou. Todo o teu apoio e carinho fizeram com que esta batalha parecesse sempre mais fácil. Obrigado do fundo do meu coração!

Por fim, agradecer à minha família, em especial, aos meus pais e ao meu irmão por todo o apoio, motivação e coragem que me deram para enfrentar este longo trajeto e pela oportunidade que me deram para me preparar melhor para a vida profissional. Sem dúvida que estes 5 anos fizeram-me tornar o homem que sou hoje.

RESUMO

A presente dissertação contemplou o desenvolvimento de um sistema de visão termográfico baseado em aprendizagem profunda para a deteção de defeitos internos em materiais compósitos de matriz polimérica reforçados com fibras de vidro, carbono ou *kevlar*.

Para o seu desenvolvimento, foram utilizados provetes produzidos por manufatura aditiva, especificamente pelo processo *Fused Deposition Modeling* (FDM). A metodologia englobou o uso da termografia ativa como ferramenta para adquirir imagens termográficas e posterior rotulações, seguido da utilização de um modelo *Region Convolutional Neural Network* (R-CNN). O treino e teste desse modelo foram realizados por meio de *Transfer Learning* (TF), utilizando uma *ResNet 50*. Algoritmos como *Selective Search* (SS) e *Intersection Over Union* (IoU) foram fundamentais para a sua implementação.

A implementação de métodos de *undersampling* e *oversampling* para o balanceamento de dados, juntamente com a variação do *Learning Rate* (LR), foram fatores a ter em conta para a obtenção de resultados nas diferentes métricas (precisão, *recall*, *F1-score* e exatidão) na fase de teste da CNN. Para os testes de inspeção *offline* da CNN utilizando imagens foram obtidos resultados superiores a 92% para todas as métricas.

Por fim, foram realizados testes de inspeção *inline* utilizando um vídeo completo do ensaio de inspeção (termografia), alcançando uma exatidão na deteção de defeitos de 84,4%.

Palavras-chave: deteção, termografia, *Region Convolutional Neural Network* (R-CNN), *Transfer Learning* (TF), *Selective Search* (SS)

ABSTRACT

This dissertation involved the development of a thermographic vision system based on deep learning to detect internal defects in polymer matrix composite materials reinforced with glass, carbon or kevlar fibres.

For its development, specimens produced by additive manufacturing were used, specifically by the Fused Deposition Modelling (FDM) process. The methodology included the use of active thermography as a tool for acquiring thermographic images and subsequent labelling, followed by the use of a Region Convolutional Neural Network (R-CNN) model. This model was trained and tested by means of Transfer Learning (TF), using a ResNet 50. Algorithms such as Selective Search (SS) and Intersection Over Union (IoU) were fundamental to its implementation.

The implementation of undersampling and oversampling methods for data balancing, together with the Learning Rate (LR) variation, were factors to be taken into account in order to obtain results in the different metrics (precision, recall, F1-score and accuracy) in the CNN testing phase. For the CNN offline inspection tests using images, results of over 92% were obtained for all the metrics.

Finally, inline inspection tests were carried out using a complete video of the inspection test (thermography), achieving a defect detection accuracy of 84.4%.

Keywords: detect, thermography, Region Convolutional Neural Network (R-CNN), Transfer Learning (TF), Selective Search (SS)

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

SIGLAS

AR	Aprendizagem Residual
bb	*bounding box*
bbs	*bounding boxes*
CNN	*Convolutional Neural Network*
CNNs	*Convolutional Neural Networks*
DL	*Deep Learning*
END	Ensaios Não Destrutivos
IA	Inteligência Artificial
IoU	*Intersection over Union*
LR	*Learning Rate*
ML	*Machine Learning*
R-CNN	*Region Convolutional Neural Network*
SS	*Selective Search*
SVM	*Support Vector Machine*
TL	*Transfer Learning*
TN	Negativos Verdadeiros

1

INTRODUÇÃO

1.1 Motivação

Nos tempos atuais, a evolução tecnológica é de tal maneira acentuada que até a própria humanidade parece ter dificuldade em acompanhá-la. Em consequência, a existência de produtos mais complexos e evoluídos levam a que seja necessário também a utilização de técnicas de inspeção que revelem a existência, ou não, de defeitos e ao desenvolvimento de novos materiais que potenciem todo este desenvolvimento.

Para a inspeção de defeitos nos produtos fabricados torna-se necessário, então, aplicar técnicas de Ensaios Não Destrutivos (END). Nesta ótica, a inspeção visual revela-se uma das chaves para o fabrico de peças sem defeitos [2]. Assim sendo, de entre os END disponíveis, a termografia é vista como uma das técnicas com maior potencial para a inspeção de componentes de componentes de maior complexidade, porque permite obter imagens térmicas da peça a inspecionar, com alguma resolução e sem contacto, recorrendo a câmaras específicas e fontes de calor, permitindo identificar defeitos internos nos mesmos.

A necessidade de existência de materiais mais competitivos e com propriedades melhoradas, implica que seja preciso recorrer a materiais compósitos, uma classe de materiais avançados que são produzidos pela combinação de 2 ou mais materiais diferentes no estado sólido com diferentes propriedades químicas, físicas e mecânicas [3]. Nesta dissertação, vão ser utilizados materiais compósitos de matriz polimérica, que são vistos como um desafio de inspeção devido à sua complexidade e às suas propriedades únicas, como a baixa condutividade térmica e elétrica [4].

A inspeção dos materiais pode ser realizada manualmente ou, então, automatizando o processo. Se efetuarmos um termo comparativo, o simples facto de existir um alto nível de automação nas operações de fabrico faz com que seja necessário a utilização de sistemas de deteção de defeitos automáticos [5]. Para além disso, é fácil afirmar que o processo será muito mais rápido, versátil e preciso, se for automatizado e isto está a ser cada vez mais explorado nos dias de hoje, utilizando, para isso, modelos de *Machine Learning* (ML) e de aprendizagem profunda (*Deep Learning* (DL)). Pela sua capacidade de aprendizagem autónoma e identificação e caraterização dos defeitos, o DL é visto como o futuro da inspeção automatizada, tendo tido bastante sucesso em áreas como a identificação e classificação de objetos, deteção facial ou diagnósticos de falhas [5].

Tendo em conta a dificuldade em inspecionar materiais compósitos e a necessidade de potencializar o DL para este propósito, surge, então, a presente dissertação, onde será necessário elaborar um sistema de visão termográfico baseado em modelos de aprendizagem profunda para a deteção de defeitos internos em materiais compósitos de matriz polimérica.

1.2 Objetivos

O principal objetivo desta dissertação assenta no desenvolvimento de um sistema de visão termográfico baseado em modelos de aprendizagem profunda que permitam a deteção automática de defeitos internos em materiais compósitos de matriz polimérica. Para este objetivo ser atingido, é fundamental a criação de uma base de dados com imagens rotuladas, conceber o modelo a utilizar, treinando-o e avaliando-o.

Como um objetivo mais secundário e posterior, será necessário realizar testes de inspeção *inline*, utilizando o vídeo do ensaio de termografia, de modo a avaliar a exatidão do modelo desenvolvido.

1.3 Estrutura

O documento encontra-se estruturado em 6 capítulos e, posteriormente, subdivididos em alguns subcapítulos.

No que concerne ao capítulo 1, encontra-se descrita a motivação para a realização da dissertação, os seus objetivos e a estrutura do documento.

No capítulo 2, será realizada uma abordagem aos sistemas automáticos de inspeção visual e um leve levantamento das suas vantagens comparativamente à inspeção visual manual, depois uma leve abordagem à Inteligência Artificial (IA), ML e um foco mais específico em DL e, por fim, uma revisão bibliográfica ao que está a ser utilizado, focando, principalmente, na inspeção automática utilizando termografia, utilizando metodologias de IA, ML e DL e ainda utilizando metodologias convencionais.

No capítulo 3, apresenta-se toda a montagem experimental utilizada para os ensaios de termografia, bem como o procedimento experimental seguido para os mesmos. Ainda é dada uma breve abordagem aos provetes que foram utilizados nos ensaios

No capítulo 4, é apresentado o desenvolvimento do Sistema de Identificação de Defeitos, onde é apresentado detalhes sobre a arquitetura do mesmo, da automatização do ensaio de termografia, da geração de uma base de dados a partir do uso da termografia e, por fim, o desenvolvimento do modelo *Region Convolutional Neural Network* (R-CNN), onde são apresentados aspetos referentes à geração de dados de treino e de teste, à aplicação de um processo denominado de *Transfer Learning* (TL) e, por fim, detalhes relativos ao processo de treino e teste da *Convolutional Neural Network* (CNN) escolhida no processo anterior.

No capítulo 5, apresentam-se os resultados obtidos com a implementação do algoritmo, mais especificamente, dados relativos ao treino e teste do modelo para a identificação dos defeitos internos nos provetes utilizados. Por último, realizam-se testes de inspeção *inline* de modo a avaliar a exatidão do modelo desenvolvido.

Por fim, no capítulo 6, expõem-se as conclusões relativas à dissertação realizada, bem como as propostas de trabalhos futuros a ter em conta.

.

2

ESTADO DE ARTE

2.1 Inspeção visual na deteção de defeitos

A inspeção visual é um dos END mais básicos e rápidos para efetuar a avaliação da qualidade de produtos e ainda uma maneira de inspecionar e testar materiais. Devido à sua simplicidade, é, na maioria dos casos, utilizado como primeiro método de inspeção [6].

É possível distinguir dois tipos de inspeção visual: manual e automática. Para a deteção de defeitos, pode-se afirmar que a inspeção automática predomina sobre a inspeção manual. Por um lado, a inspeção manual é uma operação com um tempo de duração maior e com uma subjetividade na obtenção de resultados também maior, devido a ser efetuada por um ser humano que possui alterações emocionais e comportamentais. Estas alterações podem influenciar na hora da realização de uma inspeção mais cuidada e detalhada para a obtenção de uma qualidade do produto menor. Por fim, o próprio método utilizado impede ver defeitos relacionados com o interior do produto ou material, tais como falhas ou porosidades. Por outro lado, a inspeção automática permite obter os resultados pretendidos em menor tempo, onde é possível notar que é facilmente adaptado em ambientes inapropriados, trabalhando por longos períodos com alta precisão e eficiência [7], permitindo assim a obtenção de produtos com maior qualidade e eliminando o erro humano decorrente da inspeção [8].

O foco a abordar neste trabalho será, então, as potencialidades de utilizar uma inspeção visual automática e como isso está a influenciar o mundo dos END e da inspeção de defeitos.

Se olharmos para o mundo da indústria, a necessidade de se produzir a velocidades cada vez maiores e com maior complexidade nos materiais e processos utilizados, faz com que seja

natural que a introdução da automatização no processo de inspeção acontece, sendo vista como um problema desafiador e importante para a inspeção [6].

Para a existência de sistemas de inspeção visual automáticos serem uma realidade, é fundamental a evolução tecnológica a nível computacional que se tem verificado e que tem em conta a necessidade humana de tentar automatizar tarefas de modo a garantir uma otimização máxima das mesmas. Mais concretamente, o desenvolvimento do que se chama de visão por computador permitiu efetuar a aquisição, o processamento e a análise de cenários do mundo, que fazem com que seja possível tomar decisões de acordo com o definido para o caso em estudo. Isto veio aproximar estes sistemas ao que um operador faria, se a inspeção fosse realizada de maneira manual, na medida em que a diferença que pode saltar mais à vista é a diferença entre o meio pelo qual o conhecimento é obtido num sistema automático, maioritariamente através de uma câmara [9] ou, então, recorrendo a técnicas específicas, tais como a termografia para obtenção de imagens térmicas. Segundo Mera et al [8], podemos desenvolver um sistema de inspeção visual automático seguindo 3 caminhos:

- *Template-matching*, onde a imagem da peça ou material a inspecionar é comparada com uma ou mais imagens de referência;
- Através de um método de verificação de regras, onde, por iteração, se verifica se as regras pretendidas estão a ser cumpridas;
- Inspeção baseada em técnicas de inteligência artificial (IA), ML e DL, onde o sistema de inspeção aprende e generaliza as relações entre os defeitos encontrados e as caraterísticas do objeto.

Bao et al. [10] desenvolveram um sistema de inspeção visual automático para efetuar a manutenção do revestimento de *bunkers* de uma central elétrica a carvão, de modo a substituir os métodos manuais de inspeção visual de revestimentos individuais que até ao momento eram utilizados e que tinham baixa eficiência, precisão e altos riscos para a saúde do operador que a realizava. Os defeitos que se pretendiam analisar com este sistema eram a existência de corrosão nos parafusos que eram utilizados para fixar os revestimentos individuais continuamente e o empenamento dos bordos do revestimento. O sistema proposto era composto por uma câmara colocada numa plataforma que era movida por um mecanismo paralelo de quatro cabos. Quatro roldanas foram colocadas nos cantos do *bunker* de carvão e o mecanismo que permitia o seu

movimento encontrava-se no seu exterior conectado a um motor onde a sua velocidade de enrolamento era totalmente controlada e programável, de modo a se conseguir posicionar a câmara no local pretendido. Para além disto, devido ao material das placas de revestimento serem um aço inoxidável, ainda foi colocado uma fonte de luz de reflexão difusa, de modo a reduzir, numa parte inicial do processo de inspeção, essas mesmas reflexões. Isto foi introduzido, também, para se obter uma alta qualidade das áreas a serem inspecionadas aquando da aquisição das imagens. Por fim, foi realizada uma ligação entre as imagens adquiridas pela câmara e a interface com um software que permitia a sua visualização em tempo real. O sistema proposto apresenta-se na Figura 2.1. Foi desenvolvido um algoritmo para a deteção automática dos defeitos tendo em conta a posição geométrica espacial em tempo real da câmara, onde se concluiu que o mesmo identificava com precisão e eficiência os defeitos existentes. Para além disso, concluiu-se que este novo sistema poupava 97,5 % do tempo quando comparado com a inspeção manual, o que revela bem as suas capacidades quanto à poupança de tempo em inspeções.

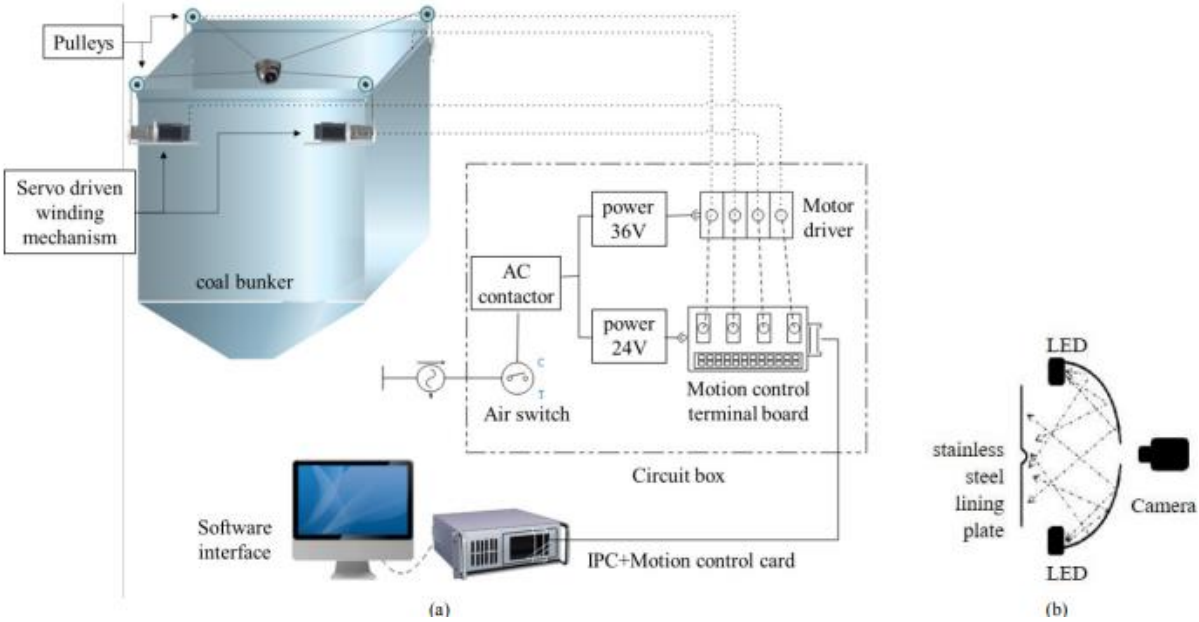


Figura 2.1 - Sistema de inspeção proposto pelos autores [10]
a) Esquema do sistema de inspeção, b) Luz difusa em cúpula

2.2 Inteligência Artificial (IA)

IA pode ser entendida como a ciência para o desenvolvimento de máquinas inteligentes ou programas de computador que tem em vista a replicação da inteligência humana [11] e tem tido uns avanços contínuos e significativos nos últimos anos. Para isso, tem sido fundamental os desenvolvimentos verificados no poder computacional em aceleradores de IA e em arquiteturas de uso geral que possibilitam que novos modelos e paradigmas que são propostos continuamente tenham uma crescente evolução na flexibilidade, aplicabilidade e escalabilidade desta informação orientada para a abordagem [12].

Para a implementação de IA, são utilizados alguns métodos, tais como, modelos cognitivos, sistemas baseados em regras impostas, algoritmos de pesquisa ou representação de conhecimento e planeamento [11].

A evolução notória que se tem verificado nos últimos anos têm sido introduzidas em áreas como a visão computacional, a robótica, os veículos autónomos [13], os diagnósticos médicos [14], a tradução linguística ou até na própria agricultura [15]. Para além do impacto que tem tido nas mais diversas áreas, a IA também pode ser encontrada no processamento de vários tipos de sinal, em grandes quantidades de informação e no processamento e avaliação de imagens [6].

Os métodos utilizados por IA, por si só não são suficientes para resolver todos os problemas, pelo que é necessário recorrer a métodos de aprendizagem que dão mais robustez e permitem resolver problemas mais complexos a seu cargo, tais como os modelos de ML e de DL.

2.3 *Machine Learning* (ML)

ML é um sub-área da IA onde o computador observa uma certa informação que lhe é fornecida e, através do seu processamento, gera modelos que podem ser utilizados para resolver problemas [11]. Dito doutra maneira, os modelos de ML conseguem transformar dados em conhecimentos [16].

Dado que a utilização deste tipo de modelos faz com que seja obrigatório a existência de um processo de treino, é possível separar os modelos segundo a quantidade ou o tipo de supervisão que lhes é fornecida durante o mesmo. Assim sendo, podemos ter três tipos

diferentes: aprendizagem supervisionada, aprendizagem não supervisionada e aprendizagem reforçada. Na primeira abordagem, o modelo de ML é treinado com recurso a informações rotuladas, sendo posteriormente avaliado recorrendo a um conjunto de dados que não foram utilizados no processo de treino. Duas das tarefas que mais utilizam este tipo de aprendizagem são os modelos de classificação e regressão. No que concerne ao segundo tipo de aprendizagem, são utilizados dados não rotulados de modo a determinar possíveis padrões ou estruturas nos mesmos. Isto é utilizado em tarefas relacionadas com deteções de anomalias, com redução dimensional e de visualização e com agrupamentos. Por último, na aprendizagem por reforço existe um compromisso entre o agente e o ambiente de modo a se obter a melhor estratégia para a resolução do problema [11]. Na Figura 2.2 é possível observar um esquema de cada um dos tipos de aprendizagem com mais detalhe.

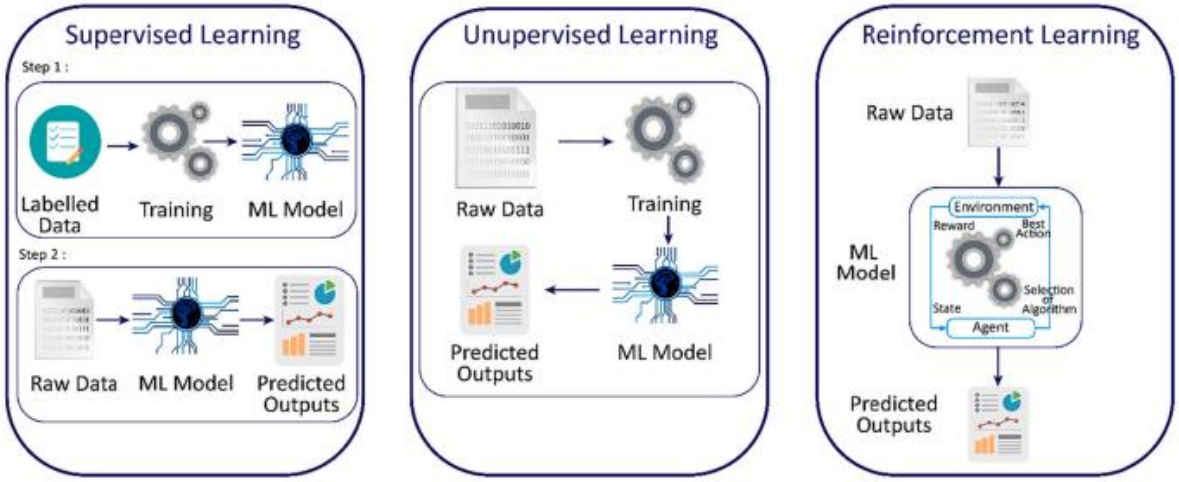


Figura 2.2 - Esquema de classificação dos modelos de acordo com o tipo de aprendizagem [11]

Vistos os diferentes tipos de aprendizagem, é possível indicar alguns modelos que se inserem dentro de ML. Entre eles, *k-Nearest Neighbors* [17], *Support Vector Machines* (SVMs) [18], *Logistic Regression*, *Linear Regression*, árvores de decisão e, por fim, redes neuronais (do inglês *Neural Network*), que vão ser abordadas com mais detalhe no subcapítulo 2.4.

As potencialidades da utilização de modelos ML são elevadas, pelo que estão a ser utilizadas em variadíssimas áreas. A avaliação da qualidade da água é um dos pontos que já está a ser explorado, fazendo uso das capacidades de avaliação dos modelos de aprendizagem supervisionada [19]. A introdução destes modelos em sistemas de energia integrados na

construção possibilita que os edifícios tenham a capacidade de se adaptar ao meio ambiente envolvente e consigam realizar uma gestão de energia independente [20]. Por fim, também tem sido verificada a utilização de algoritmos de ML para a melhoria do desempenho de perfuração nas indústrias de petróleo e gás, através da otimização do parâmetro *Rate of Penetration* (ROP) [21].

2.4 *Deep Learning* (DL)

DL é um sub-grupo de ML que se dedica maioritariamente à aplicação de redes neuronais artificiais nos seus modelos como método de resolução dos problemas colocados. Redes neuronais artificiais (do inglês *Artificial Neural Network* (ANN)) podem ser entendidas como um sistema computacional inspirado na rede neural biológica do cérebro humano, que é constituído por múltiplas camadas de neurónios que se encontram interligados e que funcionam como elementos de processamento [5]. Comparativamente com modelos de ML, os de DL são muito úteis para trabalhar com uma dimensão de dados elevada, devido à sua alta capacidade computacional [11]. Estes são determinantes no desempenho dos algoritmos, na medida em que é necessário garantir que existe quantidade e qualidade dos mesmos para a realização do treino das redes neuronais [22] e, ainda, a existência de uma base de dados balanceada para o algoritmo ser efetivo e preciso para vários aspetos do caso em estudo e não apenas um caso específico.

As aplicações dos modelos de DL podem ser encontradas nas mais diversas áreas, dando ênfase às aplicações baseadas em processamento de imagem, onde é possível denotar um sucesso excecional em tarefas como deteção e classificação de objetos, deteção facial, reconhecimento padrão, diagnóstico de falhas e seguimento de alvos [5].

Existem vários tipos de redes neuronais, tais como *Deep Neural Network* (DNN), *Fully Convolutional Network* (FCN), *Generative adversarial network* (GAN) ou *Convolutional Neural Network (*CNN). Estas últimas serão as que merecerão maior destaque e serão abordadas posteriormente.

2.4.1 *Convolutional Neural Networks* (CNNs)

CNNs são um tipo de rede neuronal que incluem uma operação matemática chamada de convolução numa das suas camadas [5]. Na sua estrutura, apresentam três tipos de camadas/níveis: camada de convolução, camada de *pooling*, que é utilizada para reduzir a dimensão da imagem, e uma camada de conexão total, onde ocorre a classificação [23]. Geralmente, as camadas de convolução são seguidas de outra de *max pooling* ou de convolução e a camada de classificação encontra-se no final da estrutura [11]. Estas camadas servem para realizar o processamento da imagem a fim de classificá-la de acordo com as caraterísticas que foram sendo retiradas ao longo da estrutura com os seus devidos pesos. Para iniciar este processo, é necessário colocar um *input*, que são as imagens que se querem analisar. Na Figura 2.3 é possível observar um esquema de uma estrutura geral de uma CNN.

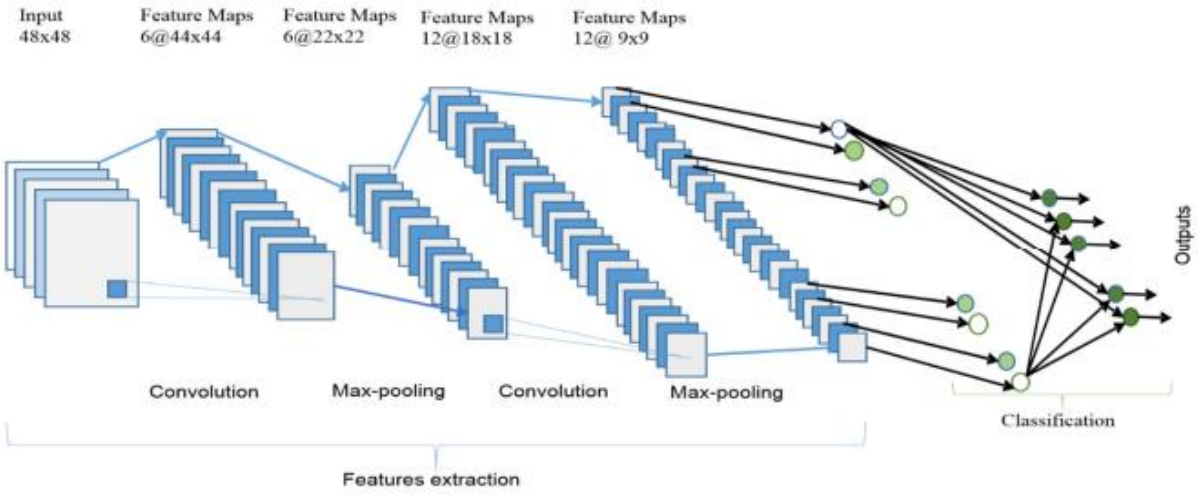


Figura 2.3 - Arquitetura geral de uma CNN [23]

É possível afirmar que a camada de convolução é o pilar fundamental da estrutura deste tipo de redes neuronais. Utiliza componentes como detetores de filtro/ *kernel*/ caraterísticas e um mapa de caraterísticas. Basicamente, a convolução é um processo onde se seleciona uma parte da imagem inicial e onde é realizado, através do uso do detetor referido, um varrimento em toda a imagem para verificação das suas caraterísticas e posterior apresentação 2D de pesos que contém cada zona da imagem. Para se obter a matriz resultante deste processo, é efetuada a multiplicação entre valor do pixel da imagem e os pesos dos filtros. O número de filtros

utilizados tem influência na profundidade dos resultados [11]. Depois de cada convolução, uma função de ativação é aplicada.

Existem alguns híper parâmetros que influenciam e ditam a arquitetura dos algoritmos a utilizar, tais como o tamanho do filtro, o número de filtros ou os passos que são necessários realizar [11]. Para além disto, é preciso também ter em conta o *Learning Rate* (LR), visto que a sua otimização vai ter impacto no número de ciclos necessários para treinar o modelo e na sua taxa de sucesso.

CNNs são utilizadas, principalmente, para processamento de imagem, no que se refere à deteção e classificação de objetos. Tendo isto em conta, é possível identificar alguns modelos que foram desenvolvidos e que derivam do uso das caraterísticas deste tipo de redes neuronais. *Region Convolutional Neural Network* (R-CNN) [24] é uma rede neuronal que foi introduzida para reduzir o tempo de processamento [25] e aumentar a velocidade. Mais tarde, foram introduzidas a *Fast* R-CNN e a *Faster* R-CNN, que tornaram o processo sucessivamente mais rápido e, por fim, a *Mask* R-CNN que veio para facilitar o treino e generalizar tarefas, o que provocou que fosse possível realizar outras tarefas com o mesmo enquadramento [25].

2.5 Sistemas de inspeção

A inspeção visual desempenha um papel fundamental para a identificação de defeitos em produtos ou materiais. O facto de ter sido possível a automatização dos processos de inspeção, fez com que o tempo de inspeção diminuísse de maneira drástica e que a qualidade do produto final fosse aumentando gradualmente.

Para a geração destes processos automatizados, é importante a utilização de câmaras que permitam a obtenção das imagens que irão ser processadas. Nesta ótica, a termografia, um END, pode desempenhar uma peça chave para a inspeção. O facto de as câmaras utilizadas para o processo de termografia usarem radiação infravermelha permite a obtenção de imagens térmicas que são muito interessantes de analisar e possibilitam a inspeção de defeitos não apenas superficiais, mas também internos.

Em adição, as potencialidades do uso de modelos de IA, ML e DL para a deteção e classificação podem ser perfeitamente interligadas com o uso de câmaras termográficas, devido a serem necessárias bases de dados que permitam aos modelos, principalmente de ML e DL, terem um treino adequado e tirarem assim o máximo partido dos mesmos para este propósito.

Para além disso, a elevada capacidade computacional destes modelos possibilita uma inspeção cada vez mais complexa e detalhada dos objetos a inspecionar.

Jacintha et al. [26] realizaram um estudo para a identificação e categorização de defeitos em válvulas de encaixe de ferro fundido. Para isso, foi utilizada uma câmara térmica que seguia as caraterísticas de termografia infravermelha passiva de modo a detetar defeitos como fissuras, porosidades ou defeitos internos. Numa primeira fase, apenas foi utilizada a câmara para a inspeção dos defeitos, mas rapidamente foi notado que não ocorriam flutuações no fluxo de calor que permitissem observar tais defeitos. Tendo isto em conta, optaram por combinar este ensaio de termografia com líquidos penetrantes, o que ajudou na identificação e observação desses mesmos defeitos. Para a segmentação dos defeitos da válvula foi proposta a variação local ponderada baseada em pixéis para efetuar o agrupamento difuso da estrutura (WLVPBFC), que consiste na alteração da escala média de cinza por uma escala de cinza de filtro ponderado e na utilização da função Gaussiana de base radial para obter boa precisão, de modo a ser possível o agrupamento de pixéis com valores com peso parecidos nas imagens. Na Figura 2.4 encontra-se o diagrama com o modelo proposto.

Para a obtenção dos resultados, utilizaram-se três filtros: um com um valor médio, outro com um valor mediano e um com um filtro ponderado. Para se verificar a correta classificação, utilizou-se vários parâmetros de avaliação, tais como exatidão, precisão ou sensibilidade. Os autores concluíram que o filtro ponderado se destacou em algumas métricas utilizadas, como as duas primeiras mencionadas, alcançando valores de 69,4 % e 80,1 %, respetivamente e recomendaram futuros trabalhos utilizando algoritmos de DL em vez do utilizado.

Continuando a explorar a deteção de defeitos utilizando sistemas automáticos com termografia, Ghidoni et al. [2] apresentaram um sistema de inspeção visual automatizado para a inspeção de fissuras em peças metálicas. A termografia foi a técnica usada para a obtenção das imagens a analisar e a parte da automação deriva do uso de um braço robótico para efetuar o movimento de modo a varrer a superfície de toda a peça, devido às ondas térmicas serem geradas um pouco por toda a peça pela fonte. O sistema encontra-se formado, então, por um robô, uma câmara térmica e uma fonte laser e é apresentado na Figura 2.5.

Figura 2.4 - Diagrama do modelo proposto para a identificação dos defeitos [26].

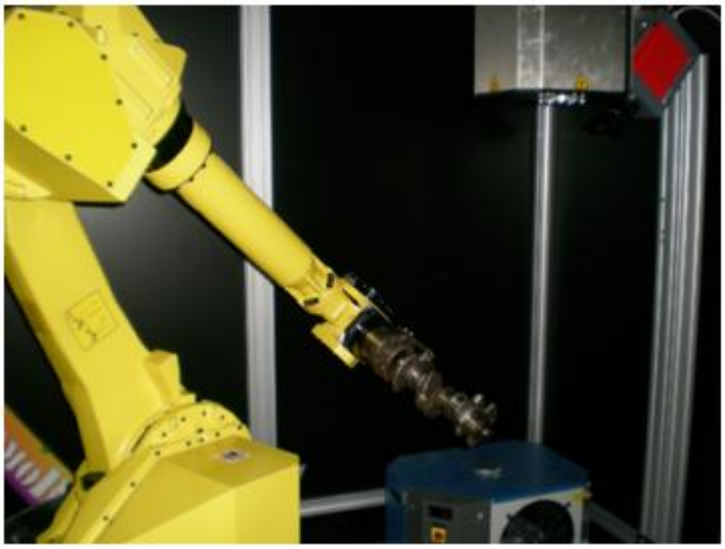


Figura 2.5 - Sistema de inspeção automatizado gerado por Ghidoni et al. [2].

Para realizar os testes ao sistema, utilizou-se uma árvore de Cames metálica. Para a realização da classificação foi aplicado um modelo SVM, de ML. Os resultados permitiram aos autores perceber que o sistema criado era adequado para a inspeção de peças de geometria complexa, que era muito robusto devido ao conhecimento completo do processo de imagem e que permitia identificar fissuras de reduzida dimensão.

Um caso específico de ligação um sistema automático de inspeção ligando termografia e ML é o estudo feito por Choudary et al. [27], onde o objetivo era realizar um diagnóstico efetivo e constante para detetar possíveis falhas nos rolamentos de motores de indução. Mais uma vez, a termografia foi utilizada para a captura das imagens necessárias. Apesar de ser considerado um ensaio com muitas vantagens, neste caso específico, os autores depararam-se com um problema relacionado com a presença de ruído e com informações insignificantes que afetavam o desempenho do processo. Para resolver isto, introduziram um método baseado em transformação de onda bidimensional discreta (sigla em inglês, 2D-DWT) e usaram-no para testar diferentes falhas associadas ao funcionamento dos rolamentos, tais como falta de lubrificação e defeitos interiores ou exteriores.

Para se conseguir extrair os melhores atributos, foi necessária a aplicação do *Principal Component Analysis* (PCA) e, posteriormente, utilizou-se o critério de *Mahalanobis distance* (MD) para selecionar as caraterísticas para se obter um resultado ótimo.

Para efetuar a classificação dos defeitos e avaliar o desempenho dos diferentes algoritmos de ML, os autores utilizaram o *Complex Decision Tree* (CDT), a *linear discriminant analysis* (LDA) e ainda o *Support Vector Machine* (SVM). Toda esta metodologia proposta encontra-se na Figura 2.6.

Os resultados obtidos permitiram concluir que o modelo supervisionado de ML, SVM, foi o que revelou ser mais efetivo na classificação de defeitos nos rolamentos, quando comparado com os outros dois modelos utilizados.

Com a revisão da literatura efetuada até este ponto, é possível afirmar que a inspeção de peças metálicas tem um certo domínio nos estudos efetuados. Todavia, é necessário ter em atenção a deteção de defeitos em outros materiais, como por exemplo, materiais compósitos que, como são constituídos por dois ou mais tipos de materiais, são vistos como um desafio para a inspeção.

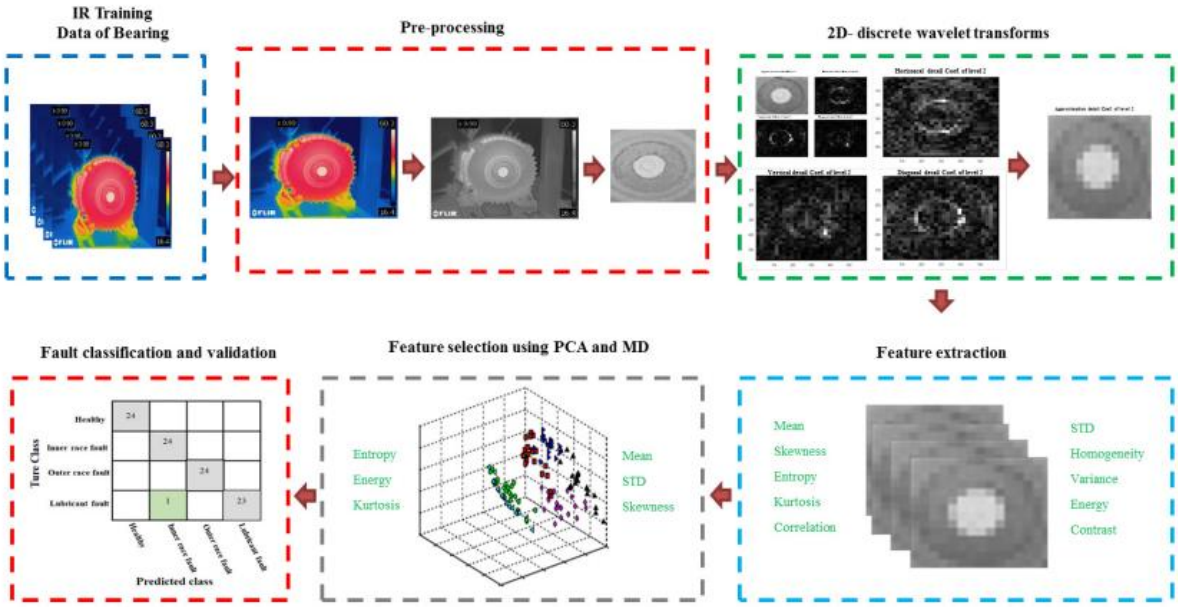


Figura 2.6 - Metodologia proposta pelos autores para a deteção das falhas nos rolamentos [27]

Antonello et al. [28] apresentaram um sistema de inspeção automático para polímeros reforçados com fibra de carbono, no âmbito do projeto Europeu *Thermobot*. Este sistema consistia num braço robótico que continha uma câmara infravermelha, uma lâmpada e o material a analisar.

Defeitos como porosidades, inclusões, fração volúmica incorreta das fibras ou mesmo a posição da cola entre duas folhas de fibra de carbono consecutivas são os mais comuns neste tipo de materiais. Utilizando uma técnica de Termografia por fase pulsada (sigla em inglês, PPT) é possível identificar a disposição da cola nas camadas superficiais do material, pelo que foi o END usado para a inspeção.

Para a análise das imagens térmicas retiradas para inspeção, os autores utilizaram uma técnica convencional chamada *UnSharp Masking* (USM). A técnica consiste na utilização de um algoritmo que é responsável por detetar contornos. Para além desta técnica, primeiramente é aplicado um filtro de ruído, seguido desta técnica, passando posteriormente por outra técnica chamada *thresholding* e, por último, *artifact filtering*, responsável pelo controlo final e possível remoção de possíveis pequenos contornos não necessários. Na Figura 2.7 apresentam-se as três fases da inspeção, com a última imagem a ser aquela onde os defeitos associados às posições da cola estão identificados.

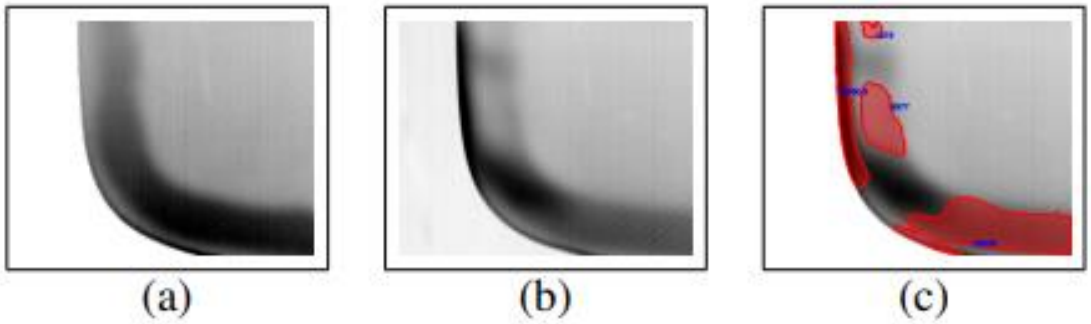


Figura 2.7 - Exemplo de comparação [28].
a) Imagem de referência retirada do ensaio de termografia utilizado; b) Segundo frame; c) Resultado da deteção

Tendo em conta a literatura analisada, é possível afirmar que os sistemas automáticos de inspeção visual já são uma realidade e serão, sem dúvida, o futuro da deteção de defeitos. Para isso, a termografia, enquanto END, tem um papel muito importante, na medida em que permite analisar imagens térmicas e detetar defeitos subsuperficiais, para além dos superficiais. A utilização de modelos de IA, ML e DL vieram para ficar e, devido à sua alta capacidade de computação e aos processos de treino associados, são vistos como uma grande potencialidade a ser explorada para a identificação dos mesmos.

Todavia, é necessário ter em atenção aspetos como o nível de ruído associado ao processo de obtenção das imagens ou o tipo de ensaio mais adequado para a identificação dos diferentes defeitos.

O DL é visto como uma arma com uma potencialidade muito elevada para a deteção e classificação de objetos, permitindo elevar a fasquia da complexidade das peças ou materiais a analisar, pelo que é deveras recomendado a utilização dos seus modelos para a deteção de defeitos. Visto isto, a inspeção de materiais compósitos, utilizando termografia e modelos de DL é um aspeto que ainda não está a ser muito explorado, pelo que é um fator a ter em conta.

Por fim, depois de realizada toda esta análise, o tema da dissertação consiste, então, na análise de defeitos internos em materiais compósitos de matriz polimérica com reforço de fibra de carbono, vidro ou *kevlar*. Para isso, será necessário utilizar um modelo de DL que faz todo o sentido pela alta taxa de sucesso que tem tido na área da identificação e caraterização, aliado ao uso de termografia como meio para se obter as imagens necessárias ao treino do modelo e ao desenvolvimento do trabalho. O seu desenvolvimento poderá vir a ser importante para o

futuro da inspeção de produtos, dada a competitividade que os materiais compósitos apresentam em termos de funcionalidade e propriedades, comparativamente com os metais, por exemplo.

3

METODOLOGIA EXPERIMENTAL

3.1 Estrutura utilizada para os ensaios de termografia

Para a realização da presente dissertação, tornou-se indispensável a utilização do ensaio de termografia ativa para a obtenção de imagens térmicas dos provetes analisados, sendo necessário aquecer a superfície dos mesmos para a localização e visualização dos defeitos presentes.

Tendo isso em conta, tornou-se necessário a utilização de uma montagem experimental que o permitisse, onde o aparato laboratorial empregue foi o seguinte:

- **Câmara termográfica IRS336 da *Automation Technology***: foi a câmara utilizada para captação dos termogramas. O facto de ser termográfica possibilita analisar a evolução térmica ao longo do tempo das superfícies dos provetes a analisar. Esta encontra-se entre a fonte de radiação (lâmpadas) e o provete, estando centrada e a 25 cm do último. É possível observar o seu formato na Figura 3.1. Segundo [29], a câmara apresenta uma resolução de 336 × 256 *px*, uma gama de medição de temperatura que oscila entre os – 25 ºC e os 135 ºC com uma precisão de ± 2 ºC e a ligação para a visualização da imagem é realizada através de *GigE Vision*.

Figura 3.1 - Câmara Termográfica IRS336 da marca *Automation Technology*.[29]

- **Lâmpada Philips PAR38 IR:** foram utilizadas 4 lâmpadas incandescentes infravermelhas para a emissão da radiação necessária à excitação térmica dos provetes em estudo. Cada lâmpada corresponde a 175 W de potência, o que resulta num total de 700 W [30]. A orientação das lâmpadas está de acordo com Gonçalves [9] e encontram-se a uma distância de 60 cm da superfície do provete;
- **Relé de 10 A da FINDER e módulo DAQmx da *National Instruments*:** ambos os constituintes se encontram alimentados por uma fonte de tensão de corrente direta contínua ALF2902M da ELC com 6 V. Estes componentes são fundamentais para que o ensaio de termografia seja possível de realizar automaticamente, acendendo as lâmpadas pelo tempo que é definido pelo utilizador e sendo o módulo DAQmx responsável pela aquisição de dados [30]. A montagem destes aparelhos apresenta-se na Figura 3.2.

Para além dos componentes mencionados anteriormente, será imprescindível mencionar que é estritamente necessário a utilização de um computador que contém a interface para a utilização da montagem experimental e de uma estrutura metálica para se posicionar todos os dispositivos necessários.

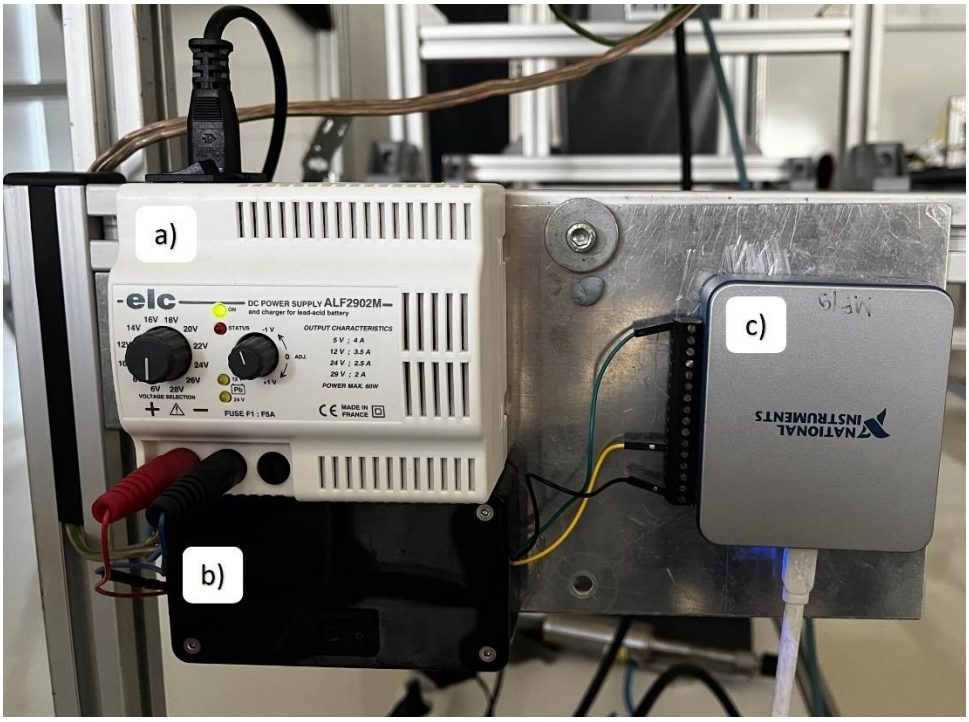


Figura 3.2 - Montagem utilizada para a aquisição de dados. a) Fonte de tensão de corrente direta ALF2902M da ELC; b) Interruptor que contêm o relé de 10 A da FINDER; c) Módulo DAQmx da *National Instruments*.

Para entender melhor como surgiu a montagem experimental, será necessário remeter a estudos realizados em dissertações anteriores.

A bancada utilizada para a realização foi criada e desenvolvida por Gomes [31], sendo posteriormente melhorada por Silva [30], onde o principal destaque vai para a implementação do controlo manual da estação e tendo as suas modificações finais realizadas por Gonçalves [9], onde a direção das lâmpadas utilizadas foi alterada para uma posição oblíqua relativamente à superfície de maneira a otimizar os ensaios realizados. No âmbito desta dissertação, foram adicionados uns suportes para fixar a amostra que irão ser abordados com mais detalhe posteriormente.

Depois de analisada a estrutura que havia sido preparada por Gonçalves [9], observou-se que o local onde os provetes estavam fixados estava em contacto direto com a estrutura metálica de alumínio, pelo que, devido a este material apresentar alta condutividade térmica, haveria o risco de, durante os ensaios, o escoamento térmico observado na superfície do provete ser mais rápida do que o pretendido e impossibilitasse a visualização detalhada e clara da evolução do mesmo. Deste modo, optou-se por bem realizar a modelação 3D e posterior impressão de uns

suportes que permitissem a fixação dos provetes, minimizando, assim, o escoamento do calor para o material envolvente, dada a baixa condutividade térmica dos polímeros, em geral.

Para a projeção destes suportes, reparou-se que os perfis utilizados para a fixação das amostras eram *Bosch* 30 × 30 mm e, a partir daí, utilizaram-se as informações presentes no documento técnico e medições manuais auxiliares para a sua modelação utilizando o software SOLIDWORKS. O seu aspeto pode ser observado na Figura 3.3a), bem como as principais medidas associadas (Figura 3.3b)). Salientar a necessidade de utilizar um parafuso sextavado interior de cabeça escareada e uma porca M5 para realizar o aperto de modo a fixar o provete ao suporte.

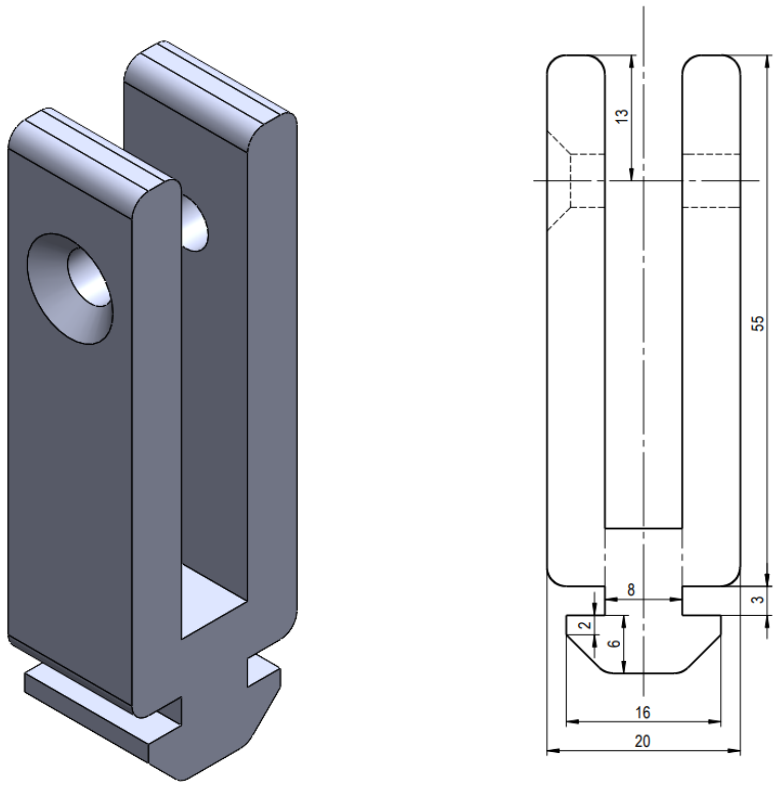


Figura 3.3 - a) Perspetiva isométrica do suporte modelado; b) Vista frontal com as principais medidas.

Depois deste primeiro passo, utilizou-se uma impressora 3D, mais especificamente, uma *Original* Prusa i3 MK3S+, para a produção de dois destes suportes para adicionar à montagem experimental. Para a impressão, utilizou-se como material o polímero termoplástico PLA e um *in-fill* de 20% de modo a minimizar os efeitos provocados pela condução térmica entre os

materiais. Com esta melhora, a montagem experimental passou a estar como o ilustrado na Figura 3.4 e os suportes e sua montagem estão ilustrados na Figura 3.4e).

Figura 3.4 - Montagem experimental. a) Sistema para automatização do ensaio; b) Lâmpada Philips PAR38 IR; c) Câmara termográfica IRS336 da *Automation Technology*; d) Provete; e) Suporte.

3.2 Procedimento experimental

Numa primeira fase, o sistema de inspeção foi montado e ajustado de modo a serem percetíveis a alterações nos resultados provocadas pela variação de alguns fatores, tais como a distância luzes-amostra, a distância câmara-amostra, a existência ou não de luz artificial ou a profundidade a que o defeito se encontrava dependendo da orientação dada ao provete. Para a verificação dos resultados, utilizou-se o código MATLAB elaborado por Gonçalves [9] que permitia avaliar a evolução térmica da superfície do provete através de vídeo e a identificação dos defeitos existentes no mesmo.

Deste modo, conclui-se que o ambiente mais propício e que garantia maior controlo e uniformidade aos ensaios seria um ambiente onde não existissem fontes de calor na direção em que a câmara estaria montada. Para além disso, a luzes presentes na sala também não estariam ligadas, de modo a garantir que o mínimo de luz afetava cada ensaio. Por último, as distâncias

consideradas ótimas seriam de 60 cm, entre a parte mais próxima da câmara e a superfície do provete e de 35 cm, entre a parte frontal da câmara e a superfície do provete.

Para garantir a automatização do ensaio, é necessário realizar a ligação entre o computador e a câmara termográfica através de uma interface *Gigabit Ethernet*, e entre o computador e o módulo DAQmx através de uma ligação USB. Ambas as ligações serão vitais para a visualização dos resultados obtidos da gravação de vídeo no software MATLAB.

Os dados obtidos dos ensaios realizados aos vários provetes que vão ser apresentados no subcapítulo 3.3, serão indispensáveis para a posterior criação de uma base de dados que será utilizada para o modelo R-CNN que vai ser criado, treinado e testado.

3.3 Caracterização dos provetes utilizados

Foram utilizados 4 provetes diferentes fabricados por Silva [30]. Todos apresentam na sua constituição uma matriz composta por 7 camadas de polímero termoplástico PLA. O que distingue um dos outros é o material de reforço empregado. Existe um provete reforçado com fibra de carbono, outro com fibra de vidro, outro com *kevlar* e um último em que não existiu um material de reforço adicionado. Os provetes têm umas dimensões de 120 × 120 mm^2 com uma espessura de 5,5 mm. O aspeto de um provete utilizado pode ser notado na Figura 3.5 e a secção transversal da distribuição do material de reforço para cada provete pode ser observada na Figura 3.6, onde é visível a distinção entre o material e a fibra.

Figura 3.5 - Vista frontal de um dos provetes utilizados (reforçado com fibra de vidro).

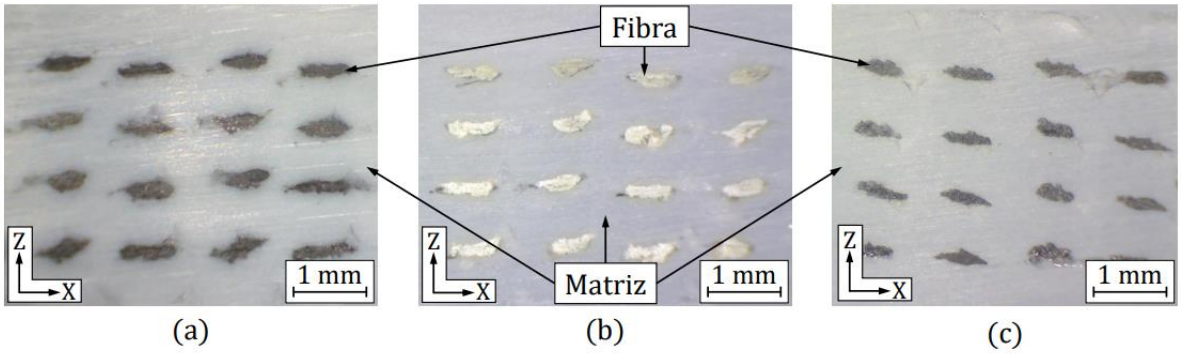


Figura 3.6 - Secção transversal dos provetes. a) Fibra de carbono; b) *Kevlar*; c) Fibra de vidro [30].

No processo de fabrico dos provetes, foram adicionadas duas delaminações retangulares de dimensões 12 × 50 × 0,5 mm^3 centradas no eixo Y, que conferem 2 defeitos a cada um. Um dos defeitos encontra-se situado a meia espessura e o outro a 1,38 mm da superfície superior. Este último pode ser transformado num defeito mais profundo ou menos profundo dependendo da orientação do provete aquando do ensaio [30]. Na Figura 3.7, observa-se com mais detalhe a localização dos defeitos e o tipo de defeito que lhes é associado, segundo o próprio fabricante.

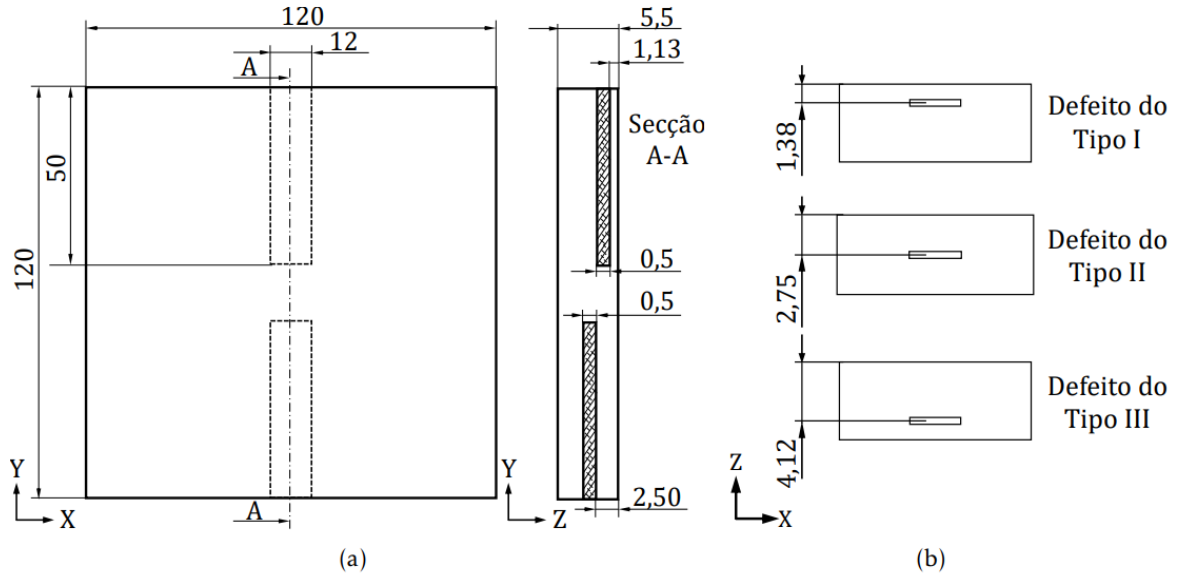


Figura 3.7 - Representação das (a) dimensões dos provetes fabricados e (b) tipos de defeitos [30].

4

DESENVOLVIMENTO DO SISTEMA DE IDENTIFICAÇÃO DE DEFEITOS

4.1 Arquitetura do sistema de identificação de defeitos

Para a realização da presente dissertação, o objetivo principal assenta no desenvolvimento de um sistema de visão termográfico baseado em modelos de aprendizagem profunda que permitam a deteção automática de defeitos internos em materiais compósitos de matriz polimérica.

Para isso, combinou-se o poder da automatização do ensaio de termografia com a capacidade de utilização de redes neuronais para a resolução de problemas, ou seja, utilizaram-se imagens obtidas através do ensaio de termografia para o treino e elaboração de um modelo R-CNN que permite a identificação dos defeitos na superfície dos provetes. O sistema utilizado para a primeira parte encontra-se presente no subcapítulo 3.1, onde o código realizado em MATLAB por Gonçalves [9], no que se refere à obtenção de um vídeo do ensaio realizado, e a posterior adição de gravação de imagens de 10 em 10 *frames* numa dada diretoria, permite ao utilizador conseguir obter os dados necessários para a fase seguinte. Tendo as imagens obtidas sido rotuladas, pode-se passar à próxima fase, onde serão introduzidas num modelo R-CNN, este elaborado recorrendo à linguagem de programação Python, sendo tratadas antes de serem utilizadas por uma rede neuronal, neste caso, uma *ResNet v2 50*, para ser treinada e avaliada ou, então, caso a rede neuronal já tenha sido treinada anteriormente para o problema em questão, as imagens obtidas podem ser diretamente passadas para uma fase de teste, onde, com a ajuda do algoritmo de *Selective Search* (SS) e das predições realizadas pelo modelo, serão

apresentadas as zonas onde os defeitos foram encontrados. Na Figura 4.1 apresenta-se um fluxograma com a arquitetura elaborada para o sistema de identificação de defeitos.

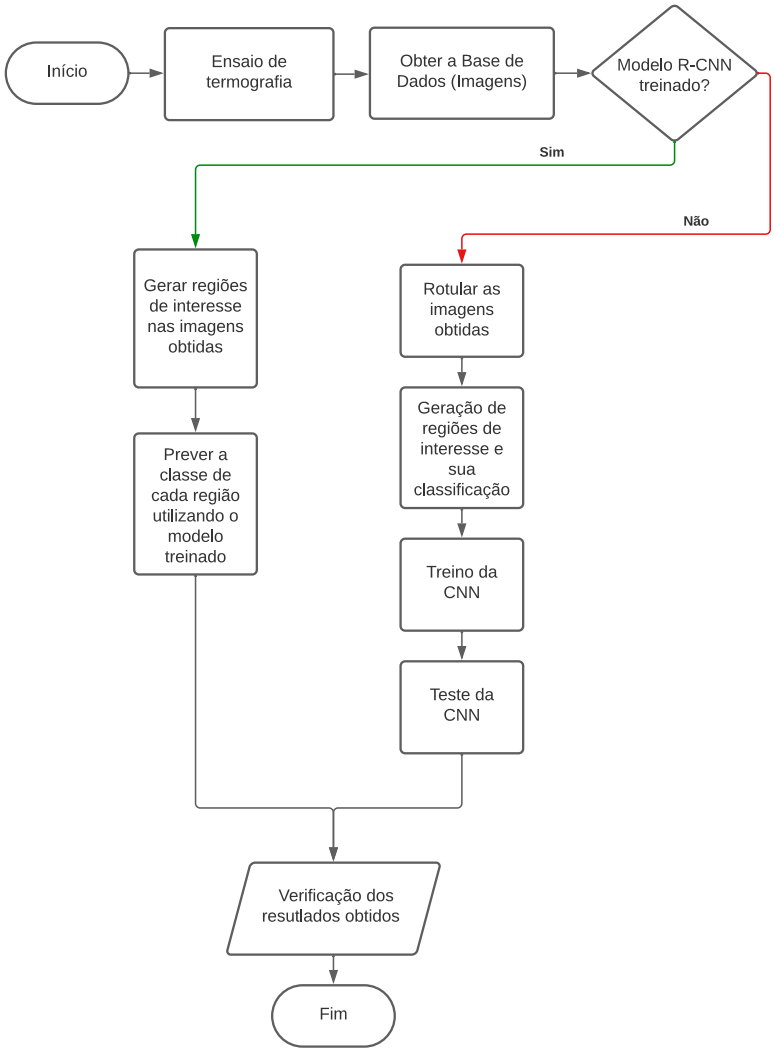


Figura 4.1 - Fluxograma relativo à arquitetura do sistema de identificação dos defeitos.

4.2 Automatização do ensaio de termografia

O ensaio de termografia foi utilizado para obter os dados necessários para a construção do modelo. A automatização do ensaio de termografia permite que seja muito fácil o seu controlo e monitorização. Como se encontra descrito por Gonçalves [9], o módulo DAQmx, apresentado anteriormente no subcapítulo 3.1, encontra-se corretamente conectado ao programa realizado através do uso da *Data Acquisition Toolbox* e permite, então, efetuar o controlo temporal do acender das lâmpadas durante o ensaio, permitindo, deste modo, realizar

uma excitação térmica da superfície da amostra mais vigiada. Para além disso, também é efetuada uma aquisição de vídeo que acontece ao mesmo tempo que a excitação térmica é iniciada. Para isso, é necessário comprovar que o hardware está devidamente conectado e atualizado e que seja reconhecido pelo programa desenvolvido e utilizar a *Image Acquisition Toolbox* para estabelecer a conexão entre a câmara termográfica e o programa. Todo este programa foi desenvolvido em MATLAB por Gonçalves [9] e foi utilizado para a aquisição de vídeo necessária. Na Figura 4.2 pode ser encontrado com mais detalhe os vários tempos representativos associados às operações executadas ao longo do código, sendo que vão aparecendo certas mensagens a informar se as operações começaram ou terminaram.

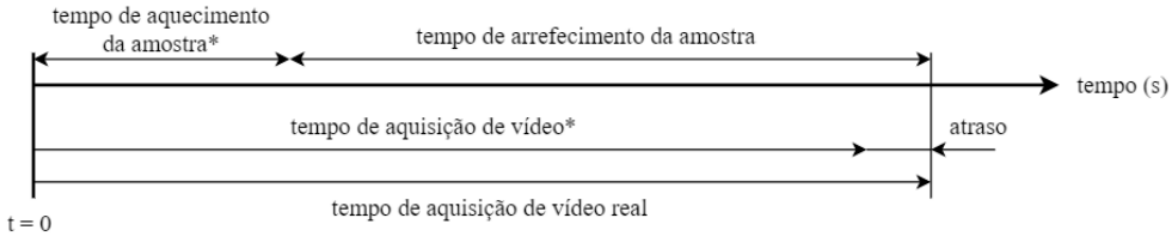


Figura 4.2 - Esquema representativo das operações que são realizadas pelo código elaborado num ensaio de termografia e a sua influência no tempo de aquisição de vídeo. Os valores com um asterisco representam os parâmetros definidos pelo utilizador [9].

4.3 Base de dados

Neste subcapítulo será apresentado com maior detalhe a criação e geração da base de dados que será utilizada para o desenvolvimento do modelo. Na Figura 4.3 apresenta-se o fluxograma detalhado dos vários passos que são realizados desde que se inicia um ensaio até obter os dados necessários para o modelo.

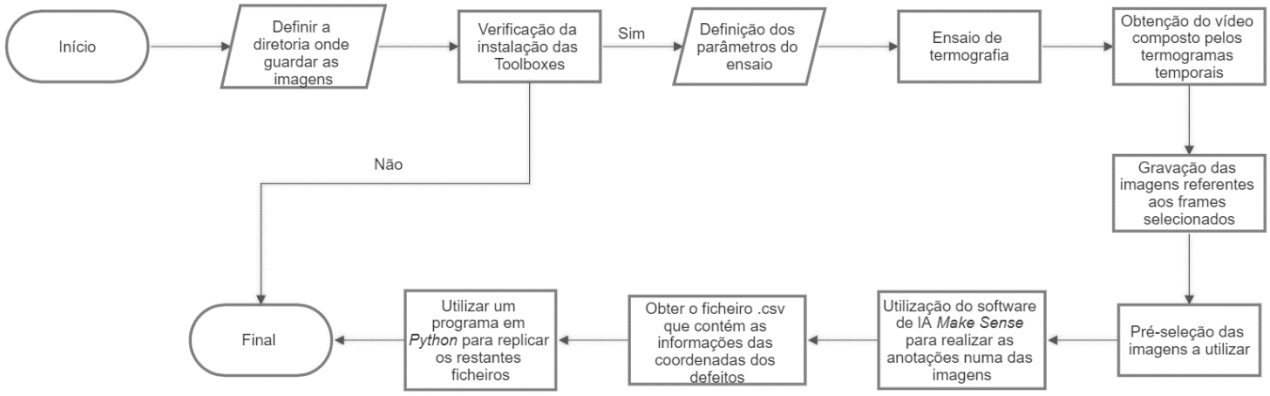


Figura 4.3 - Fluxograma relativo ao processo de geração da base de dados a partir de um ensaio.

4.3.1 Processo de obtenção das imagens

Todos os ensaios efetuados nos provetes tiveram a duração de 120 segundos, com uma excitação térmica de 20, sendo que foram realizados vários ensaios por cada um, na medida de encontrar os defeitos presentes na horizontal ou na vertical ou com maior ou menor intensidade em diferentes zonas dos mesmos, de modo a tentar obter o máximo número de casos possíveis para evitar o *overfitting* do modelo que foi desenvolvido, ou seja, que todas as imagens inseridas no mesmo tenham os defeitos na mesma posição.

Após a aquisição de vídeo, foram adicionadas linhas de código que permitem efetuar a gravação de *frames* resultantes de cada ensaio em questão como imagens. Visto que o vídeo resultante alcançava cerca de 1050 *frames*, optou-se por escolher 1 a cada 10, o que resulta em cerca de 105 imagens obtidas em cada ensaio. Estas imagens são guardadas num formato "Image_{i}.jpg", onde $i \in \mathbb{N}_0$, e numa diretoria a ser definida pelo utilizador no próprio código. Durante o decorrer do código vão surgindo mensagens que informam o utilizador se as imagens estão a ser gravadas corretamente, detetando se o nome já existia previamente e ajustando o mesmo se for o caso. O código utilizado para a automatização do ensaio de termografia e posterior gravação de imagens encontra-se presente no Apêndice A.

4.3.2 Rotulação dos dados

De modo a ser possível desenvolver um modelo de identificação de defeitos, é de vital importância fazer a rotulação dos dados obtidos, isto é, identificar os defeitos presentes em cada imagem fazendo anotações.

Numa fase prévia à rotulação, é fundamental fazer uma pré-seleção das imagens que serão utilizadas para o modelo. Como pode ser observado na Figura 4.2, existem duas fases que caraterizam cada ensaio: a fase de aquecimento e a fase de arrefecimento. Na fase inicial do aquecimento e na fase final do arrefecimento, normalmente, não é possível observar a posição nem a forma dos defeitos, devido a, na primeira situação, não ter ocorrido ainda um aquecimento que permita que os vazios sejam notados e, na segunda situação, se ter dado o escoamento térmico da superfície do provete, uniformizando o mesmo, pelo que as imagens referentes a essas situações foram descartadas.

Depois de concluído o passo anterior, realizaram-se as anotações pertinentes em cada uma das imagens. Para este efeito, utilizou-se, então, um software de inteligência artificial denominado *Make Sense* [32] que se encontra disponível na internet, que permite então adicionar um conjunto de imagens, criar *labels* e depois utilizar diferentes formas geométricas para inserir as anotações em cada imagem. Na Figura 4.4 ilustra-se uma imagem do software utilizado, onde se observa como é realizada a anotação dos defeitos de uma imagem, utilizando a *label* específica denominada de "Defeito".

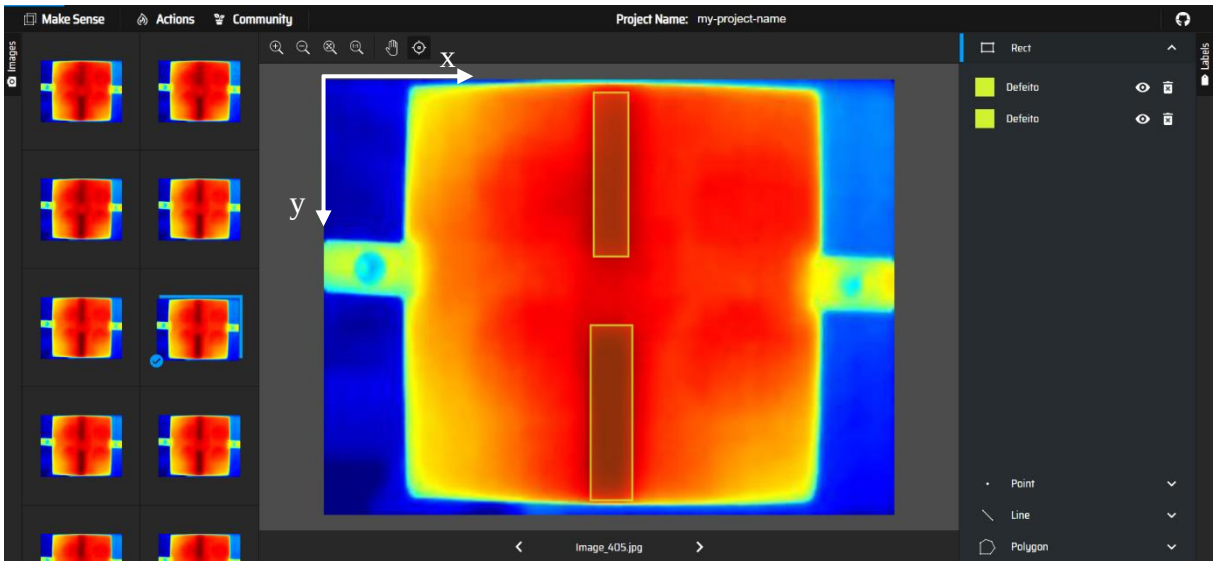


Figura 4.4 - Representação das anotações dos defeitos de uma imagem no software *Make Sense* [32].

Durante um ensaio de termografia o provete não é movido, pelo que os defeitos presentes estarão sempre na mesma posição, independentemente da evolução térmica observada pela câmara. Assim sendo, não é necessário que as anotações se realizem imagem a imagem, mas apenas numa delas. Do próprio software extrai-se um ficheiro com um formato ".csv" que contém o número de defeitos associadas à imagem em questão numa primeira linha e as coordenadas espaciais da posição dos mesmos nas linhas imediatamente a seguir. Neste caso, como os defeitos presentes nos provetes são retangulares, são indicadas 4 coordenadas para formar a *bounding box* (bb), ordenadas propositadamente da seguinte maneira: valor de x mínimo, valor de y mínimo, valor de x máximo e valor de y máximo. Os valores de coordenadas devolvidos encontram-se de acordo com o sistema de eixos apresentado na Figura 4.4 e estão de acordo com os limites de pixéis presentes na imagem (336 x 256). Um exemplo do ficheiro mencionado encontra-se presente na Tabela 4.1. Para as restantes imagens do mesmo ensaio, desenvolveu-se um programa, utilizando a linguagem de programação *Python*, que se encontra presente no Apêndice B.1 de modo a replicar os ficheiros, mantendo os nomes quer para as imagens quer para as anotações. De referir que estes ficheiros se encontram numa diretoria diferente das imagens dos ensaios.

Tabela 4.1 - Representação genérica de um ficheiro csv com a informação das coordenadas das *bounding boxes* (bbs) dos defeitos.

Número de defeitos (0 a n)			
Coordenada em x mínima do 1° defeito	Coordenada em y mínima do 1° defeito	Coordenada em x máxima do 1º defeito	Coordenada em y máxima do 1º defeito
...			
Coordenada em x mínima do defeito n	Coordenada em y mínima do defeito n	Coordenada em x máxima do defeito n	Coordenada em y máxima do defeito n

4.4 *Region Convolutional Neural Network* (R-CNN)

Para o desenvolvimento do modelo de deteção de defeitos, optou-se pela seleção de R-CNN. Este modelo em questão foi introduzido por Girshick et al. , [33] na tentativa de utilizar as altas capacidades das CNN para a seleção de regiões de interesse, de modo a ser possível a identificação dos vários objetos em questão. A sua escolha em detrimento de modelos como a

Fast R-CNN, a *Faster* R-CNN ou a *Mask* R-CNN deveu-se à sua menor complexidade e ao menor poder computacional necessário para a sua implementação.

A implementação do modelo foi realizada no software *Visual Studio Code*, utilizando *Python* como linguagem de programação. Para além disso, foram necessárias várias bibliotecas para o desenvolvimento do modelo, entre as quais *Tensorflow* [34], *Keras* [35], *Open-CV* [36], *Scikit-learn* [37], *Matplotlib* [38], *Pandas* [39] e *Numpy* [40], pelo que foram instaladas dentro de um ambiente criado no mesmo software, sendo este imprescindível para usar o código elaborado. O código elaborado foi baseado no modelo R-CNN apresentado por Thakur [41], onde foi desenvolvido um modelo R-CNN para a identificação de aviões.

A implementação deste tipo de modelo envolve que sejam seguidos uma série de passos. Primeiramente, é necessário que as imagens obtidas pelo ensaio de termografia passem por um algoritmo denominado de *Selective Search* (SS), de modo a serem geradas regiões de interesse. Numa segunda fase, utiliza-se uma métrica denominada de *Intersection Over Union* (IoU) que calcula a sobreposição entre as regiões criadas anteriormente e as rotulações realizadas manualmente no software *Make Sense* [32], atribuindo uma dada classe a cada região. Depois empregam-se essas mesmas regiões e respetivas rotulações para o treino de um modelo CNN. Optou-se pelo uso de um processo chamado de TL, onde o treino se realizou numa CNN pré-treinada. Por último, ocorre a fase do teste onde se usam as primeiras 2000 regiões de interesse geradas pelo algoritmo de SS e o modelo treinado para prever quais serão as zonas onde se encontram defeitos num conjunto de imagens. Na Figura 4.5 apresenta-se um resumo da sequência de etapas, sem muito detalhe, que compõe a R-CNN elaborada. Todo o código utilizado encontra-se apresentado no Apêndice B.

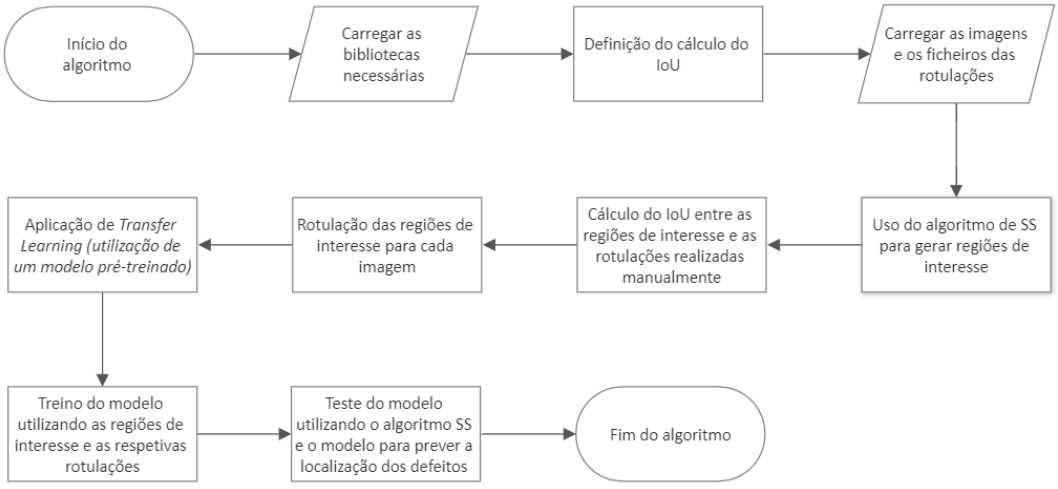


Figura 4.5 - Fluxograma referente às várias etapas da R-CNN elaborada.

4.4.1 Geração de regiões de interesse e sua classificação

Neste subcapítulo serão apresentados detalhes relativos ao processo de geração de regiões de interesse e posterior uso do algoritmo *Intersection Over Union* (IOU) para a classificação das regiões obtidas.

4.4.1.1 Selective Search (SS)

A geração de regiões de interesse é um dos pilares desta dissertação. Assim sendo, optou-se pelo uso de um algoritmo que combina a potencialidades de realizar uma pesquisa exaustiva e de fazer segmentação numa imagem e que foi introduzido por Uijlings et al. , [42] denominado de *Selective Search* (SS).

Este algoritmo assenta em 3 bases que são [42]:

- os objetos dentro da imagem poderão apresentar qualquer escala, sendo a escolha das regiões de interesse realizada a partir de um algoritmo hierárquico de semelhanças;
- a existência de uma certa diversificação na estratégia a adotar na hora de agrupar regiões que pode ser através da complementaridade espacial de cores, da combinação de semelhança de cores, textura, tamanho e preenchimento ou, por último, da variação das regiões iniciais complementares;
- a velocidade de computação do algoritmo para a identificação de possíveis locais de objetos é bastante elevada.

No que concerne à sua aplicação nesta etapa do desenvolvimento, foi extremamente útil para gerar regiões de interesse para cada imagem utilizada, regiões essas que são usadas posteriormente para serem rotuladas e se obterem os dados finais a serem empregues para o treino do modelo. Para a rotulação limitou-se o número de regiões a utilizar para um valor próximo das 2000, para garantir que as condições mencionadas no subcapítulo 4.4.1.2 quanto às limitações do número de casos positivos e negativos fossem cumpridos. Para a implementação deste método, foi essencial recorrer a uma função existente na biblioteca *OpenCV* [36], *cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()*, que contém as subfunções necessárias para a implementação do método e que, sem dúvida, simplificaram em muito a realização do algoritmo. Na Figura 4.6 observa-se um exemplo de obtenção de regiões de interesse utilizando o algoritmo de SS e na Figura 4.7 ilustra-se alguns exemplos das regiões de interesse selecionadas que irão ser classificadas utilizando o IoU, apresentado no subcapítulo 4.4.1.2.

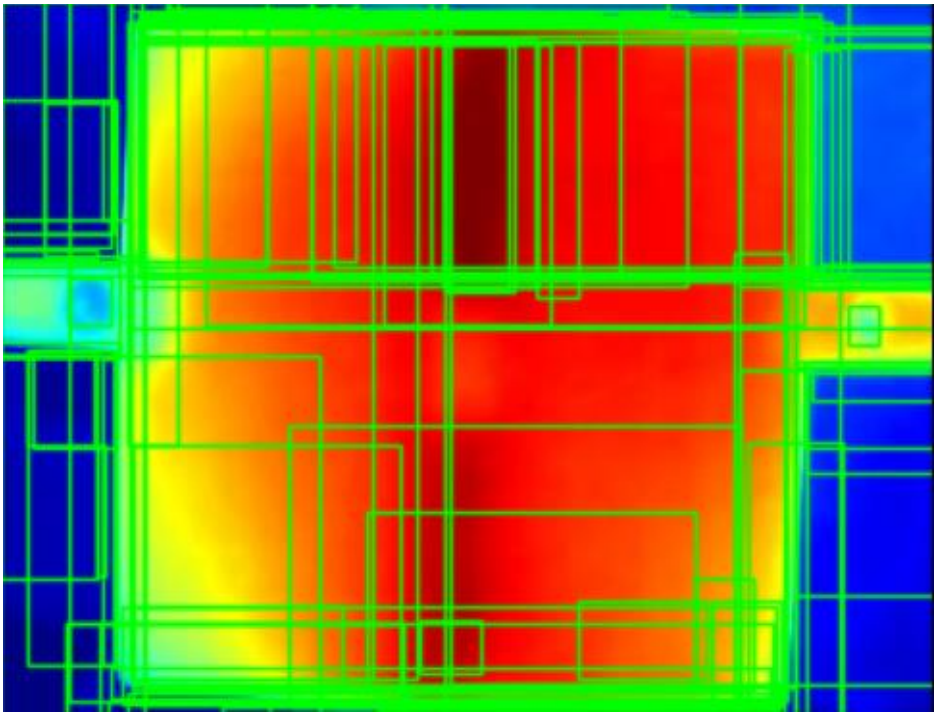


Figura 4.6 - Exemplo da geração de regiões de interesse numa imagem utilizando SS.

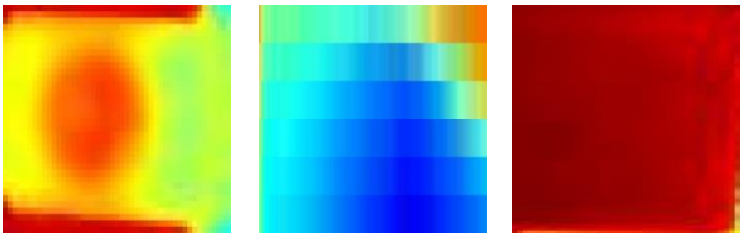


Figura 4.7 - Exemplos de regiões de interesse a serem classificadas.

4.4.1.2 *Intersection over Union* (IoU)

Para a implementação do modelo é crucial utilizar uma métrica denominada de IoU. Para aplicá-la, necessita-se de estar a trabalhar com imagens. e de ter 2 bbs em simultâneo sobre as quais a métrica irá trabalhar e avaliar a sua precisão.

O cálculo do IoU é obtido através do uso das 4 coordenadas espaciais que derivam da formação de cada bounding box (bb), de maneira geral, diferentes. As imagens a analisar são da superfície de um provete, pelo que o sistema de eixos a utilizar será Oxy, ou seja, 2 dimensões (2D). Assim sendo, 2 das coordenadas são para a posição mínima e máxima em x (x1 e x2) e as outras 2, analogamente, para a posição em y (y1 e y2). Tendo isto em conta, o cálculo da métrica é realizado através de uma operação de divisão entre a interseção das áreas

das 2 bbs e a sua união. Mais concretamente, a interseção entre as duas áreas é obtida através da utilização dos valores máximos de x1 e y1 e dos valores mínimos de x2 e y2, ou seja, é realizada uma operação aritmética de multiplicação entre a diferença de x2 e x1 e de y2 e y1, pela ordem apresentada e a união das duas áreas pode ser obtida através da operação aritmética da soma das duas áreas individualmente, que resulta da multiplicação dos lados de um retângulo, e da subtração da interseção das mesmas. O resultado terá de ser um valor entre 0 e 1, sendo 0 a inexistência de qualquer interseção entre as áreas e 1 a sua sobreposição total. O aspeto do cálculo da IoU está descrito na Equação 4.1.

$$\mathbf{IoU} = \frac{\text{Área de interseção}}{\text{Área de união}} = \frac{\big(\text{mín.}\,(\text{x2}) - \text{máx}(\text{x1})\big) \times (\text{mín.}\,(\text{y2}) - \text{máx}(\text{y1}))}{\text{Área bb 1} + \text{Área bb 2} - \text{Área de interseção}} \tag{4.1}$$

No âmbito desta dissertação, esta métrica foi utilizada com a finalidade de saber se uma dada região de interesse obtida de uma imagem é classificada como 1 (caso positivo), se tem defeito, ou como 0 (caso negativo), se o mesmo não se verifica, de modo a criar os dados necessários para o treino do modelo. Para isso, utilizaram-se as rotulações descritas no subcapítulo 4.3.2, as bbs verdadeiras, onde todo o defeito é abrangido, e as bbs obtidas pelo processo de SS mencionado no subcapítulo 4.4.1.1. Para separar os casos positivos dos negativos, admitiu-se que o valor de IoU fosse superior a 0,7 estar-se-ia perante um caso positivo e a imagem seria classificada com 1, enquanto se o valor fosse inferior a 0,3 corresponderia a um caso negativo e a imagem seria classificada com 0. As regiões que se situam entre os 0,3 e os 0,7 foram descartadas na tentativa de não induzir falsos positivos e falsos negativos no treino da CNN. Também foi imposto um limite de 20 ocorrências para cada caso, regulados com contadores e variáveis para controlar o ciclo. Na

Figura 4.8 ilustra-se a classificação das regiões de interesse formadas através da SS, sendo que a verde se encontram as regiões que obtiveram um valor de IoU superior a 0,7 e a branco as que obtiveram um valor de IoU inferior a 0,3. Também que é nesta fase que ocorre a transformação do *shape* da imagem onde se passa a ter imagens com 224 x 224 pixéis. A necessidade desta alteração prende-se pelo próprio *input shape* que a rede neuronal que foi selecionada e apresentada no subcapítulo 4.4.2 utiliza. Salientar ainda que cada região de interesse observada e classificada é a informação que vai ser usada aquando do treino e

validação da rede neuronal. O código correspondente à criação de dados está presente no Apêndice B.2.

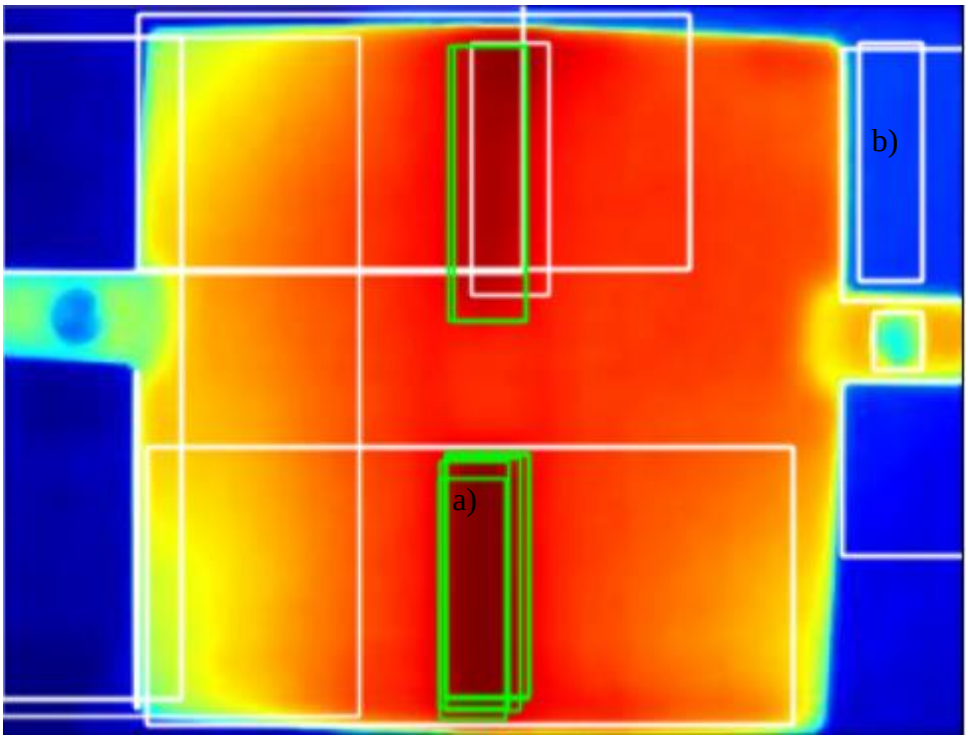


Figura 4.8 - Rotulação de cada região gerada por SS através do cálculo do IoU. a) Exemplo de regiões com um IoU superior a 0,7 (a verde); b) Exemplo de regiões com um IoU inferior a 0,3 (a branco).

4.4.2 *Transfer Learning* (TL)

Depois de obtida uma base de dados com informação suficiente em termos de quantidade e variedade, é necessário ter um modelo de DL para treinar e testar. Para isso, é preciso optar por construir um de raiz ou utilizar um modelo pré-treinado. Como a construção de um modelo de DL de raiz é um processo exaustivo e demorado, optou-se pela utilização de um modelo pré-treinado. A escolha realizada induz que o processo a ser utilizado se designe de TL.

TL é um método que utiliza um modelo base que vai ser treinado com uma dada base de dados, diferente da base de dados apresentada, de modo a resolver um certo problema. Depois é necessário reaproveitar as caraterísticas aprendidas e transferi-las para o modelo que queremos utilizar com uma base de dados própria e com o problema a resolver [43]. A capacidade que as CNNs apresentam na hora de aprenderem uma dada hierarquia de deteção de caraterísticas faz com que este método seja ágil e facilmente se adapte a um novo problema.

4.4.2.1 Escolha do tipo de CNN

O processo de escolha da CNN revelou-se um desafio na medida em que foi necessário conjugar vários fatores, entre eles, a precisão, o número de parâmetros da rede e o número de operações necessárias para classificar uma imagem. Depois de uma análise mais detalhada da Figura 4.9, chegou-se à conclusão de que a CNN escolhida seria a *ResNet-50*, que com a classificação da base de dados *ImageNet* obteve um valor máximo de precisão acima dos 75% e apresenta um número de operações para classificar uma imagem relativamente baixo, quando comparado com outra redes neuronais com precisões parecidas.

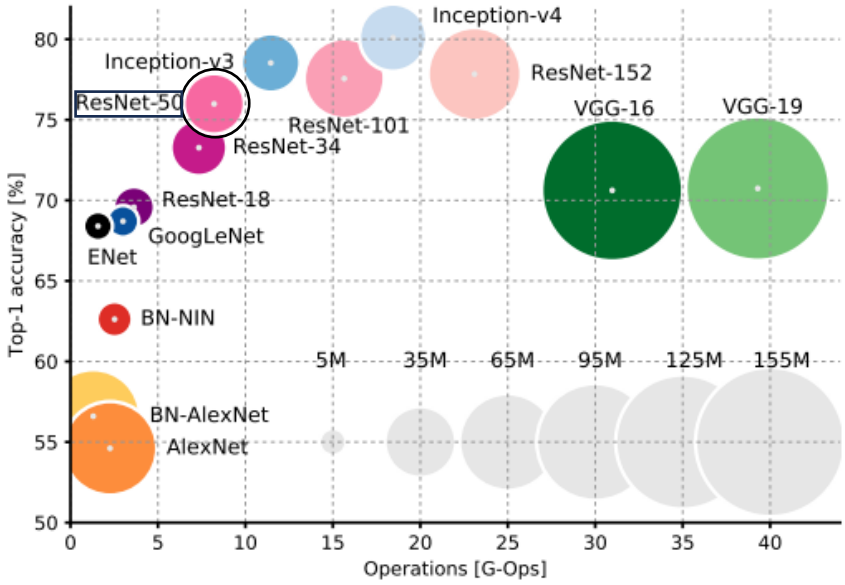


Figura 4.9 - Gráfico utilizado para a seleção do modelo pré-treinado. O gráfico apresenta a relação entre a precisão mais alta utilizando a classificação da base de dados da *ImageNet* e o número de operações necessárias para classificar uma imagem. O tamanho dos círculos é proporcional ao número de parâmetros da rede [44].

4.4.2.2 Arquitetura da *ResNet-50*

Para compreender melhor o modelo elaborado, será imprescindível conhecer melhor a rede neuronal escolhida. A *ResNet-50* é uma CNN que se insere no domínio das redes residuais (do Inglês *Residual Networks*) e que foi introduzida por He et al. , [45], devido à crescente dificuldade existente no treino de redes mais profundas, provocada pela saturação do valor da precisão que resulta na sua degradação.

Assim sendo, para solucionar o problema anterior, os autores introduziram os blocos de aprendizagem residuais (AR), que consistem em utilizar o valor de entrada de um bloco, x, adicionando-o ao resultado das operações realizadas nas camadas interiores, F(x), obtendo-se o valor de saída com a utilização da função de ativação *ReLU*, H(x). Isto possibilita a transição de 1 ou mais camadas de uma rede neuronal sem a realização das operações indicadas. Caso o valor de entrada e de saída tenham dimensões diferentes, é necessário que o bloco a adicionar tenha uma operação de convolução (1 x 1) para garantir a sua igualdade. Ao primeiro caso dá-se o nome de bloco de identidade e ao último o nome de bloco de convolução [45]. Na Figura 4.10a) ilustra-se um exemplo de adição de um bloco de identidade a um conjunto de operações e na Figura 4.10b) um exemplo de adição de um bloco de convolução.

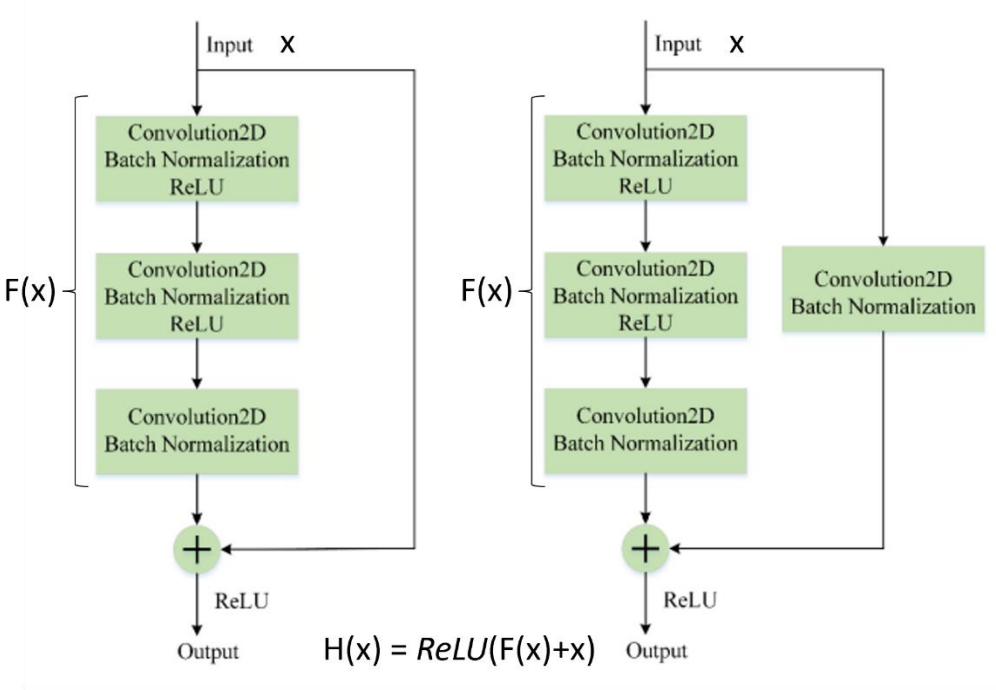


Figura 4.10 – Tipo de blocos a adicionar para uma rede neuronal residual. a) Bloco de identidade; b) Bloco de convolução [46]

Visto então o modo de funcionamento da AR, pode-se apresentar a arquitetura da rede neuronal a ser utilizada, *ResNet*-50. Neste caso, optou-se por utilizar uma versão melhorada da rede utilizada (*ResNet v2 50*), desenvolvido por He et al. [47], os mesmos autores, que

otimizaram ao máximo o uso do tipo AR apresentado na Figura 4.10a). Deste modo, podemos separar o nosso modelo em 7 partes, sendo a sua constituição a seguinte:

- introdução de uma imagem RGB 224 x 224 pixéis;
- operação de convolução 7 x 7 com modificação da dimensão para 64, com uma operação de *max pooling*;
- 3 ciclos de uma operação de convolução 1 x 1 e uma de 3 x 3, mantendo a dimensão e uma última 1 x 1, alterando a dimensão para 256;
- 4 ciclos de uma operação de convolução 1 x 1 e uma de 3 x 3, com dimensão 128 e uma última 1 x 1, alterando a dimensão para 512;
- 6 ciclos de uma operação de convolução 1 x 1 e uma de 3 x 3, com dimensão 256 e uma última 1 x 1, alterando a dimensão para 1024;
- 3 ciclos de uma operação de convolução 1 x 1 e uma de 3 x 3, com dimensão 512 e uma última 1 x 1, alterando a dimensão para 2048;
- classificação da imagem, tendo em conta que esta parte vai ser adaptada para a dissertação em questão, ao ter um classificador que contém 2 classes para determinar a existência ou não de defeito, sendo a classe 1 a existência e a classe 0 a não existência de defeito, utilizando para isso a função de ativação *softmax*.

A arquitetura da rede utilizada na presente dissertação encontra-se representada na Figura 4.11.

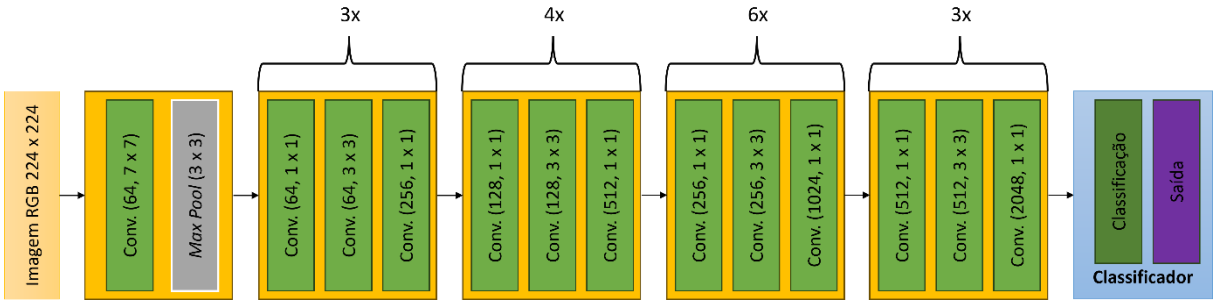


Figura 4.11 - Arquitetura da rede neuronal utilizada (Adaptado de [48])

Para tornar mais fácil a tarefa de utilização do modelo, recorreu-se ao *TensorFlow Hub*, uma biblioteca do *TensorFlow* [34], que contém, entre outra coisas, modelos pré-treinados que podem ser facilmente adaptados a um novo problema bem como a sua implementação em *Python*, utilizando a biblioteca *Keras* [35]. Deste modo, escolheu-se uma rede neuronal com o

aspeto mencionado anteriormente, ainda que apenas a parte referente aos vetores de caraterísticas da imagem, ou seja, sem o bloco referente à classificação, que havia sido previamente treinado por uma base de dados da *Google*, a *ImageNet* (ILSVRC-2012-CLS) [49].

4.4.3 Processo de treino da CNN

Neste subcapítulo serão apresentados os detalhes relativos ao processo de treino da rede neuronal para ser possível utilizá-lo na identificação dos defeitos caraterísticos presentes nas imagens da superfície dos provetes, entre os quais, o processo de separação de dados e *data augmentation*, a compilação do modelo e a definição dos parâmetros considerados necessários. Todo o código elaborado encontra-se no Apêndice B.3.

4.4.3.1 Separação dos dados e *Data Augmentation*

Numa fase ainda prévia ao treino, é necessário separar os dados em 3 partes: dados de treino, de validação e de teste. Para isso, recorreu-se a uma função específica da biblioteca *Scikit-learn* [37], *train_test_split*, que permite fazer essa separação de maneira automática, definindo apenas as percentagens necessárias para cada tipo. Referir que as imagens para teste são separadas inicialmente, de maneira manual, e colocadas numa diretoria própria. No caso da presente dissertação, definiu-se que os efeitos do treino se utilizariam 70% dos dados disponíveis, para a validação 10% e para o teste do modelo os 20% restantes.

Na tentativa de evitar ao máximo situações de *overfitting* do modelo, recorreu-se a uma função específica da biblioteca *Keras* [35], *ImageDataGenerator*, para gerar imagens que permitissem ao modelo visualizar os defeitos no máximo número de posições diferentes (*Data Augmentation)*. Sabendo que as imagens obtidas apenas continham orientados horizontalmente e verticalmente, utilizou-se a função anterior para realizar rotações com um máximo de 90º, permitindo também inverter a imagem horizontalmente e verticalmente, para se obter imagens que apresentassem diferentes orientações angulares. Aquando da geração das imagens, estas sofrem um processo de alteração de escala, chamado de normalização, onde se divide o valor de cada píxel por 255 (devido ao valor do RGB estar situado entre 0 e 254) para se obter valores entre 0 e 1, com o intuito de otimizar o treino a efetuar. Isto foi aplicado apenas aos dados de treino e aos dados de validação.

4.4.3.2 Funções de treino

Para o treino do modelo ser realizado, é preciso definir funções responsáveis pela sua evolução, entre os quais se inserem o tipo de optimizador, as perdas e a métrica a ser utilizada. A sua implementação foi realizada utilizando a biblioteca *Keras* [35].

Quanto ao optimizador, optou-se por utilizar *Adam*, introduzido por Kingma et al. [50], que é um algoritmo de primeira ordem baseado em gradientes para a otimização de funções estocásticas objetivas. A sua implementação é simples, a sua computação é eficiente e necessita pouco recursos de memória.

Para o cálculo das perdas durante o processo de treino, utilizou-se a função *categorical cross-entropy*. O seu uso é adequado à existência de 2 ou mais classes e a sua computação é realizada cruzando as perdas das rotulações e as predições do modelo. Em termos matemáticos, resulta do cálculo das médias entre a distribuição probabilística prevista e a atual para todas as classes inerentes ao problema e o seu valor considerado perfeito é 0. O cálculo das perdas segundo esta função encontra-se na Equação 4.2 [51], onde y_i representa o valor real para a classe i e o $\log \hat{y}_i$ representa o logaritmo natural da probabilidade da previsão efetuada para a classe i.

$$\textbf{Perdas} = -\sum_{i=1}^{\substack{\textbf{número} \\ \textbf{de classes}}} \boldsymbol{y_i \cdot \log \hat{y}_i} \tag{4.2}$$

Por último, a métrica a ser utilizada para análise vai ser a precisão. A sua utilização tem como base saber com quanta frequência as predições que o modelo faz são iguais às rotulações realizadas num certo conjunto de dados. O seu cálculo consiste só em utilizar uma operação aritmética de divisão entre os casos favoráveis e os casos totais.

4.4.3.3 Híper parâmetros de treino

O último passo prévio à realização do treino do modelo é a definição de alguns parâmetros necessários.

A definição do *batch size* serve para determinar o número de amostras de treino numa iteração, sendo que entre cada iteração ocorre a atualização dos parâmetros internos do modelo. Aquando da realização do treino do modelo reparou-se que o valores a utilizar para este híper parâmetro tinha de ser elevado, dado a demora temporal que um valor baixo neste parâmetro

resultava. Assim sendo, o valor escolhido para o processo de treino foi de 256 ou 512 e 512 para o processo de validação, dependendo do tempo total necessário para executar o treino da CNN. O valor escolhido vai influenciar o número de iterações por cada *epoch* que pode ser calculado sabendo o parâmetro definido anteriormente e número de amostras totais utilizadas no processo de treino (Equação 4.3).

$$Número\ de\ iterações\ por\ epoch = \frac{Números\ total\ de\ amostras}{Batch\ size} \tag{4.3}$$

Para além disso, também foi necessário definir o número de *epochs* do modelo, sendo este híper parâmetro representativo do número de vezes que os dados de treino serão utilizados para efeitos do treino da rede neuronal. Definiu-se como 100 o valor máximo a utilizar.

Ainda é necessário otimizar o que é considerado o híper parâmetro mais importante para se obter um bom desempenho do modelo tendo em vista a resolução do problema, o LR. Este gere a quantidade de alterações realizadas nos parâmetros durante cada etapa do processo [52]. O valor definido para este híper parâmetro foi de 10^{-5}, numa primeira fase, tendo sido realizado um processo de variação do mesmo, com os seus valores a iniciarem em 10^{-1} e 10^{-2} e a reduzirem-se para metades a cada 5 *epochs*.

4.4.3.4 *Callbacks* do treino

Para efetuar o controlo do treino, foram implementadas duas funções de modo a ser possível, monitorizar a evolução do treino.

Por um lado, utilizou-se a função *Early Stopping* [35] para permitir parar o treino da rede neuronal, de maneira automática, caso certas condições sejam verificadas. No caso em questão, caso não ocorram alterações durante 10 *epochs* no sentido de melhorar o valor das perdas da validação do modelo, o treino é parado de imediato.

Por outro lado, utilizou-se a função *Checkpoint* [35] que grava o melhor resultado tendo em conta a evolução de um dado parâmetro. No caso em questão, o parâmetro a ser monitorizado foram as perdas na validação, onde pretende obter o valor mínimo possível, pelo que caso o valor da *epoch* atual seja menor ao mínimo anteriormente encontrado, então, é gravado um ficheiro que contém os parâmetros internos da rede em questão nessa *epoch*.

4.4.4 Teste da CNN

Para comprovar que o treino foi bem realizado e possibilita a resolução do problema em questão, é necessário realizar testes que o verifiquem.

Numa fase inicial, são utilizadas várias métricas para avaliar o desempenho da CNN para as classes propostas. Tendo em conta que a avaliação do desempenho da classe para a existência de defeito será a mais importante, utilizaram-se as seguintes: precisão, *recall*, *F1-score* e exatidão. O cálculo das métricas é efetuado a partir de uma matriz de confusão que contempla a existência de falsos positivos (FP), falsos negativos (FN), positivos verdadeiros (TP) e negativos verdadeiros (TF), tendo em conta a classe verdadeira e a classe prevista pelo modelo. As equações associadas às métricas são as seguintes [53]:

$$Precisão = \frac{TP}{TP + FP} \tag{4.4}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.5}$$

$$F1 - score = 2 \times \frac{Precisão \times Recall}{Precisão + Recall} \tag{4.6}$$

$$Exatidão = \frac{TP + TN}{TP + FP + FN + TN} \tag{4.7}$$

Para terminar o processo de teste da CNN, é necessário recorrer ao método de obtenção de regiões de interesse apresentado no subcapítulo 4.4.1.1 e à CNN treinada previamente. Tendo em vista a obtenção de regiões de interesse, utilizam-se as imagens previamente separadas, sendo que, em cada uma delas, são retiradas e selecionadas as primeiras 2000 regiões para serem rotuladas e classificadas de acordo com as duas classes existentes. Salientar que é necessário que as regiões passem por um processo de normalização, ou seja, efetuar a mesma alteração de escala mencionada no subcapítulo 4.4.3.1 antes de a CNN realizar as predições da localização dos defeitos em cada imagem introduzida. Caso a região em questão apresente um valor para a classe 1, referente à existência de defeito, superior a 0,7, então, na imagem introduzida irá ser colocado um retângulo referente à região de interesse em questão. A verificação da precisão é feita através da visualização das predições dos defeitos realizados pelo modelo nas imagens inseridas. O código elaborado para a obtenção da matriz de confusão, cálculo das métricas e apresentação visual de um teste de inspeção é apresentado no

Apêndice B.4. Na Figura 4.12 apresenta-se um exemplo da identificação dos defeitos de uma imagem utilizando as predições do modelo.

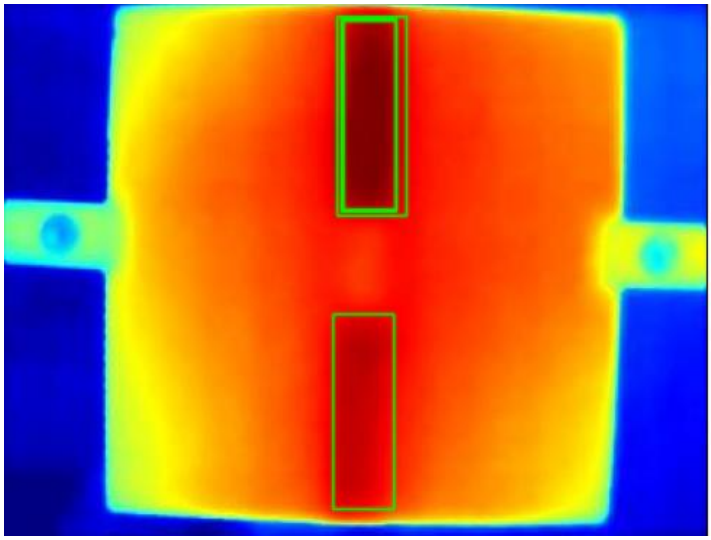


Figura 4.12 - Exemplo de identificação de defeitos realizada pelo modelo treinado (retângulos a verde são as predições realizadas).

5

APRESENTAÇÃO E DISCUSSÃO DOS RESULTADOS

5.1 Introdução

Neste capítulo são apresentados os principais resultados obtidos com o modelo R-CNN desenvolvido. Para isso, é fundamental que a escolha dos parâmetros seja realizada numa ótica da sua própria otimização. O principal parâmetro a analisar com maior atenção será o LR.

Para a implementação do modelo R-CNN, foi utilizada uma base de dados com 958 imagens da superfície dos vários provetes utilizados e as suas rotulações realizadas de acordo com o descrito no subcapítulo 4.3.2. Estas imagens foram utilizadas para a geração das regiões de interesse e sua posterior classificação que depois serão utilizadas para o treino, validação e teste da CNN. Um ponto importante a realçar é a necessidade de utilizar um conjunto de dados balanceados, isto é, introduzir o mesmo número de amostras para cada classe existente. Aquando da geração de regiões de interesse e sua classificação notou-se que existia uma maior facilidade em serem atingidas as condições em que existe ausência de defeitos (classe 0) do que a presença de defeitos (classe 1).

Deste modo, tornou-se imprescindível recorrer a métodos que permitissem o equilíbrio dos dados na fase de treino e validação e também na fase de testes da CNN. As escolhas dos métodos acabaram por ser o *undersampling* e o *oversampling*. No primeiro caso, os dados são adaptados ao número de amostras existentes da classe 1, ou seja, existe um corte na quantidade de amostras utilizados da classe 0. No último caso, existe uma replicação do número de amostras da classe 1 até se atingir o número de amostras da classe 0. Para além disso, utilizou-se a variação do LR, em forma de escada para avaliar a evolução e desempenho do modelo.

Para a obtenção dos resultados, utilizou-se a base de dados que continha 958 imagens mencionada anteriormente. De acordo com o definido no subcapítulo 4.4.3.1, isto implica que sejam utilizadas 671 imagens para o treino da CNN (70%), 96 para efeitos de validação (10%) e 191 para o teste (20%).

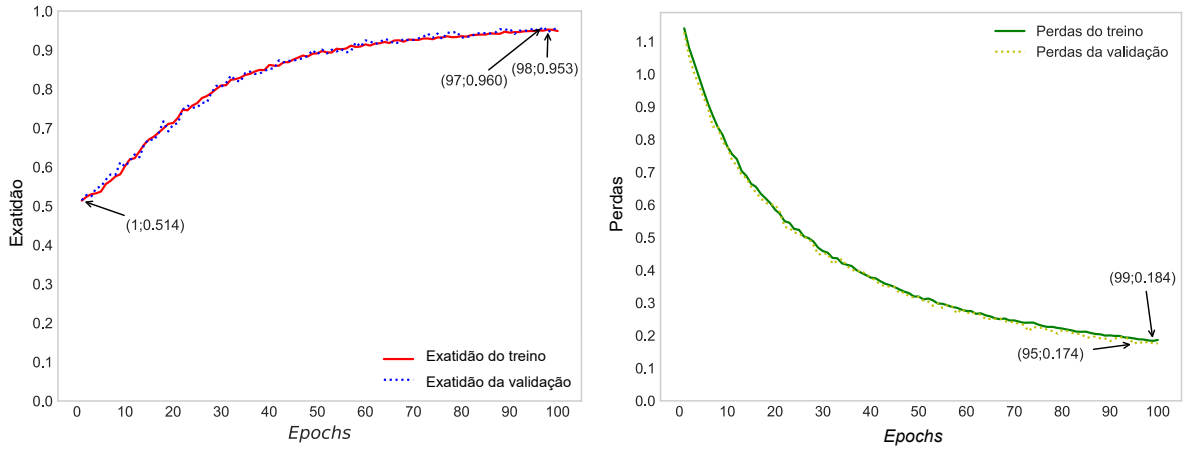
Por último, realizaram-se testes de inspeção *inline*, utilizando o vídeo do ensaio de termografia, com o intuito de avaliar o desempenho da CNN, através do cálculo da sua exatid

5.2 Resultados utilizando o método de *undersampling*

Depois de realizada a geração de regiões de interesse e sua classificação tal como é apresentada no subcapítulo 4.4.1, o número de amostras total ascendeu a 18539, sendo 15340 pertencentes à classe 0 (ausência de defeitos) e os restantes 3199 pertencentes à classe 1 (presença de defeitos). Devido ao problema de ocorrer o problema de balanceamento referido anteriormente, utilizou-se o método de *undersampling* e o número de amostras que foram utilizadas para o processo de treino e validação da CNN reduziram-se para 6398. A utilização da função de divisão de dados introduzida e detalhada no subcapítulo 4.4.3.1 dividiu o número de amostras em 5598 para o processo de treino e as restantes 800 para o processo de validação.

As funções de treino utilizadas foram as apresentadas no subcapítulo 4.4.3.2 e os *callbacks* foram os expostos no subcapítulo 4.4.3.4. Quanto aos parâmetros utilizados, o LR obedeceu ao valor mencionado no subcapítulo 4.4.3.3 e o *batch size* utilizado para o processo de treino foi de 256, enquanto para o processo de validação foi de 512. Na Figura 5.1 ilustram-se os gráficos referentes à evolução da exatidão e das perdas referentes ao processo de treino e de validação do modelo, em função do número de *epochs*.

Partindo da análise dos gráficos da Figura 5.1, é possível concluir que o processo de treino foi concluído na *epoch* 100, o limite imposto para o processo de treino da CNN, o que implica que ocorreu uma redução progressiva no valor das perdas da validação. Para além disso, observa-se uma evolução crescente do valor das exatidões do treino e da validação na Figura 5.1a), mais acentuada numa fase inicial, sendo mais ou menos constante a partir da *epoch* 50, verificando-se um valor máximo de exatidão do treino de 0,953 para a *epoch* 98 e um valor máximo na exatidão da validação de 0,960 na *epoch* 97. Quanto às perdas, a partir da Figura 5.1b) é possível concluir que estas tiveram um decaimento mais acentuado até às 50 *epochs*, mantendo-se constante a partir daí. Os valores mínimos atingidos para o treino foram de 0,184,



a) b)

Figura 5.1 - Gráfico da evolução da a) exatidão e das b) perdas em função do número de *epochs* relativos ao processo de treino e de validação (método de *undersampling*).

na *epoch* 99, para a validação e 0,174, na *epoch* 95. A estabilização dos valores da precisão e das perdas, para ambos os casos, é um indício de que o treino da CNN foi realizado corretamente e a aproximação das duas curvas, em ambos o caso, pode ser um indício de que não existe o perigo de se estar num caso de *overfitting*.

Realizado o processo de treino da CNN, é necessário testar a mesma e comprovar qual o seu desempenho da mesma perante novos casos. Para isso, utilizou-se então 191 imagens, que, após o processo de geração de regiões de interesse, resultou num total de 4928 amostras, sendo 3820 pertencentes à classe 0 e os restantes 1108 à classe 1. Mais uma vez, dado o desbalanceamento dos dados, utilizou-se o método de *undersampling*, com o qual se obteram um total de 2216 amostras. Utiliza-se, então, a CNN treinada previamente para prever qual será a classe para cada uma das amostras e compara-se essa previsão com o rotulado no processo de geração de regiões de interesse. Com isto, constrói-se uma matriz de confusão e calculam-se métricas como a precisão, o *recall* e o *F1-score* para cada classe e ainda se apresenta o valor da exatidão (do inglês *accuracy*). Os valores para tais métricas situam-se entre 0 e 1, sendo 1 o caso perfeito. Para o caso em questão, apresenta-se na Figura 5.2 a matriz de confusão referente ao teste realizado e na Tabela 5.1 os valores correspondentes às métricas utilizadas.

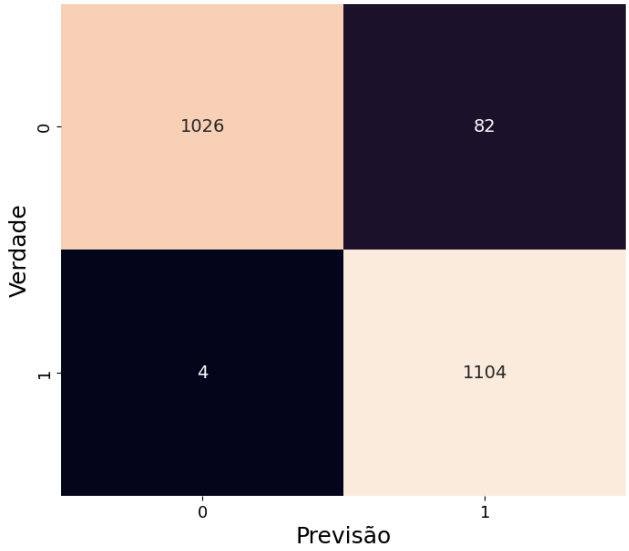


Figura 5.2 - Matriz de confusão para o teste realizado ao desempenho da CNN (método de *undersampling*).

Tabela 5.1 - Valores obtidos no teste para as diferentes métricas para cada classe e o valor da exatidão do teste realizado (método de *undersampling*).

	Métricas			
	Precisão	***Recall***	***F1-score***	**Exatidão**
Classe 0 (Ausência de defeito)	0,996	0,926	0,960	0,961
Classe 1 (Presença de defeito)	0,931	0,996	0,963	

Da análise dos valores obtidos, conclui-se que a avaliação do desempenho da CNN foi excelente, na medida em os valores alcançados foram superiores a 0,9 para todas as métricas analisadas. Salientar que a classe mais importante e que permite a resolução do problema é a classe 1, associada a deteção de defeitos. A exatidão associada ao teste da CNN também revelou um resultado bastante alto, ao atingir um valor de 0,961, o que veio comprovar que a CNN conseguiu aprender a identificar as zonas que apresentavam defeito e também as zonas em que isto não se verificava. Por último, toda esta análise converge para o facto de o balanceamento dos dados ser algo a ter em conta para a otimização dos resultados do treino e do teste de uma CNN.

Para completar o processo de teste do modelo e de acordo com o detalhado no subcapítulo 4.4.4, empregou-se várias imagens na tentativa de visualizar se era possível a identificação de

defeitos. As imagens em questão podiam ter os defeitos orientados verticalmente ou horizontalmente. Assim sendo, na Figura 5.3 apresentam-se alguns resultados do teste de 2 casos onde os defeitos foram localizados com sucesso.

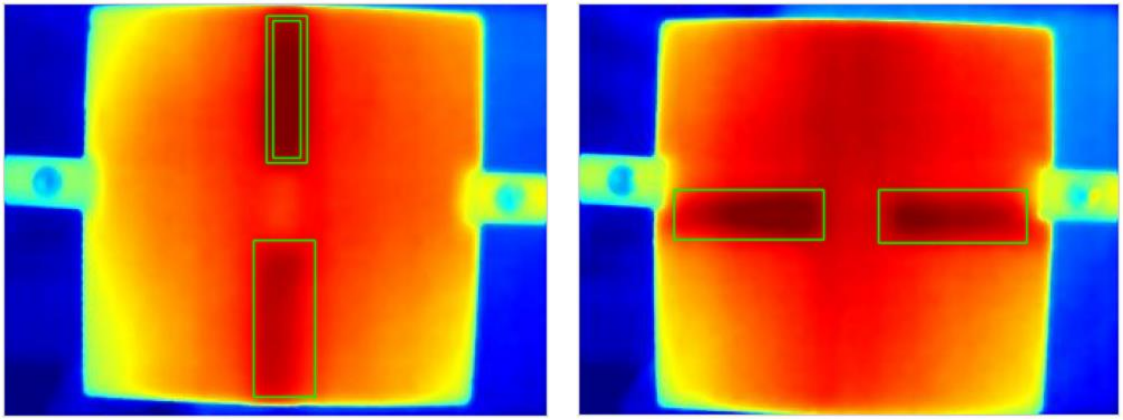


Figura 5.3 - Resultados da deteção de defeitos com sucesso utilizando a CNN treinada (método de *undersampling*).

Mesmo que os resultados obtidos tenham revelado ser um sucesso, existem casos onde tal não aconteceu de todo. Diferenças de contraste dentro da própria imagem ou semelhança de intensidades entre regiões com defeito e regiões sem defeito provocados pelo escoamento de calor dentro da própria superfície do provete levam a que ocorram algumas falhas na sua deteção. Na Figura 5.4 estão ilustrados dois casos onde o sistema de identificação não conseguiu detetar corretamente os dois defeitos existentes.

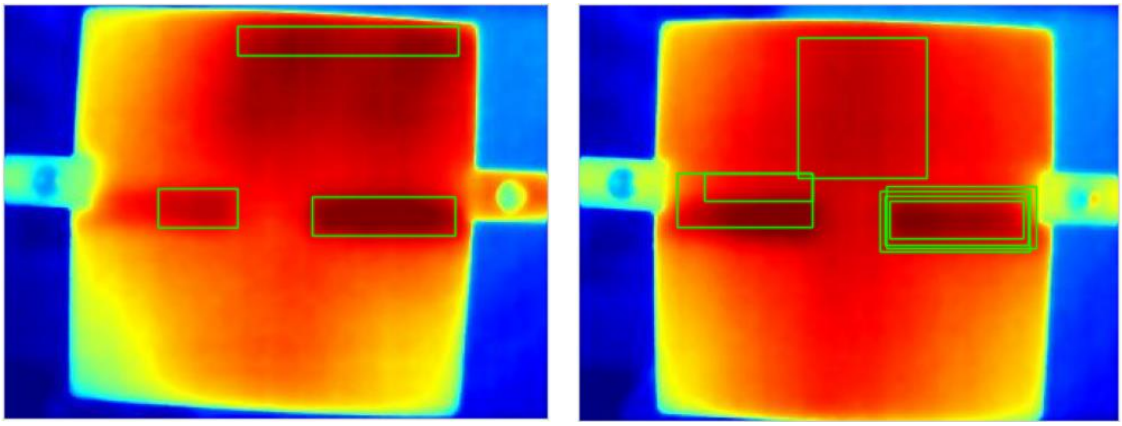


Figura 5.4 - Resultados de deteção de defeitos com falhas (método de *undersampling*).

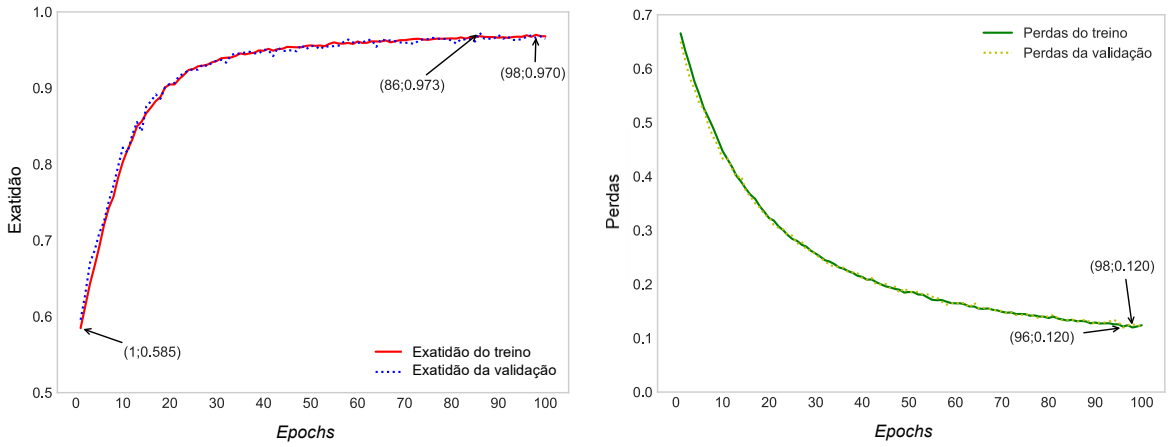
Perante os resultados observados na mesma figura, é possível concluir que a CNN treinada apresentou uma certa facilidade na deteção dos defeitos, o que demonstra a qualidade do treino efetuado. Ainda assim, é preciso afirmar que, em zonas onde o contraste se assemelhava aquelas onde os defeitos se encontravam localizadas, a rede neuronal sentiu uma certa dificuldade e reconheceu essas zonas como sendo defeitos. Este aspeto está relacionado com o escoamento de calor que se verifica ao longo da superfície do provete ao longo do tempo e que pode criar este tipo de confusões à identificação dos seus defeitos. Isto observa-se com maior detalhe nos retângulos a verdes (falsos positivos) localizados na parte superior da Figura 5.4.

5.3 Resultados utilizando o método de *oversampling*

O processo de geração de regiões resultou no mesmo número de amostras descritas no subcapítulo 5.2. Neste caso, optou-se por utilizar o método de *oversampling*. Devido a problemas relacionados com o processamento computacional do treino, foi necessário limitar o número de amostras a 15000, igualmente distribuídas pelas duas classes.

As funções de treino utilizadas foram as apresentadas no subcapítulo 4.4.3.2 e os *callbacks* foram os expostos no subcapítulo 4.4.3.4. Quanto aos parâmetros utilizados, o LR obedeceu ao valor mencionado no subcapítulo 4.4.3.3 e o *batch size* utilizado para o processo de treino foi aumentado para 512, enquanto para o processo de validação manteve-se igual ao processo descrito no caso anterior.

Na Figura 5.5 apresentam-se os gráficos referentes à evolução da exatidão e das perdas referentes ao processo de treino e de validação do modelo, em função do número de *epochs*, para o novo método aplicado.



a) b)

Figura 5.5 - Gráfico da evolução da a) exatidão e das b) perdas em função do número de *epochs* relativos ao processo de treino e de validação (método de *oversampling)*.

Partindo da análise dos gráficos da Figura 5.5, é possível concluir que o processo de treino foi concluído na *epoch* 100, tal como no caso anterior, o que implica que ocorreu uma redução progressiva no valor das perdas da validação. Para além disso, observa-se uma evolução crescente do valor das exatidões do treino e da validação na Figura 5.1a), onde se verifica uma certa estagnação no processo de treino a partir da *epoch* 40, verificando-se um valor máximo de exatidão do treino de 0,970 na *epoch* 98 e um valor máximo na exatidão da validação de 0,973 na *epoch* 86. Quanto às perdas, a partir da Figura 5.1b) é possível concluir que estas tiveram um decaimento mais acentuado até às 50 *epochs*, mantendo-se constante a partir daí ao final do treino. Os valores mínimos atingidos foram de 0,120, na *epoch* 9, para a validação e 0,120, na *epoch* 96. A estabilização dos valores da precisão e das perdas, para ambos os casos, é um indício de que o treino da CNN foi realizado corretamente e a aproximação das duas curvas, em ambos os casos, pode ser um indício de que não existe o perigo de se estar num caso de *overfitting*.

Novamente, é necessário realizar o teste da CNN, onde o processo de obtenção dos dados a utilizar foi o mesmo especificado para o método de *undersampling*, resultando num total de 2216 amostras. Utiliza-se, então, a CNN treinada previamente para prever qual será a classe para cada uma das amostras e compara-se essa previsão com o rotulado no processo de geração de regiões de interesse. Com isto, constrói-se, novamente, uma matriz de confusão e calculam-se métricas como a precisão, o *recall* e o *F1-score* para cada classe e ainda se apresenta o valor

da exatidão. Para o caso em questão, apresenta-se na Figura 5.6 a matriz de confusão referente ao teste realizado e na Tabela 5.2 os valores obtidos para cada classe nas diferentes métricas.

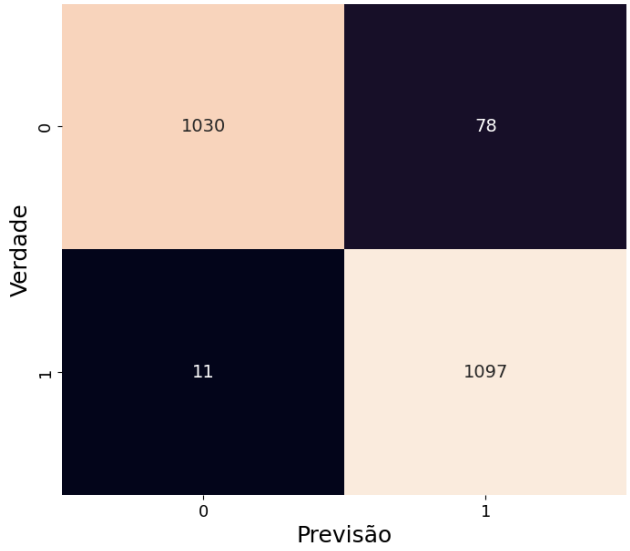


Figura 5.6 - Matriz de confusão para o teste realizado ao desempenho da CNN (método de *oversampling)*.

Tabela 5.2 - Valores obtidos no teste para as diferentes métricas para cada classe e o valor da exatidão do teste realizado (método de *oversampling)*.

	Métricas			
	Precisão	***Recall***	***F1-score***	**Exatidão**
Classe 0 (Ausência de defeito)	0,989	0,930	0,959	0,960
Classe 1 (Presença de defeito)	0,934	0,990	0,961	

A partir da Tabela 5.2, é possível concluir que as métricas alcançadas para a classe 1 demonstraram o elevado desempenho do treino realizado, onde alcançaram valores sempre superiores a 0,9. Também se pode concluir para a exatidão do teste se obteve um valor quase perfeito, com o mesmo a ser 0,96. Os valores obtidos revelam que a aprendizagem da CNN foi realizada com bastante sucesso e que posteriores testes poderão ter resultados bastante satisfatórios para a identificação dos defeitos.

Numa última fase de teste do modelo criado, introduziram-se várias imagens para, através do processo descrito no subcapítulo 4.4.4, ver se a CNN era capaz de prever a localização dos defeitos presentes nas mesmas. As imagens em questão podiam ter os defeitos orientados

verticalmente ou horizontalmente e permitiram realizar uma avaliação mais visual. Tendo isto em conta, ilustram-se na Figura 5.7 os resultados da deteção de defeitos com sucesso em duas imagens onde os retângulos a verde representam as regiões onde o modelo de DL identifica a presença de defeitos.

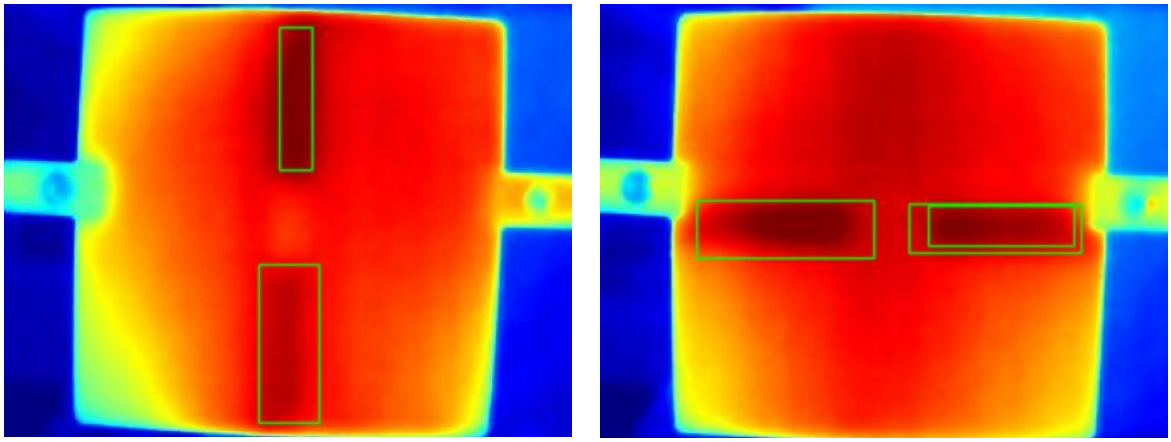


Figura 5.7 – Resultados da deteção de defeitos com sucesso utilizando a CNN treinada (método de *oversampling*).

Embora os resultados tenham, em grande parte, sido bem-sucedidos, houve ocasiões em que ocorreram falhas na deteção dos defeitos, como ilustrado na Figura 5.8, onde o sistema não conseguiu identificar corretamente os dois defeitos presentes.

Mais uma vez, conclui-se que a avaliação do modelo obteve excelentes desempenhos no que toca a deteção de defeitos nos provetes utilizados. Todavia, a existência de contrastes semelhantes em diferentes zonas das imagens nos termogramas adquiridos, devido ao escoamento de calor ocorrido durante a fase de arrefecimento realizados, pode ser um fator fundamental para explicar os erros ocorridos aquando da utilização do sistema para a deteção dos defeitos.

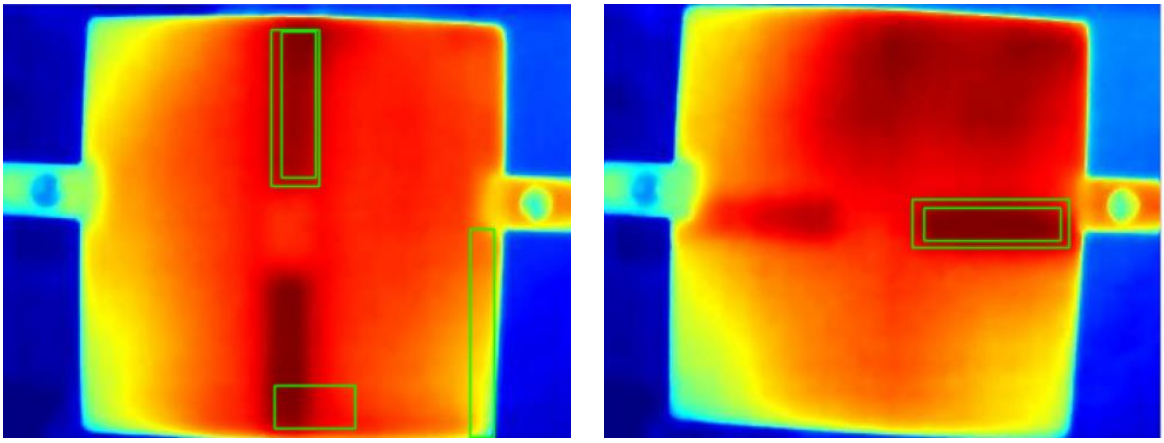
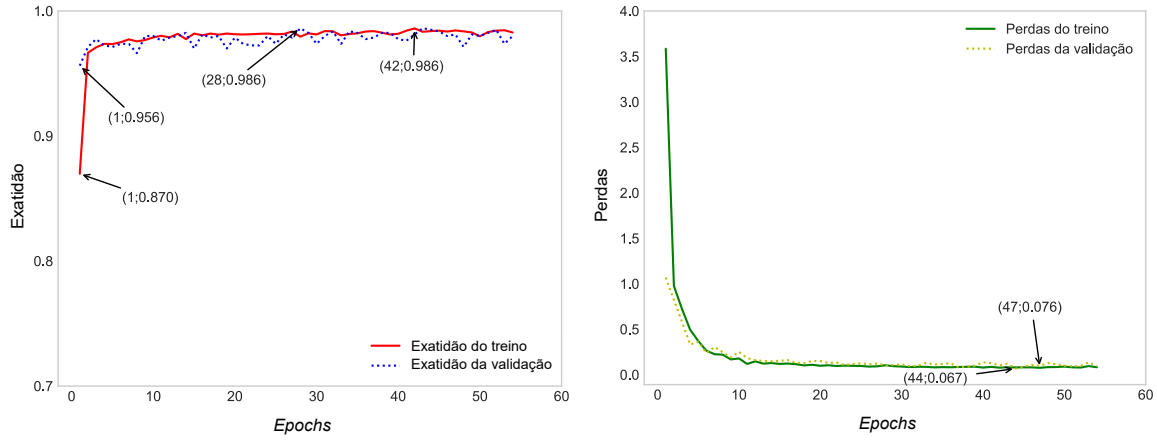


Figura 5.8 - Resultados da deteção de defeitos onde ocorreram falhas (método de *oversampling*).

5.4 Resultados com variação do *Learning Rate* (LR)

Neste subcapítulo apresenta-se o treino da CNN utilizando a variação do híper parâmetro LR ao longo do mesmo. Para isso, implementou-se um algoritmo que, partindo de um certo valor inicial para o LR, faz com que, a cada cinco *epochs*, o valor desça para metade do considerado atual no momento.

Tendo isto em conta, utilizou-se a mesma metodologia para o treino e teste implementado no subcapítulo 5.2, e, numa primeira fase, alterou-se o valor do LR de 10^{-5} para 10^{-1}. Efetuado o processo de treino e validação da CNN, ilustram-se na Figura 5.9 os gráficos referentes à evolução da exatidão e das perdas referentes ao processo de treino e de validação do modelo, em função do número de *epochs*.



a) b)

Figura 5.9 - Gráfico da evolução da a) exatidão e das b) perdas em função do número de *epochs* relativos ao processo de treino e de validação (Variação do LR com valor inicial de 10^{-1}).

Da análise dos gráficos, conclui-se que o facto do valor do LR iniciar-se alto traduz-se numa diminuição do número de *epochs* necessárias para a realização do treino, devido à convergência dos resultados ocorrer rapidamente, sem a necessidade de muitos ciclos de treino. Para além disso, verificam-se perdas maiores no início do processo, ocorrendo uma estabilização a partir da *epoch* 10. Os valores mínimos atingidos foram de 0,067 para as perdas de treino, na *epoch* 44, e 0,076 para as perdas de validação, na *epoch* 47. Todavia, os valores máximos atingidos superaram os obtidos nos subcapítulos 5.2 e 5.3, atingindo valores máximos de exatidão de 0,986 para as duas linhas observadas. O valor de LR atingido no final do treino foi de $9{,}765625 \times 10^{-5}$.

Na fase de teste da CNN, obteve-se a matriz de confusão correspondente e realizou-se o cálculo das métricas pertinentes. Na Figura 5.10 é apresentada a matriz de confusão correspondente ao teste da CNN e na Tabela 5.3 o cálculo das respetivas métricas.

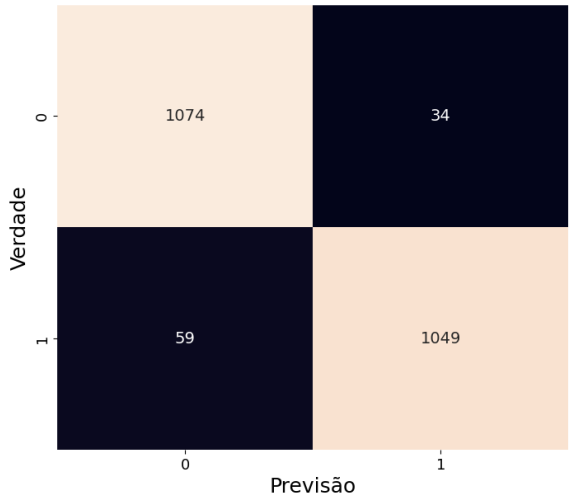


Figura 5.10 - Matriz de confusão para o teste realizado ao desempenho da CNN (LR inicial de 0,1).

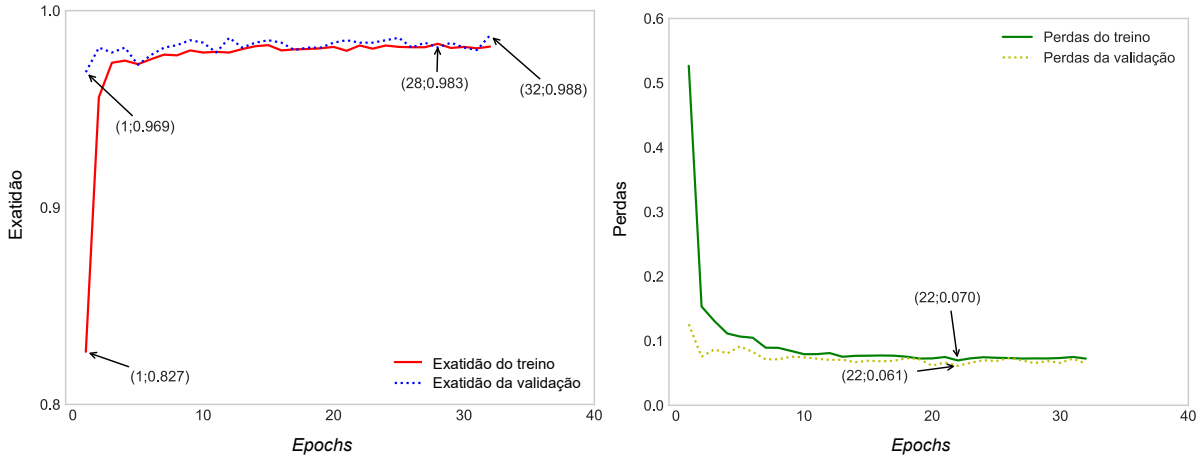
Os valores das métricas presentes na Tabela 5.3 revelam que ocorreu um aumento na precisão para a classe 1 relativamente aos valores obtidos para os métodos de *undersampling* e *oversampling*. Isto confirma que a utilização da variação do LR veio melhorar ainda mais a avaliação do desempenho da CNN no que se refere à classe principal a ser avaliada. Ainda assim, o desenvolvimento tão rápido e o número de *epochs* que a CNN teve sem sofrer grandes alterações podem resultar em fenómenos de *overfitting*. Na introdução de imagens para o

Tabela 5.3 - Valores obtidos no teste para as diferentes métricas para cada classe e o valor da exatidão do teste realizado (LR inicial de 0,1).

	Métricas			
	Precisão	***Recall***	***F1-score***	**Exatidão**
Classe 0 (Ausência de defeito)	0,948	0,969	0,959	0,958
Classe 1 (Presença de defeito)	0,969	0,947	0,958	

modelo localizar os defeitos, não existem grandes aspetos a considerar, apenas que o desempenho foi excelente na deteção precisa dos defeitos.

Uma última experiência foi realizada, na qual variou-se o valor inicial do LR para 10^{-2}, de modo a verificar quais seriam as principais alterações face aos resultados obtidos anteriormente neste subcapítulo. Deste modo, seguiu-se, novamente, a metodologia apresentada anteriormente no subcapítulo 5.4 e obtiveram-se na os novos gráficos referentes à evolução da exatidão e das perdas referentes ao processo de treino e de validação do modelo, em função do número de *epochs*.



a) b)

Figura 5.11 - Gráfico da evolução da a) exatidão e das b) perdas em função do número de *epochs* relativos ao processo de treino e de validação (Variação do LR com valor inicial de 10^{-2}).

Mais uma vez, a existência de um LR elevado provocou que a aprendizagem da CNN durante o processo de treino fosse realizada rapidamente, terminando o treino na *epoch* 32, tendo o valor convergido imediatamente, no ínicio do processo de treino, para um valor superior aos 0,8 e a tender rapidamente para valores acima dos 0,95. Para além disso, os valores máximos

de exatidão para o treino e para a validação chegaram perto dos 0,99 no final do processo, o que indica um desempenho excelente da CNN, apesar de este ter de ser verificado na fase de teste. No que concerne às perdas, os valores das perdas do treino e da validação tenderam para valores inferiores a 0,1, o que também revela um bom desempenho do modelo CNN.

Na fase de teste da CNN, obteve-se, então, a matriz de confusão correspondente e realizou-se novamente o cálculo das métricas. Na Figura 5.12 é apresentada a matriz de confusão correspondente ao teste da CNN e na Tabela 5.4 os valores das respetivas métricas, depois de efetuado o seu cálculo.

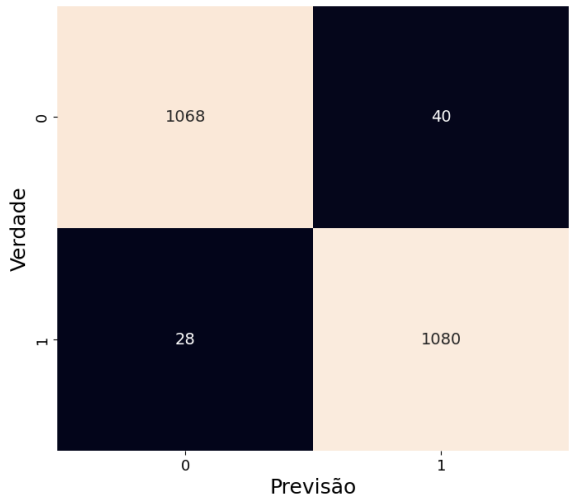


Figura 5.12 - Matriz de confusão para o teste realizado ao desempenho da CNN (LR inicial de 0,01).

Tabela 5.4 - Valores obtidos no teste para as diferentes métricas para cada classe e o valor da exatidão do teste realizado (LR inicial de 0,01).

	Métricas			
	Precisão	***Recall***	***F1-score***	**Exatidão**
Classe 0 (Ausência de defeito)	0,974	0,964	0,969	0,969
Classe 1 (Presença de defeito)	0,964	0,975	0,969	

Partindo dos resultados obtidos na Tabela 5.4, conclui-se que, com a modificação apresentada neste teste, atingiram-se os melhores resultados de todos os testes realizados no que se refere à precisão da localização da presença de defeito e a exatidão do teste da CNN. Os valores atingidos em todas as métricas superaram os 0,95, pelo que a avaliação do desempenho

do teste da CNN foi excelente, ou seja, a CNN em questão conseguiu aprender com excelência a identificar o tipo de defeitos pretendido.

5.5 Resultados de testes *inline*

Na dissertação realizada por Gonçalves [9], a deteção automática dos defeitos nos provetes compósitos de matriz polimérica realizou-se a partir de métodos convencionais de processamento de imagem. Os métodos aplicados basearam-se na aplicação de diferentes tipos de filtros às imagens captadas durante um teste de termografia na tentativa de isolar as regiões que continham defeitos, na divisão da imagem em várias regiões e na avaliação da incidência que certas regiões acabam por ter ao longo de todo o teste.

Na mesma dissertação, para realizar a inspeção de um provete, utilizaram-se *frames* retirados do vídeo adquirido pela câmara termográfica, ocorrendo um pré-processamento que resulta na eliminação de uma parte inicial e final do ensaio por se tratar de *frames* onde não é visível a presença ou ausência de defeitos. A identificação dos defeitos é realizada com base no acompanhamento de *blobs* (do inglês, *Binary Large Objects*), zonas da imagem mais claras ou mais escuras do que a sua vizinhança, ao longo dos *frames* para ser possível detetar se a zona se trata de ruído de imagem ou, então, de uma zona de defeito. Em termos de resultados, estes podem ser observados com maior detalhe no artigo desenvolvido no âmbito da conferência dos *Non Destructive Tests* (NDT) [54].

Na presente dissertação, o foco foi utilizar as potencialidades de um modelo de DL do tipo *Resnet v2 50* para resolver o mesmo problema. Deste modo, tornou-se necessário a elaboração de um algoritmo que permitisse a deteção dos defeitos utilizando como base a obtenção de um vídeo do ensaio realizado. Cada vídeo contém mais de 1000 *frames*, pelo que se optou por descartar os primeiros e últimos 25% dos mesmos, relativos a uma fase de aquecimento da superfície do provete e à estabilização do escoamento onde se visualiza uma superfície uniforme, respetivamente. Dos *frames* restantes, optou-se pelo uso de apenas 20, devido a uma limitação associada ao poder computacional, tendo sido escolhido 1 *frame* por cada 10 elegível até se atingir o valor pretendido. Para isso, recorreu-se ao algoritmo de SS, apresentado no subcapítulo 4.4.1.1, e à CNN treinada e testada no subcapítulo 5.2, de modo a ser possível localizar as regiões a serem utilizadas para a deteção e localização. As regiões consideradas foram aquelas em que o valor probabilístico dado pelo modelo era superior a 0,7.

Para auxiliar o algoritmo de SS na geração de regiões candidatas a conterem defeitos, foi estabelecido um limite máximo de 10000 (píxeis^2) para a área da região. Após o processamento de cada região pelo modelo de DL, o centroide foi calculado para aquelas classificadas como contendo defeitos, sendo os seus valores dependentes das coordenadas x e y, que nos clarifica a posição superior esquerda da região, e as coordenadas w e h, que nos dá a largura e altura da mesma, respetivamente. Assim sendo, o centroide de uma região é dado por $\bar{x}$ e $\bar{y}$ que são calculados da seguinte forma:

$$\bar{x} = x + \frac{w}{2} \tag{5.1}$$

$$\bar{y} = y + \frac{h}{2} \tag{5.2}$$

Ao longo do tempo, nas várias *frames* que compõem um teste de termografia, o centroide de um defeito raramente é detetado no mesmo píxel. De forma a superar esta dificuldade, utilizou-se um algoritmo de Aprendizado de Máquina da biblioteca Scikit-learn [37], chamado *Agglomerative Clustering*, para agrupar todos os centroides relacionados a um mesmo defeito em um único *cluster*. Para o controlo do número de *clusters* a serem utilizados, é necessária existir um compromisso entre a distância entre *clusters* distintos. Com a utilização deste algoritmo, é possível calcular o centroide de cada *cluster* através do cálculo da média das coordenada x e y dos centroides das regiões referentes a cada um. Para o cálculo da área dos defeitos encontrados, optou-se pela realização da média da largura, w, e da altura, h, das mesmas regiões referidas anteriormente. Estes valores são necessários para se conseguir apresentar a posição e localização dos defeitos detetados aquando dos testes de inspeção feitos. O algoritmo desenvolvido encontra-se presente no Apêndice B.5.

Foram realizados 20 testes de inspeção *inline*, onde, devido à limitação do número de provetes utilizados, recorreu-se a diferentes posições para se obter um maior número de vídeos para inspecionar, com o objetivo de se observar os dois defeitos presentes.

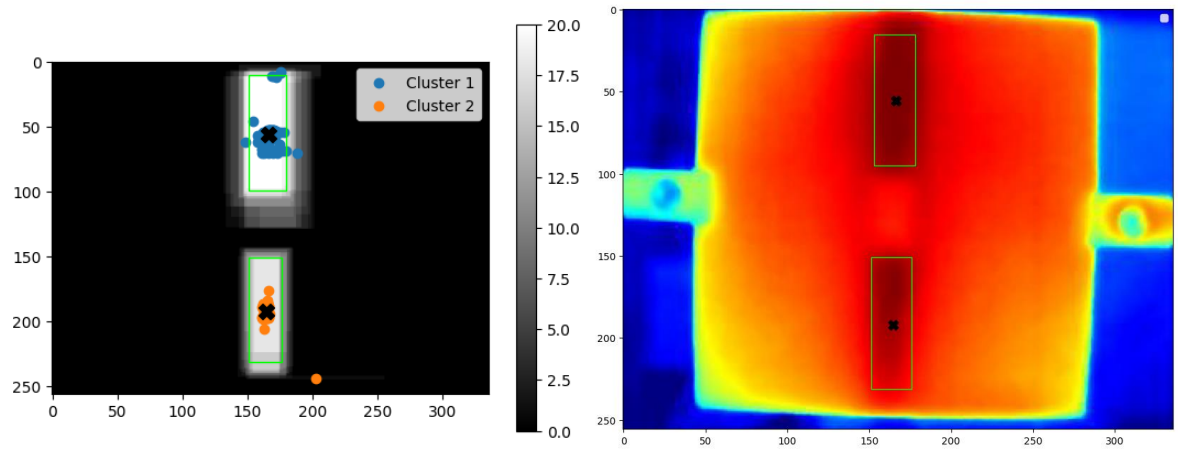
Os resultados destes testes são apresentados na forma de uma matriz de confusão, onde importa destacar que o valor referente aos TN não irá ser apresentado, visto que a inspeção é realizada com o objetivo de encontrar os defeitos nos provetes. Assim sendo, apresenta-se na Figura 5.13 a matriz de confusão referente aos testes realizados.

Figura 5.13 - Matriz de confusão referente aos testes de inspeção *inline* realizados.

Da análise da matriz, foi possível concluir que o valor da exatidão atingido foi de cerca de 0,844, ou seja, 84,4 %, o que revela elevado desempenho dos sistemas de inspeção.

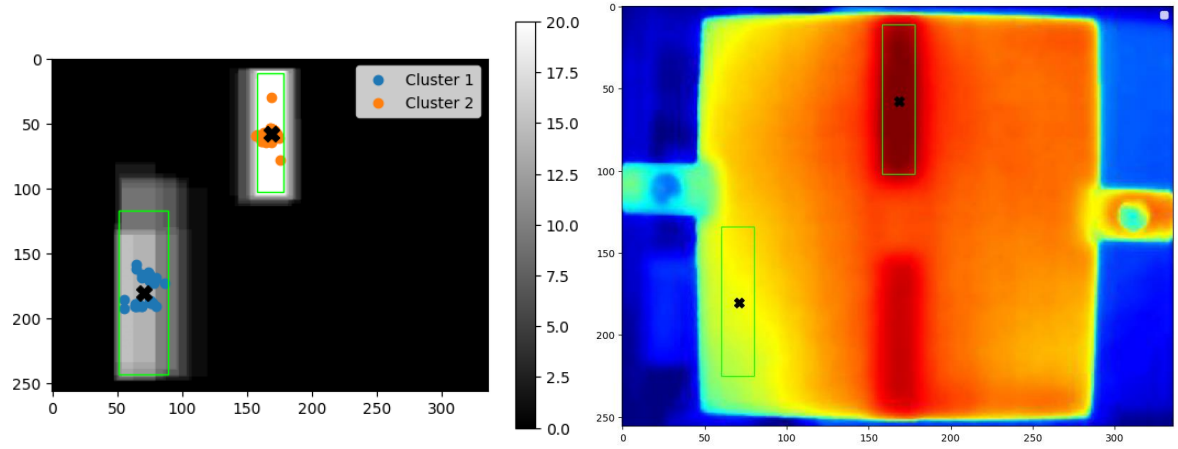
Para a apresentação dos resultados obtidos, optou-se pela utilização de uma matriz igual ao tamanho de uma imagem (336 x 256 píxeis), de modo a ser possível visualizar os centroides das regiões classificadas como defeituosas e o próprio centroide de cada *cluster*, bem como a área do defeito a ele associado. Esta matriz apresenta uma escala de cores cinzentas, onde é possível observar o número de incidências que dadas regiões tiveram ao longo do teste realizado, numa tentativa de confirmar os resultados obtidos. Também é apresentado um dos *frames* analisados com o resultado do teste de inspeção, onde se consegue observar o centro e a área dos defeitos encontrados. Para comprovar isto, ilustra-se na Figura 5.14 um caso onde os 2 defeitos foram identificados com sucesso, apresentando-se a sua matriz de incidência e um dos *frames* analisados com a localização e área prevista.

Em alguns casos não foi possível identificar os defeitos presente nos provetes, ocorrendo falhas, quer na deteção de um dos defeitos ou na deteção incorreta de uma zona como defeito. Este caso pode ser visualizado com maior detalhe na Figura 5.15, onde vemos os dois casos mencionados.



a) b)

Figura 5.14 - Resultados de um teste de inspeção bem-sucedido. a) Matriz de incidência em cada píxel e representação dos *clusters*. b) *Frame* com o centroide dos defeitos detetados, bem como as respetivas áreas.



a) b)

Figura 5.15 - Resultados de um teste de inspeção parcialmente bem-sucedido. a) Matriz de incidência em cada píxel e representação dos *clusters*. b) *Frame* com o centroide dos defeitos detetados, bem como as respetivas áreas.

Dos resultados obtidos, é possível concluir que os materiais de reforço existentes nos provetes não afetam a qualidade da inspeção e que a diferença de contraste por vezes observada em certa zonas dos *frames* analisados, como se pode observar na Figura 5.15b) ou a profundidade a que o defeito se encontra, relativamente à superfície da amostra analisada, pode dificultar o processo de inspeção. Ainda assim, é possível afirmar que existiu um bom desempenho por parte do algoritmo desenvolvido no processo de inspeção *inline*

6

CONCLUSÕES E TRABALHOS FUTUROS

6.1 Conclusões

Neste subcapítulo são apresentadas as conclusões relativas ao trabalho elaborado na presente dissertação.

É fundamental dizer que os principais objetivos delineados no início da presente dissertação foram alcançados com sucesso. A junção das potencialidades do ensaio de termografia com a aplicação de modelos baseados em aprendizagem profunda fez com que fosse possível desenvolver um sistema de visão termográfico baseado em aprendizagem profunda que permitisse a deteção automática de defeitos internos existentes em materiais compósitos de matriz polimérica.

As caraterísticas do problema em questão levaram a que o modelo a ser selecionada fosse uma R-CNN, onde os algoritmos de SS e IoU tornaram possível a geração de dados para efetuar o treino e o teste da rede neuronal, através das imagens obtidas em ensaios de termografia e posteriores rotulações das mesmas. Devido à complexidade de criação e ao elevado tempo que seria necessário despender neste processo, utilizou-se um processo de TF, onde se selecionou uma *ResNet v2 50* para ser a base para o treino, tendo em vista a resolução do problema.

Para garantir que os dados se encontravam balanceados, testaram-se dois métodos: *undersampling* e *oversampling*. Numa fase de avaliação da aplicação dos dois métodos, concluiu-se que a avaliação do desempenho da CNN foi elevada, ao serem obtidos valores para as métricas de avaliação, precisão, *recall* e *F1-score*, superiores a 0,9, para a classe referida como existência de defeito, o que é um indicador de que a CNN conseguiu aprender com detalhe e consegue prever a existência de defeitos. A exatidão também foi um fator a ter conta e foi alcançado um valor de 0,96 em ambos os casos. Assim sendo, concluiu-se que o modelo utilizado conseguia prever e localizar os defeitos com bastante precisão.

Para além dos métodos utilizados, implementou-se a variação do LR ao longo do treino da CNN, onde o mesmo descia para metade a cada 5 *epochs*. Os resultados obtidos para um valor inicial de 0,1 e 0,01 demonstraram que os ciclos de treino eram bastante mais reduzidos do que nos métodos de *undersampling* e de *oversampling* e que ocorria uma subida abrupta no valor da exatidão do treino e da validação para valores acima de 0,9 ainda numa fase muito precoce do treino, permitindo assim alcançar valores máximos próximo dos 0,99. Na fase de teste, os valores das métricas analisadas revelaram alcançar valores superiores a 0,95, o que revela um bom desempenho da CNN.

Por fim, a realização de testes de inspeção *inline* revelaram ter um valor de exatidão de 0,844, ou seja, 84,4 %. Os resultados permitiram concluir que os materiais de reforço existentes nos provetes não afetam a qualidade da inspeção e que a pouca diferença de contraste por vezes existente entre zonas com e sem defeito ou a maior profundidade dos defeitos podem ser um obstáculo que reduz a capacidade de o modelo realizar corretamente a deteção.

Em suma, os resultados obtidos revelaram-se bastante concretos e exatos, evidenciando a capacidade do modelo de DL desenvolvido apresenta para detetar defeitos internos em materiais compósitos de matriz polimérica.

6.2 Propostas de trabalhos futuros

O trabalho desenvolvido na presente dissertação apresenta espaços para melhorias e evoluções. A título sugestivo, apresentam-se algumas propostas para desenvolvimentos futuros:

1. Desenvolver um algoritmo que tenha em conta todas as *frames* recolhidas durante um ensaio de inspeção, incluindo aquelas em que o defeito não é visível a olho nu;
2. Utilizar modelos de DL mais complexos para a resolução do problema (*Fast* R-CNN e *Faster* R-CNN);
3. Gerar uma base de dados maior para efetuar o treino e o teste da CNN utilizada;
4. Aprimorar a capacidade de o modelo R-CNN identificar apenas uma vez cada defeito encontrado numa imagem (por exemplo, tendo em consideração o uso do centro de massa das regiões propostas);
5. Alargar a resolução do problema a outros tipos de defeito (humidade ou inclusões) e a defeitos com diferentes geometrias.

BIBLIOGRAFIA

[1] J. M. Lourenço, 'The NOVAthesis LATEX Template User's Manual'. Universidade NOVA, Lisboa, 2021. [Online]. Available: https://github.com/joaomlourenco/novathesis_word/raw/master/novathesis_word-FINAL-EN.pdf.

[2] S. Ghidoni, M. Antonello, L. Nanni, and E. Menegatti, 'A thermographic visual inspection system for crack detection in metal parts exploiting a robotic workcell', in *Robotics and Autonomous Systems*, Elsevier B.V., Dec. 2015, pp. 351–359. doi: 10.1016/j.robot.2015.07.020.

[3] R. M. Mahamood, E. T. Akinlabi, M. Shukla, and S. Pityana, 'Functionally Graded Material: An Overview', in *World Congress on Engineering* , London, 2012, pp. 1593–1597.

[4] M. A. Machado, M. I. Silva, A. P. Martins, M. S. Carvalho, and T. G. Santos, 'Double active transient thermography', *NDT and E International*, vol. 125, Jan. 2022, doi: 10.1016/j.ndteint.2021.102566.

[5] P. M. Bhatt *et al.*, 'Image-Based Surface Defect Detection Using Deep Learning: A Review', *Journal of Computing and Information Science in Engineering*, vol. 21, no. 4. American Society of Mechanical Engineers (ASME), Aug. 01, 2021. doi: 10.1115/1.4049535.

[6] I. Kuric *et al.*, 'Approach to Automated Visual Inspection of Objects Based on Artificial Intelligence', *Applied Sciences (Switzerland)*, vol. 12, no. 2, Jan. 2022, doi: 10.3390/app12020864.

[7] J. Yang, S. Li, Z. Wang, H. Dong, J. Wang, and S. Tang, 'Using deep learning to detect defects in manufacturing: A comprehensive survey and current challenges', *Materials*, vol. 13, no. 24. MDPI AG, pp. 1–23, Dec. 02, 2020. doi: 10.3390/ma13245755.

[8] C. Mera, M. Orozco-Alzate, J. Branch, and D. Mery, ‘Automatic visual inspection: An approach with multi-instance learning’, *Comput Ind*, vol. 83, pp. 46–54, Dec. 2016, doi: 10.1016/j.compind.2016.09.002.

[9] M. Gonçalves, ‘Desenvolvimento de Sistema de Visão Termográfico para Deteção de Defeitos em Materiais Compósitos de Matriz Polimérica’, Universidade NOVA de Lisboa, 2022.

[10] N. Bao, H. Kuang, A. Simeone, L. Zhu, and Y. Fan, ‘A machine vision-based automatic inspection system for power station coal bunkers maintenance’, in *Procedia CIRP*, Elsevier B.V., 2021, pp. 250–255. doi: 10.1016/j.procir.2021.10.040.

[11] S. K. Baduge *et al.*, ‘Artificial intelligence and smart vision for building and construction 4.0: Machine and deep learning methods and applications’, *Automation in Construction*, vol. 141. Elsevier B.V., Sep. 01, 2022. doi: 10.1016/j.autcon.2022.104440.

[12] Y. Li and J. Zhan, ‘SAIBench: Benchmarking AI for Science’, *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, vol. 2, no. 2, p. 100063, Apr. 2022, doi: 10.1016/j.tbench.2022.100063.

[13] X. Di and R. Shi, ‘A survey on autonomous vehicle control in the era of mixed-autonomy: From physics-based to AI-guided driving policy learning’, *Transp Res Part C Emerg Technol*, vol. 125, Apr. 2021, doi: 10.1016/j.trc.2021.103008.

[14] B. Prabhakar, R. K. Singh, and K. S. Yadav, ‘Artificial intelligence (AI) impacting diagnosis of glaucoma and understanding the regulatory aspects of AI-based software as medical device’, *Computerized Medical Imaging and Graphics*, vol. 87. Elsevier Ltd, Jan. 01, 2021. doi: 10.1016/j.compmedimag.2020.101818.

[15] M. Wakchaure, B. K. Patle, and A. K. Mahindrakar, ‘Application of AI techniques and robotics in agriculture: A review’, *Artificial Intelligence in the Life Sciences*, vol. 3, p. 100057, Dec. 2023, doi: 10.1016/j.ailsci.2023.100057.

[16] H. das Virgens, ‘Deteção de defeitos em asfalto utilizando deep learning’, Instituto Politécnico de Bragança, Bragança, 2021.

[17] S. Gaur, P. Kalani, and M. Mohan, ‘Harmonic-to-noise ratio as speech biomarker for fatigue: K-nearest neighbour machine learning algorithm’, *Med J Armed Forces India*, Jan. 2023, doi: 10.1016/J.MJAFI.2022.12.001.

[18] M. P. Behera, A. Sarangi, D. Mishra, and S. K. Sarangi, ‘A Hybrid Machine Learning algorithm for Heart and Liver Disease Prediction Using Modified Particle Swarm

Optimization with Support Vector Machine', *Procedia Comput Sci*, vol. 218, pp. 818–827, Jan. 2023, doi: 10.1016/J.PROCS.2023.01.062.

[19] M. Zhu *et al.*, 'A review of the application of machine learning in water quality evaluation', *Eco-Environment & Health*, vol. 1, no. 2, pp. 107–116, Jun. 2022, doi: 10.1016/j.eehl.2022.06.001.

[20] K. Alanne and S. Sierla, 'An overview of machine learning applications for smart buildings', *Sustainable Cities and Society*, vol. 76. Elsevier Ltd, Jan. 01, 2022. doi: 10.1016/j.scs.2021.103445.

[21] A. Nautiyal and A. K. Mishra, 'Machine Learning Application in Enhancing Drilling Performance', *Procedia Comput Sci*, vol. 218, pp. 877–886, 2023, doi: 10.1016/j.procs.2023.01.068.

[22] L. C. Souza, E. Celso, and A. de França, 'Detecção de Defeitos em Tecidos Através de Redes Neurais Convolucionais'. [Online]. Available: https://www.oreilly.com/library/view/python-advanced-guide/9

[23] M. Z. Alom *et al.*, 'A state-of-the-art survey on deep learning theory and architectures', *Electronics (Switzerland)*, vol. 8, no. 3. MDPI AG, Mar. 01, 2019. doi: 10.3390/electronics8030292.

[24] R. Girshick, J. Donahue, T. Darrell, and J. Malik, 'Region-Based Convolutional Networks for Accurate Object Detection and Segmentation', *IEEE Trans Pattern Anal Mach Intell*, vol. 38, no. 1, pp. 142–158, Jan. 2016, doi: 10.1109/TPAMI.2015.2437384.

[25] Y. Permanasari, B. N. Ruchjana, S. Hadi, and J. Rejito, 'Innovative Region Convolutional Neural Network Algorithm for Object Identification', *Journal of Open Innovation: Technology, Market, and Complexity*, vol. 8, no. 4. MDPI, Dec. 01, 2022. doi: 10.3390/joitmc8040182.

[26] V. Jacintha, S. Karthikeyan, and P. Sivaprakasam, 'Surface Flaw Detection of Plug Valve Material Using Infrared Thermography and Weighted Local Variation Pixel-Based Fuzzy Clustering Technique', *Advances in Materials Science and Engineering*, vol. 2022, 2022, doi: 10.1155/2022/7919532.

[27] A. Choudhary, D. Goyal, and S. S. Letha, 'Infrared Thermography-Based Fault Diagnosis of Induction Motor Bearings Using Machine Learning', *IEEE Sens J*, vol. 21, no. 2, pp. 1727–1734, Jan. 2021, doi: 10.1109/JSEN.2020.3015868.

[28] M. Antonello, S. Ghidoni, and E. Menegatti, ‘Autonomous robotic system for thermographic detection of defects in upper layers of carbon fiber reinforced polymers’, in *IEEE International Conference on Automation Science and Engineering*, IEEE Computer Society, Oct. 2015, pp. 634–639. doi: 10.1109/CoASE.2015.7294149.

[29] ‘IR Smart Camera IRSX Series’.

[30] H. Silva, ‘Simulação e validação experimental de Termografia Ativa para Compósitos de Matriz Polimérica reforçados com Fibras Contínuas Produzidos por Manufatura Aditiva’, Universidade NOVA de Lisboa, 2021.

[31] A. Gomes, ‘Ensaios Não Destrutivos para Impressão 3D de Materiais Compósitos de Matriz Polimérica’, Universidade NOVA de Lisboa, 2019.

[32] ‘Make Sense’. [Online]. Available: https://www.makesense.ai/

[33] R. Girshick, J. Donahue, T. Darrell, and J. Malik, ‘Rich feature hierarchies for accurate object detection and semantic segmentation’, Nov. 2013, [Online]. Available: http://arxiv.org/abs/1311.2524

[34] Google, ‘TensorFlow’. Accessed: Apr. 17, 2023. [Online]. Available: https://www.tensorflow.org/

[35] Keras, ‘Keras: Simple.Flexible.Powerful’. [Online]. Available: https://keras.io/

[36] OpenCV, ‘OpenCV’. Accessed: Apr. 17, 2023. [Online]. Available: https://opencv.org/

[37] Scikit-learn, ‘scikit-learn: machine learning in Python — scikit-learn 1.3.0 documentation’. [Online]. Available: https://scikit-learn.org/stable/

[38] Matplotlib, ‘Matplotlib — Visualization with Python’. Accessed: Apr. 17, 2023. [Online]. Available: https://matplotlib.org/

[39] Pandas, ‘Pandas - Python Data Analysis Library’. Accessed: Apr. 17, 2023. [Online]. Available: https://pandas.pydata.org/

[40] NumPy, ‘NumPy’. Accessed: Apr. 17, 2023. [Online]. Available: https://numpy.org/

[41] R. Thakur, ‘Step-by-Step R-CNN Implementation From Scratch In Python’. Oct. 18, 2019. Accessed: May 03, 2023. [Online]. Available: https://towardsdatascience.com/step-by-step-r-cnn-implementation-from-scratch-in-python-e97101ccde55

[42] J. R. R. Uijlings, K. E. A. Van De Sande, T. Gevers, and A. W. M. Smeulders, ‘Selective search for object recognition’, *Int J Comput Vis*, vol. 104, no. 2, pp. 154–171, Sep. 2013, doi: 10.1007/s11263-013-0620-5.

[43] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, 'How transferable are features in deep neural networks?', Nov. 2014, [Online]. Available: http://arxiv.org/abs/1411.1792

[44] C. Kawatsu, A. Zhao, and J. Crossman, 'Gesture Recognition for Robotic Control Using Deep Learning', Michigan, Aug. 2017. [Online]. Available: https://www.researchgate.net/publication/320084139

[45] K. He, X. Zhang, S. Ren, and J. Sun, 'Deep Residual Learning for Image Recognition', Dec. 2015, [Online]. Available: http://arxiv.org/abs/1512.03385

[46] D. Chen, F. Hu, G. Nian, and T. Yang, 'Deep residual learning for nonlinear regression', *Entropy*, vol. 22, no. 2, Feb. 2020, doi: 10.3390/e22020193.

[47] K. He, X. Zhang, S. Ren, and J. Sun, 'Identity Mappings in Deep Residual Networks', Mar. 2016, [Online]. Available: http://arxiv.org/abs/1603.05027

[48] D. Kvak, 'Leveraging Computer Vision Application in Visual Arts: A Case Study on the Use of Residual Neural Network to Classify and Analyze Baroque Paintings', Oct. 2022, doi: 10.20944/preprints202210.0448.v1.

[49] Google, 'Feature vectors of images with ResNet V2 50 trained on ImageNet (ILSVRC-2012-CLS).' Accessed: Jun. 15, 2023. [Online]. Available: https://tfhub.dev/google/imagenet/resnet_v2_50/feature_vector/5

[50] D. P. Kingma and J. Ba, 'Adam: A Method for Stochastic Optimization', Dec. 2014, [Online]. Available: http://arxiv.org/abs/1412.6980

[51] Pragati, 'How to choose cross-entropy loss function in Keras?' Accessed: Jun. 15, 2023. [Online]. Available: https://androidkt.com/choose-cross-entropy-loss-function-in-keras/

[52] J. Brownlee, 'How to Configure the Learning Rate When Training Deep Learning Neural Networks'. Accessed: Sep. 16, 2023. [Online]. Available: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

[53] Md. M. Islam *et al.*, 'A Deep Learning Model For Cotton Disease Prediction Using Fine-Tuning With Smart Web Application In Agriculture', *Intelligent Systems with Applications*, p. 200278, Nov. 2023, doi: 10.1016/j.iswa.2023.200278.

[54] M. S. Gonçalves, M. A. Machado, T. G. Santos, and N. Mendes, 'Automatic defect detection in fiber-reinforced polymer matrix composites using thermographic vision data', 2023. [Online]. Available: https://www.ndt.net/?id=28128

A

CÓDIGO MATLAB RELATIVO AO ENSAIO DE TERMOGRAFIA

```
% O código permite ao utilizador realizar um ensaio de termografia
% automaticamente, sendo o mesmo capaz de selecionar o tempo em que a
% câmara vai estar a adquirir vídeo e o tempo de aquecimento do
provete a analisar. Adquirido o vídeo, são guardadas imagens dos
termogramas obtidos,partindo dos frames constituintes do mesmo, numa
diretoria a ser definida pelo utilizador.
% Para garantir o correto funcionamento deste sistema é necessário
%que o utilizador tenha instaladas a Image Processing Toolbox, a
Image Acquisition Toolbox, a Data Acquisition Toolbox e a Computer
%Vision Toolbox.
% Caso o utilizador pretenda fazer a sua própria aquisição de imagem,
% é recomendado que seja utilizada a câmara termográfica IRS336 da
%marca Automatic Technology e se sigam as recomendações presentes no
%seu manual aquando da sua configuração. O presente algoritmo também
%permite que o utilizador processe ensaios que tenham sido realizados
%previamente através deste sistema.
% Para que o ensaio de termografia decorra automaticamente, é também
% necessário que seja garantida a conexão entre o MATLAB e um sistema
%de aquisição de dados, composto por uma fonte de tensão de corrente
% direta ALF2902M da ELC e relé da FINDER e um módulo DAQmx da
%National Instruments.
% O código referente à automatização do ensaio de termografia e toda
%a parte referente à gravação dos termogramas em formato de vídeo foi
%feita por Gonçalves [5] na dissertação realizada em 2022, sendo a
%parte final referente à gravação das imagens da autoria de Daniel
%Simões.

clear
clc
% Verificação da instalação da Image Processing Toolbox.
IPTinstalado = license('test', 'image_toolbox');
if ~IPTinstalado
       % O utilizador não tem a toolbox instalada.
       mensagem = sprintf(['Infelizmente, a Image Processing Toolbox
não está ' ...
        'instalada.']);
```

```
        return;
end

% Verificação da instalação da Image Acquisition Toolbox.
IATinstalado = license('test', 'Image_Acquisition_Toolbox');
if ~IATinstalado
        % O utilizador não tem a toolbox instalada.
        mensagem = sprintf(['Infelizmente, a Image Acquistion Toolbox
não está ' ...
        'instalada.']);
    return;
end

% Verificação da instalação da Data Acquisition Toolbox.
DATinstalado = license('test', 'Data_Acq_Toolbox');
if ~DATinstalado
        % O utilizador não tem a toolbox instalada.
        mensagem = sprintf(['Infelizmente, a Data Acquistion Toolbox
não está ' ...
        'instalada.']);
    return;
end

% Verificação da instalação da Computer Vision Toolbox.
CVTinstalado = license('test', 'Video_and_Image_Blockset');
if ~CVTinstalado
        % O utilizador não tem a toolbox instalada.
        mensagem = sprintf(['Infelizmente, a Computer Vision Toolbox
não está ' ...
        'instalada.']);
    return;
end

% Caixa de diálogo para definição do tempo de aquisição de vídeo (em
% segundos) que o utilizador pretende para o ensaio.
prompt = {'Defina o tempo de aquisição de vídeo em segundos:'};
dlgtitle = 'Tempo de aquisição de vídeo';
dims = [1 180];
definput = {'90'}; % Pré-definição de 90 segundos (1 minuto e meio)
f = inputdlg(prompt,dlgtitle,dims,definput);
tvid=str2double(f);
if (size(tvid)<1) | (tvid > 180) % Tempo máximo de 180 segundos (3
minutos)
    fprintf(['%s: Tempo de aquisição inválido.\n' ...
        'Tente novamente.\n'], datestr(now,'HH:MM:SS'))
    return;
end

% Caixa de diálogo para definição do tempo de aquecimento que o
utilizador
% pretende para o ensaio.
prompt = {'Defina o tempo de aquecimento da amostra:'};
dlgtitle = 'Tempo de aquecimento da amostra';
dims = [1 60];
```

```
definput = {'20'}; % Pré-definição de 20 segundos
f = inputdlg(prompt,dlgtitle,dims,definput);
taquec=str2double(f);
if (size(taquec)<1) | (taquec > 60) % Tempo máximo de 60 segundos
    fprintf(['%s: Tempo de aquecimento inválido.\n' ...
        'Tente novamente.\n'], datestr(now,'HH:MM:SS'))
    return;
end

% É dada a opção ao utilizador de realizar um novo ensaio ou de
processar
% um ensaio existente.
pergunta = questdlg(['Pretende realizar um novo ensaio ou processar
um '...
    'ensaio existente?'], ...
        'Opção', ...
        'Realizar ensaio','Processar ensaio','Processar ensaio');
switch pergunta
    case 'Realizar ensaio'
        resposta = 1;
    case 'Processar ensaio'
        resposta = 2;
end

%% 1. Automatização do ensaio de termografia
% Nesta secção dá-se a aquisição de vídeo através da câmara IRS336
% durante o período previamente definido pelo utilizador. O começo
%desta% aquisição dá-se em simultâneo com o começo do aquecimento da
%amostra para o ensaio de termografia que será analisado.
% Esta secção ocorre apenas se o utilizador pretender realizar um
%novo ensaio em vez de processar um existente.

if resposta==1
    imaqreset; % Atualização do hardware de aquisição de imagem.

    % Conexão entre o MATLAB e o sistema de aquisição de dados.
    d = daqlist("ni");
    d = daq("ni");
    addoutput(d,"Dev1","port0/line3","Digital")
    % Conexão da câmara termográfica IRS336.
    v = videoinput("gige", 1, "Mono16");

    % Configuração das propriedades da câmara para aquisição de
imagem.
    v.ReturnedColorspace = "grayscale";
    v.BayerSensorAlignment = "bggr";
    set(v,'PreviewFullBitDepth','on') % Permite pré-visualizar
imagens em uint16.
    src = getselectedsource(v);
    src.PacketSize = 1484;
    % Aquisição de vídeo durante o período de tempo especificado pelo
utilizador.
    numSeconds = tvid;
    v.FramesPerTrigger = Inf;
```

```
145
146     fprintf(['%s: Foi agora iniciada a aquisição de vídeo' ...
147         '.\nPor favor aguarde.\n'], datestr(now,'HH:MM:SS'))
148     tic % Começo da contagem do tempo real de aquisição de vídeo.
149     start(v); % Começo da aquisição de vídeo.
150
151     % Automatização do ensaio. As luzes responsáveis pelo aquecimento
152 da
153     % amostra serão acendidas e desligadas automaticamente de acordo
154 com o
155     % tempo de aquecimento previamente definido pelo utilizador.
156     write(d,1);
157     t = timer('TimerFcn', ['stat=false; disp(''Luzes desligadas.\nPor
158 favor ' ...
159         'aguarde enquanto se dá a aquisição de
160 vídeo.'')'],'StartDelay',taquec);
161     start(t)
162     stat=true;
163     while(stat==true)
164         disp('.')
165         pause(1)
166     end
167     write(d,0);
168
169     pause(numSeconds);
170     stop(v); % Fim da aquisição de vídeo.
171     tensaio=toc; % Fim da contagem do tempo real de aquisição de
172 vídeo.
173     fprintf(['%s: A aquisição de vídeo terminou' ...
174         '.\nPor favor aguarde.\n'], datestr(now,'HH:MM:SS'))
175
176     % Com o final do ensaio, os dados recolhidos são guardados com o
177 nome
178     % "amostra" no espaço de trabalho do MATLAB e na diretoria do
179 código.
180     amostra = getdata(v, v.FramesAvailable);
181     save('amostra.mat', 'amostra');
182     frames=size(amostra,4); % Recolha do número de frames do vídeo do
183 ensaio.
184     % Cálculo da taxa de aquisição de vídeo, frames por segundo,
185 tendo em conta
186     % o tempo real de aquisição de vídeo.
187     fps=frames/tensaio;
188     tarref=tensaio-taquec;
189 end
190 %% 2. Pré-processamento de imagem
191 % De modo a facilitar o processamento das imagens, esta secção
192 % garante o pré-processamento das imagens captadas, aprimorando a
193 %qualidade das mesmas. É também feita uma conversão da classe
194 %numérica de imagem de uint16 para uint8, otimizando o espaço ocupado
195 %pelas variáveis resultantes e o tempo de operação ao longo do
196 %algoritmo. O corpo do provete é reconhecido num dos frames captados,
197 %para que seja possível recortar a região de interesse nas imagens a
198 %processar.
```

```
% Caso o utilizador pretenda processar um ensaio previamente
%realizado,% o código permite ao utilizador escolher o ficheiro a
%analisar.
if resposta==2
    fprintf('%s: Escolha o ficheiro .mat a analisar.\n',
datestr(now,'HH:MM:SS'))
    uiopen;
    frames=size(amostra,4);
    fps=frames/tvid;
end
tic

fprintf(['%s: As imagens do vídeo adquirido serão agora pré-
processadas.' ...
    '\nPor favor aguarde.\n'], datestr(now,'HH:MM:SS'))

% Simplificação das imagens captadas e aprioramento da sua qualidade.
for t=1:frames
    % Ajuste de contraste. (imadjust)
    amostrauint16(:,:,1,t)=imadjust(amostra(:,:,1,t));
    % Redução do ruído da imagem. (medfilt2)
    amostrasRuido(:,:,:,t) = medfilt2(amostrauint16(:,:,:,t),[5 5]);
    % Aprimoramento da nitidez da imagem. (wiener2)
    amostraNitida(:,:,:,t)=wiener2(amostrasRuido(:,:,:,t),[5 5]);
    % Transformação da classe numérica da imagem de uint16 para
uint8. (im2uint8)
    amostrauint8(:,:,1,t)=im2uint8(amostraNitida(:,:,1,t));
    % Transformação do mapa de cores da imagem captada de preto e
branco
    % para cores. (ind2rgb)
    amostraRGB1(:,:,:,t)=ind2rgb(amostrauint8(:,:,:,t),jet);
end

fprintf('%s: O pré-processamento foi concluído com sucesso.\n',
datestr(now,'HH:MM:SS'))

implay(amostraRGB1); % Visualização do ensaio a cores.

number = 0; % Initialize the number counter outside the loop

for i = 1:10:size(amostraRGB1,4)
    % Read each frame
    Img = amostraRGB1(:,:,:,i);

    %To display all the frames
    %figure,imshow(Img);

    % Specify the directory path for saving the images
    directory = 'C:\Users\Dany\Desktop\Dissertação\Processamento de
ensaios exemplo';

    % Generate the image filename
    Img_name = ['Image_', num2str(number), '.jpg'];
    fullFilePath = fullfile(directory, Img_name);
```

```
253
254     % Check if the file already exists
255     while exist(fullFilePath, 'file') == 2
256         number = number + 1; % Increment the number
257         Img_name = ['Image_', num2str(number), '.jpg'];
258         fullFilePath = fullfile(directory, Img_name);
259     end
260
261     % Save the image
262     imwrite(Img, fullFilePath);
263
264     % Display a message indicating successful saving
265     disp(['Image ', num2str(i), ' saved as ', Img_name]);
266
267     number = number + 1; % Increment the number for the next
268 iteration
269 end
```

B

CÓDIGO *PYTHON* PARA O MODELO R-CNN

B.1 Replicação de ficheiro .csv

```python
import os       #Import the necessary libraries
import csv

def get_available_filename(base_filename, i):   #Function to detect if the filename is available
    output_file = f"{base_filename}_{i}.csv"
    if not os.path.exists(output_file):
        return output_file
    else:
        return get_available_filename(base_filename, i + 1)

def read_csv_and_save_multiple_times(input_file, base_output_filename, num_times):
    # Open the CSV file for reading
    with open(input_file, 'r') as csv_file:
        reader = csv.reader(csv_file)

        # Read the header row (optional if you want to include it in the output)
        header = next(reader)

        for i in range(num_times):
            output_file = get_available_filename(base_output_filename, i + 1)
            # Open the new output CSV file for writing
            with open(output_file, 'w', newline='') as out_csv_file:
                writer = csv.writer(out_csv_file)

                # Write the header to the output file (optional, depends on your preference)
                writer.writerow(header)

                # Write each row from the input file to the output file
                for row in reader:
                    writer.writerow(row)

```

```
            # Reset the reader to read the input file from the beginning in
the next iteration
            csv_file.seek(0)
            next(reader)  # Skip the header row for the next iteration

if __name__ == "__main__":
    input_csv_file = "Image_0.csv"  # Replace with the path to your input CSV
file
    base_output_filename = "Image"  # Replace with the desired base output
filename
    num_times = 52  # Number of times to save the CSV file

    read_csv_and_save_multiple_times(input_csv_file, base_output_filename,
num_times)
```

B.2 Processamento de dados

```
import os,cv2                           #Import the necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf

path = "Imagens"    #Directory where all the images are
annot = "Anotações" #Directory where all the respectives labels are

#The function get_iou calculates the Intersection over Union (IOU) between
two bounding boxes (rectangular regions)

def get_iou(bb1, bA):              #bb1 and bb2 are two dictionaries inputs
where are the 4 necessary coordinates of the bounding boxes
    assert bb1['x1'] < bb1['x2']    #Conditions that the program need to
guarantee to do the calculus correctly
    assert bb1['y1'] < bb1['y2']
    assert bb2['x1'] < bb2['x2']
    assert bb2['y1'] < bb2['y2']

    x_left = max(bb1['x1'], bb2['x1'])  #It takes the maximum or minimum to
know what's the overlapping region of the bounding boxes
    y_top = max(bb1['y1'], bb2['y1'])
    x_right = min(bb1['x2'], bb2['x2'])
    y_bottom = min(bb1['y2'], bb2['y2'])

    if x_right < x_left or y_bottom < y_top:    #If one of this specific
conditions is true, so there isn't an overlapping region between the two
bounding boxes and the return value will be 0
        return 0.0

    intersection_area = (x_right - x_left) * (y_bottom - y_top) #Simple
calculus of the intersection area
```

```
34
35      bb1_area = (bb1['x2'] - bb1['x1']) * (bb1['y2'] - bb1['y1']) # Calculus
36  of the area of the two bounding boxes individually
37      bb2_area = (bb2['x2'] - bb2['x1']) * (bb2['y2'] - bb2['y1'])
38
39      iou = intersection_area / float(bb1_area + bb2_area - intersection_area)
40  #Calculus of the Intersection over Union (IOU)
41      assert iou >= 0.0       #The value has to be between 0 and 1
42      assert iou <= 1.0
43      return iou
44
45  cv2.setUseOptimized(True);
46  ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
47
48  train_images = []        # List to store the preprocessed images
49  train_labels = []        # List to store labels corresponding to each image
50
51  for e,i in enumerate(os.listdir(annot)):
52      try:
53          filename = i.split(".")[0]+".jpg"        #Create the filename by
54  replacing the extensio with ".jpg"
55          print(e,filename)
56          image = cv2.imread(os.path.join(path,filename))      #Read the image
57  from the path
58          df = pd.read_csv(os.path.join(annot,i))              #Read the
59  annotation data from CSV
60          im_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)      #Convert image
61  to RGB
62          gtvalues=[]          # List to store ground truth bounding box
63  coordinates
64          for row in df.iterrows():
65              x1 = int(row[1][0].split(" ")[0])           #Extract the
66  coordinates of the bounding box from the annotation data
67              y1 = int(row[1][0].split(" ")[1])
68              x2 = int(row[1][0].split(" ")[2])
69              y2 = int(row[1][0].split(" ")[3])
70              gtvalues.append({"x1":x1,"x2":x2,"y1":y1,"y2":y2})      #Append
71  the values to gtvalues to use it later
72
73          #Initialize Selective Search for the image
74          ss.setBaseImage(image)
75          ss.switchToSelectiveSearchFast()    #Switch to Fast Selective Search
76          ssresults = ss.process()    #Results of the Selective Search process
77          imOut = im_rgb.copy()       #Copy of the RGB image to a new variable
78
79          counter = 0          #Counter for positive samples (when label = 1)
80          falsecounter = 0     #Counter for negative samples (when label = 0)
81          flag = 0             #Flag to control loop termination
82          pflag = 0            #Flag to indicate positive sample limit reached
83          nflag = 0            #Flag to indicate negative sample limit reached
84
85          for e,result in enumerate(ssresults):       #Loop through selective
86  search results
87              if e < 2000 and flag == 0:
```

```
                for gtval in gtvalues:
                    x,y,w,h = result
                    iou = get_iou(gtval,{"x1":x,"x2":x+w,"y1":y,"y2":y+h})
#Calculate Intersection Over Union (IOU)
                    #If IOU > 0.7 => Positive samples
                    if counter < 20:
                        if iou > 0.70:
                            roi = imOut[y:y+h, x:x+w]  # Extract the ROI
from the RGB image in imOut
                            resized = cv2.resize(roi, (224,224),
interpolation = cv2.INTER_AREA)    #Resized the image for what's matter
                            train_images.append(resized)    #Append the
image to train_images list
                            train_labels.append(1)          #Append the
respective label to train_labels list
                            counter += 1
                    else :
                        pflag =1        #The positive samples limit are
reached

                    #If IOU < 0.3 => Negative samples
                    if falsecounter < 20:
                        if iou < 0.30:
                            roi = imOut[y:y+h, x:x+w]  # Extract the ROI
from the RGB image in imOut
                            resized = cv2.resize(roi,(224,224),
interpolation = cv2.INTER_AREA)     #Resized the image for what's matter
                            train_images.append(resized)    #Append the
image to train_images list
                            train_labels.append(0)          #Append the
respective label to train_labels list
                            falsecounter += 1
                    else :
                        nflag = 1       #The negative samples limit are
reached

                #Terminate the loop if both positive and negative samples
limits are reached
                if pflag == 1 and nflag == 1:
                    print("inside")
                    flag = 1
    except Exception as e:
        print(e)
        print("error in "+filename)
        continue

# Convert lists to numpy arrays
X_new = np.array(train_images)
y_new = np.array(train_labels)

train_save_dir = "Training data"
label_save_dir = "Training label"

# Save the training images
```

```
for i, image in enumerate(X_new):
    image_filename = f"Image_{i}.jpg"  # Generate a unique filename for the image
    while os.path.exists(os.path.join(train_save_dir, image_filename)):
        i += 1
        image_filename = f"Image_{i}.jpg"
    image_path = os.path.join(train_save_dir, image_filename)
    cv2.imwrite(image_path, image)

# Save the respective labels of each image
for i,label in enumerate(y_new):
    label_data = [{"Label": label}]  # Create a dictionary with the label
    label_filename = f"Image_{i}.csv"  # Generate a unique filename for the label
    # Check if the file already exists, if so, increment i until a unique filename is found
    while os.path.exists(os.path.join(label_save_dir, label_filename)):
        i += 1
        label_filename = f"Image_{i}.csv"

    # Create the full path for the label file
    label_path = os.path.join(label_save_dir, label_filename)

    # Create a DataFrame from the label data
    label_df = pd.DataFrame(label_data)

    # Save the DataFrame as a CSV file
    label_df.to_csv(label_path, index=False)
    print(f"CSV file '{label_filename}' saved successfully.")


```

B.3 Treino da CNN

```
import os,cv2
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

import tensorflow as tf
import tensorflow_hub as hub

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam

train_images = "Training data"  #Paths where the images and labels are
train_labels = "Training label"

```

```
X = []
y= []
#Load the images and the respective labels to be used in the model
for e,i in enumerate(os.listdir(train_labels)):
    filename = i.split(".")[0]+".jpg"
    print(e,filename)
    image = cv2.imread(os.path.join(train_images,filename))
    df = pd.read_csv(os.path.join(train_labels,i))
    # Extract the label value from the second line of the CSV file
    label = df.iloc[0,0]
    # Append the image to X_train
    X.append(image)
    # Append the label to y_train
    y.append(label)

# Convert lists to numpy arrays
X_train = np.array(X)
y_train = np.array(y)

#Oversampling Method (run it if it's the case)
# Assuming you have two NumPy arrays: X_train (images) and y_train (labels)
positive_indices = np.where(y_train == 1)[0]
negative_indices = np.where(y_train == 0)[0]
# Randomly sample the same number of positive instances as negative
instances
number = 7500
random_positive_indices = np.random.choice(positive_indices, size=number,
replace=True)
random_negative_indices = np.random.choice(negative_indices, size=number,
replace=False)
# Combine the selected indices
selected_indices = np.concatenate([random_negative_indices,
random_positive_indices])
# Create the new oversampled dataset
X_train_oversampled = X_train[selected_indices]
y_train_oversampled = y_train[selected_indices]

#Undersampling Method (run it if it's the case)
# Assuming you have two NumPy arrays: X_train (images) and y_train (labels)
positive_indices = np.where(y_train == 1)[0]
negative_indices = np.where(y_train == 0)[0]
# Randomly sample the same number of negative instances as positive
instances
num_positive_samples = len(positive_indices)
random_negative_indices = np.random.choice(negative_indices,
size=num_positive_samples, replace=False)
# Combine the selected indices
selected_indices = np.concatenate([positive_indices,
random_negative_indices])
# Create the new undersampled dataset
X_train_undersampled = X_train[selected_indices]
y_train_undersampled = y_train[selected_indices]


```

```
#If you use that, you have to Adam to 'Adam' in the optimizer function in
the compile model
from keras.callbacks import LearningRateScheduler  #If you want other
values of lr just change the initial_lrate
def step_decay(epoch):
   initial_lrate = 0.01
   drop = 0.5
   epochs_drop = 5
   lrate = initial_lrate * math.pow(drop,
           math.floor((1+epoch)/epochs_drop))
   return lrate
lrate = LearningRateScheduler(step_decay)

#Build and compile the model for later use. For that, it was used pre-
trained ResNet_50 V2 model from TensorFlow Hub

model = Sequential([
hub.KerasLayer("https://tfhub.dev/google/imagenet/resnet_v2_50/feature_vect
or/5",trainable=False),
    layers.Dense(2, activation='softmax')       # 2 is the number of
classes that we have to classify our image and softmax the function that
will be use to do it
])
model.build([None, 224, 224, 3])  # Batch input shape.

model.compile(optimizer=Adam(learning_rate=0.00001), #Learning Rate can be
changed
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()     #Gives the layers, the output shape and the parameters
that are involved in each layer

from sklearn.preprocessing import LabelBinarizer
# Define a custom class MyLabelBinarizer that inherits from LabelBinarizer
class MyLabelBinarizer(LabelBinarizer):
    # Override the transform method to modify the transformation of labels
    def transform(self, y):
        Y = super().transform(y)            # Call the transform method of
the parent class
        if self.y_type_ == 'binary':
            return np.hstack((Y, 1-Y))      # For binary classification,
stack (Y, 1-Y) horizontally
        else:
            return Y                        # For multiclass, return the
transformed labels as-is

    # Override the inverse_transform method to handle binary and multiclass
cases
    def inverse_transform(self, Y, threshold=None):
        if self.y_type_ == 'binary':
            return super().inverse_transform(Y[:, 0], threshold)    # For
binary, call with only first column
        else:
```

```
            return super().inverse_transform(Y, threshold)

# For multiclass, call with all columns
lenc = MyLabelBinarizer()
Y =  lenc.fit_transform(y_train)

# Use sklearn library to do the split of data to apply to the train and
validation of the model
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val =
train_test_split(X_train,Y,test_size=0.125,random_state=0)

#Data Augmentation
from keras.preprocessing.image import ImageDataGenerator
trdata = ImageDataGenerator(rescale = 1./255,horizontal_flip=True,
vertical_flip=True, rotation_range=90)
traindata = trdata.flow(x=X_train, y=y_train)
vldata = ImageDataGenerator(rescale = 1./255,horizontal_flip=True,
vertical_flip=True, rotation_range=90) #
valdata = vldata.flow(x=X_val, y=y_val)

from keras.callbacks import ModelCheckpoint, EarlyStopping
checkpoint = ModelCheckpoint("rcnn_resnet50.h5", monitor='val_loss',
verbose=1, save_best_only=True, save_weights_only=False, mode='min',
period=1)
early = EarlyStopping(monitor='val_loss', min_delta=0, patience=10,
verbose=1, mode='auto')

#Train and validation of the CNN
hist = model.fit(x=traindata,
                 epochs=100,
                 validation_data=valdata,
                 callbacks=[checkpoint, early])

#Graphs of train and validation accuracy and loss
acc = hist.history['accuracy']
val_acc = hist.history['val_accuracy']
loss = hist.history['loss']
val_loss = hist.history['val_loss']
epochs = range(1,len(acc)+1)
# Add italicized text using LaTeX-style formatting
italic_text_1 = r'$\mathit{Epochs}$'
italic_text_2 = r'$\mathit{Epoch}$'
plt.plot(epochs, acc, 'r', label='Precisão do treino',)
plt.plot(epochs, val_acc, 'b', label='Precisão da
validação',linestyle='dotted')
#plt.title('Precisão do treino e da validação', fontsize=14)
plt.style.use('seaborn-whitegrid')
plt.xlabel(italic_text_1,fontsize=12)
plt.ylabel('Precisão',fontsize=12)
plt.legend(loc='lower right', fontsize=10)
## Find and label max values
max_acc = max(acc)
max_val_acc = max(val_acc)
```

```
min_acc = min(acc)
min_acc_epoch = epochs[acc.index(min_acc)]
max_acc_epoch = epochs[acc.index(max_acc)]
max_val_acc_epoch = epochs[val_acc.index(max_val_acc)]
# Add labels for some values (max for validation and train accuracy and min
for train accuracy)
plt.annotate(f'({max_acc_epoch};{max_acc:.3f}) ',
             xy=(max_acc_epoch, max_acc), xycoords='data',
             xytext=(-25, -30), textcoords='offset points',
             arrowprops=dict(arrowstyle="->", color='black'))
plt.annotate(f'({min_acc_epoch};{min_acc:.3f})',
             xy=(min_acc_epoch, min_acc), xycoords='data',
             xytext=(30, -20), textcoords='offset points',
             arrowprops=dict(arrowstyle="->", color='black'))
plt.annotate(f'({max_val_acc_epoch};{max_val_acc:.3f})',
             xy=(max_val_acc_epoch, max_val_acc), xycoords='data',
             xytext=(-70,-40), textcoords='offset points',
             arrowprops=dict(arrowstyle="->", color='black'))
# Set x-axis ticks at intervals of 5 epochs
xticks_interval = 10
plt.xticks(np.arange(0, len(acc)+10, step=xticks_interval))
plt.yticks(np.arange(0.5,1.01,0.1))
plt.grid(False)
plt.savefig('precision_graph.svg', format='svg')  # Save as SVG
plt.figure()

## Find and label min values
min_loss = min(loss)
min_val_loss = min(val_loss)
min_loss_epoch = epochs[loss.index(min_loss)]
min_val_loss_epoch = epochs[val_loss.index(min_val_loss)]

# Add labels for some values (min for validation and train loss)
plt.annotate(f'({min_loss_epoch};{min_loss:.3f}) ',
             xy=(min_loss_epoch, min_loss), xycoords='data',
             xytext=(-30, 40), textcoords='offset points',
             arrowprops=dict(arrowstyle="->", color='black'))
plt.annotate(f'({min_val_loss_epoch};{min_val_loss:.3f}) ',
             xy=(min_val_loss_epoch, min_val_loss), xycoords='data',
             xytext=(-80, -10), textcoords='offset points',
             arrowprops=dict(arrowstyle="->", color='black'))


plt.plot(epochs, loss, 'g', label='Perdas do treino')
plt.plot(epochs, val_loss, 'y', label='Perdas da
validação',linestyle='dotted')
#plt.title('Perdas do treino e da validação',fontsize=14)
plt.style.use('seaborn-whitegrid')
plt.xlabel(italic_text_1, fontsize=12)
plt.ylabel('Perdas', fontsize=12)
plt.legend(loc='best', fontsize=10)
plt.grid(False)
plt.xticks(np.arange(0, len(acc)+10, step=xticks_interval))
plt.yticks(np.arange(0,0.8,0.1))
```

```
plt.savefig('loss_graph.svg', format='svg')  # Save as SVG
plt.figure()
plt.show()
```

B.4 Teste da CNN

```
import os,cv2
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import math as math
import tensorflow as tf
import tensorflow_hub as hub


model = tf.keras.models.load_model(
       ('nameofmodel.keras'),
       custom_objects={'KerasLayer':hub.KerasLayer})
model.summary()

#The function get_iou calculates the Intersection over Union (IOU) between
two bounding boxes (rectangular regions)

def get_iou(bb1, bb2):              #bb1 and bb2 are two dictionaries
inputs where are the 4 necessary coordinates of the bounding boxes
    assert bb1['x1'] < bb1['x2']    #Conditions that the program need to
guarantee to do the calculus correctly
    assert bb1['y1'] < bb1['y2']
    assert bb2['x1'] < bb2['x2']
    assert bb2['y1'] < bb2['y2']

    x_left = max(bb1['x1'], bb2['x1'])  #It takes the maximum or minimum to
know what's the overlapping region of the bounding boxes
    y_top = max(bb1['y1'], bb2['y1'])
    x_right = min(bb1['x2'], bb2['x2'])
    y_bottom = min(bb1['y2'], bb2['y2'])

    if x_right < x_left or y_bottom < y_top:    #If one of this specific
conditions is true, so there isn't an overlapping region between the two
bounding boxes and the return value will be 0
        return 0.0

    intersection_area = (x_right - x_left) * (y_bottom - y_top) #Simple
calculus of the intersection area

    bb1_area = (bb1['x2'] - bb1['x1']) * (bb1['y2'] - bb1['y1']) # Calculus
of the area of the two bounding boxes individually
    bb2_area = (bb2['x2'] - bb2['x1']) * (bb2['y2'] - bb2['y1'])
    iou = intersection_area / float(bb1_area + bb2_area -
intersection_area) #Calculus of the Intersection over Union (IOU)
```

```
    assert iou >= 0.0      #The value has to be between 0 and 1
    assert iou <= 1.0
    return iou

test_images = []       # List to store the preprocessed images
test_labels = []       # List to store labels corresponding to each image
path = "Test data"
annot = "Test_anot"
cv2.setUseOptimized(True)
ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
for e,i in enumerate(os.listdir(annot)):
    try:
        filename = i.split(".")[0]+".jpg"       #Create the filename by
replacing the extensio with ".jpg"
        print(e,filename)
        image = cv2.imread(os.path.join(path,filename))     #Read the image
from the path
        df = pd.read_csv(os.path.join(annot,i))             #Read the
annotation data from CSV
        im_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)     #Convert image
to RGB
        gtvalues=[]         # List to store ground truth bounding box
coordinates
        for row in df.iterrows():
            x1 = int(row[1][0].split(" ")[0])          #Extract the
coordinates of the bounding box from the annotation data
            y1 = int(row[1][0].split(" ")[1])
            x2 = int(row[1][0].split(" ")[2])
            y2 = int(row[1][0].split(" ")[3])
            gtvalues.append({"x1":x1,"x2":x2,"y1":y1,"y2":y2})      #Append
the values to gtvalues to use it later

        #Initialize Selective Search for the image
        ss.setBaseImage(image)
        ss.switchToSelectiveSearchFast()    #Switch to Fast Selective
Search
        ssresults = ss.process()    #Results of the Selective Search
process
        imOut = im_rgb.copy()       #Copy of the RGB image to a new
variable

        counter = 0         #Counter for positive samples (when label = 1)
        falsecounter = 0    #Counter for negative samples (when label = 0)
        flag = 0            #Flag to control loop termination
        pflag = 0           #Flag to indicate positive sample limite
reached
        nflag = 0           #Flag to indicate negative sample limite
reached

        for e,result in enumerate(ssresults):       #Loop through selective
search results
            if e < 2000 and flag == 0:
                for gtval in gtvalues:
                    x,y,w,h = result
```

```
                    iou = get_iou(gtval,{"x1":x,"x2":x+w,"y1":y,"y2":y+h})
#Calculate Intersection Over Union (IOU)

                    #If IOU > 0.7 => Positive samples
                    if counter < 20:
                        if iou > 0.70:
                            roi = imOut[y:y+h, x:x+w]  # Extract the ROI
from the RGB image in imOut
                            resized = cv2.resize(roi, (224,224),
interpolation = cv2.INTER_AREA)    #Resized the image for what's matter
                            test_images.append(resized)    #Append the
image to train_images list
                            test_labels.append(1)          #Append the
respective label to train_labels list
                            counter += 1
                    else :
                        pflag =1        #The positive samples limit are
reached

                    #If IOU < 0.3 => Negative samples
                    if falsecounter < 20:
                        if iou < 0.30:
                            roi = imOut[y:y+h, x:x+w]  # Extract the ROI
from the RGB image in imOut
                            resized = cv2.resize(roi,(224,224),
interpolation = cv2.INTER_AREA)     #Resized the image for what's matter
                            test_images.append(resized)    #Append the
image to train_images list
                            test_labels.append(0)          #Append the
respective label to train_labels list
                            falsecounter += 1
                    else :
                        nflag = 1       #The negative samples limit are
reached

                #Terminate the loop if both positive and negative samples
limits are reached
                if pflag == 1 and nflag == 1:
                    print("inside")
                    flag = 1
    except Exception as e:
        print(e)
        print("error in "+filename)
        continue
# Convert lists to numpy arrays
X_test = np.array(test_images)
y_test = np.array(test_labels)

#Undersampling Method
# Assuming you have two NumPy arrays: X_test (images) and y_test (labels)
positive_indices = np.where(y_test == 1)[0]
negative_indices = np.where(y_test == 0)[0]


```

```
# Randomly sample the same number of negative instances as positive
instances
num_positive_samples = len(positive_indices)
random_negative_indices = np.random.choice(negative_indices,
size=num_positive_samples, replace=False)

# Combine the selected indices
selected_indices = np.concatenate([positive_indices,
random_negative_indices])

# Create the new undersampled dataset
X_test_undersampled = X_test[selected_indices]
y_test_undersampled = y_test[selected_indices]
X_test_scaled = X_test_undersampled/255
lenc = MyLabelBinarizer()
Y_test =  lenc.fit_transform(y_test_undersampled)

from sklearn.metrics import confusion_matrix, classification_report
y_pred = model.predict(X_test_scaled)
y_pred_classes = [round(element[0]) for element in y_pred]
selected_labels = [row[0] for row in Y_test]

print("Classification Report: \n", classification_report(selected_labels,
y_pred_classes))
cm =
tf.math.confusion_matrix(labels=selected_labels,predictions=y_pred_classes)
import seaborn as sn
plt.figure(figsize =(10,7))
sn.heatmap(cm,annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')

# Create the new undersampled dataset (to visualize the results of an
inspection)
cv2.setUseOptimized(True);
ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()

path= "Test_image"
#test_images = []

for e,i in enumerate(os.listdir(path)):
    img = cv2.imread(os.path.join(path,i))
    im_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    #plt.imshow(im_rgb)
    ss.setBaseImage(im_rgb)
    ss.switchToSelectiveSearchFast()
    ssresults = ss.process()
    #print(ssresults)
    imout = im_rgb.copy()
    for e,result in enumerate(ssresults):
        if e < 2000:
            x,y,w,h = result
            timage = imout[y:y+h,x:x+w]

```

```
            resized = cv2.resize(timage, (224,224), interpolation =
cv2.INTER_AREA)
            resized = resized / 255.0
            image = np.expand_dims(resized, axis=0)
            out= model.predict(image)
            print(out)
            if out[0][0] > 0.7:
                cv2.rectangle(imout, (x, y), (x+w, y+h), (0, 255, 0), 1,
cv2.LINE_AA)
    plt.figure()
    plt.grid(False)
    plt.imshow(imout)
```

B.5 Teste da inspeção por vídeo

```
import numpy as np
import os,cv2
import tensorflow as tf
import tensorflow_hub as hub
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering

#Load Model
model = tf.keras.models.load_model(
       ('nameofthemodel.keras'),
       custom_objects={'KerasLayer':hub.KerasLayer})
model.summary()


def find_rectangle(centroids, all_retangles, all_centroids):
      '''
    centroids: list of centroids which retangles need to be find, each
centroid (x,y)
    all_rectangles: list of all retangles, each retangle (x,y,x1,y1)
    all_centroids: list of all centroids, each centroid (x,y)

    output:

lenght of the rectangle in X axis
lenght of the rectangle in Y axis
mean lenght of the rectangle in X axis
mean lenght of the rectangle in Y axis

'''
    ans_x = []
    ans_y = []

    for c in centroids:
        for indx in range(len(all_centroids)):
            if (all_centroids[indx] == c).all():
```

```
                r = all_retangles[indx]
                ans_x.append(abs(r[0] - r[2]))
                ans_y.append(abs(r[1] - r[3]))
    return ans_x, ans_y, np.mean(ans_x), np.mean(ans_y)

cv2.setUseOptimized(True)
ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()

# Path to the video file
video_path = "Filename.avi"

# Threshold for classification confidence
confidence_threshold = 0.7

centroides = []

# List to store all centroids
all_centroids = []
all_retangles = []

#Area limitation
area_threshold = 10000

# Matrix for total frames of the video
som_mat = np.zeros((256, 336))

# Counter for the total number of frames processed
total_frames_processed = 0

# Open the video file
video_capture = cv2.VideoCapture(video_path)

# Check if the video file was successfully opened
if not video_capture.isOpened():
    print("Error: Could not open video file.")
    exit()

# Determine the total number of frames in the video
total_frames_in_video = int(video_capture.get(cv2.CAP_PROP_FRAME_COUNT))

# Calculate the number of frames to eliminate from the start and end (this
value can change)
frames_to_eliminate = int(total_frames_in_video * 0.25)

# Number of frames to select
frames_to_select = 20
frame_interval = 10


for frame_index in range(frames_to_eliminate, total_frames_in_video -
frames_to_eliminate, frame_interval):
    # Set the video capture to the specific frame index
    video_capture.set(cv2.CAP_PROP_POS_FRAMES, frame_index)
    ret, frame = video_capture.read()
```

```
    # Break the loop if we have reached the desired number of frames
    if total_frames_processed >= frames_to_select:
        break

    # Skip the first and last 33% of frames
    if frame_index < frames_to_eliminate or frame_index >=
total_frames_in_video - frames_to_eliminate:
        continue

    im_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    ss.setBaseImage(im_rgb)
    ss.switchToSelectiveSearchFast()
    ssresults = ss.process()
    imout = im_rgb.copy()
    mat = np.zeros((256, 336))

    # List to store centroids for the current frame
    image_centroids = []

    for e, result in enumerate(ssresults):
        if e < 2000:
            x, y, w, h = result
            timage = im_rgb[y:y + h, x:x + w]
            resized = cv2.resize(timage, (224, 224),
interpolation=cv2.INTER_AREA)
            resized = resized / 255.0
            image = np.expand_dims(resized, axis=0)
            out = model.predict(image)

            # Check if the classification confidence is above the threshold
            if out[0][0] > confidence_threshold:
                area = w * h  # Calculate the area of the region
                if area <= area_threshold:  # Discard regions with area
above the threshold
                    mat[y:y + h, x:x + w] = 1
                    xbarra = x + w / 2
                    ybarra = y + h / 2
                    all_retangles.append([x, y, x + w, y + h])
                    centroid = [xbarra, ybarra]  # Corrected order
                    cv2.rectangle(imout, (x, y), (x + w, y + h), (0, 255,
0), 1, cv2.LINE_AA)
                    image_centroids.append(centroid)
                    all_centroids.append(centroid)

    # Update the total number of frames
    total_frames_processed += 1

    # Update the som_mat matrix
    som_mat = som_mat + mat
    centroides.append(image_centroids)

# Release the video capture object
video_capture.release()

```

```
# Convert the list of all centroids to a NumPy array
all_centroids_array = np.array(all_centroids)

# Apply Agglomerative Clustering to group similar centroids
agg_clustering = AgglomerativeClustering(n_clusters=None,
distance_threshold=200.0,
compute_distances=True,compute_full_tree=True).fit(all_centroids_array)

# Plot clustered centroids
num_clusters = agg_clustering.n_clusters_  # Get the actual number of
clusters
for cluster_label in range(num_clusters):
    cluster_points = all_centroids_array[agg_clustering.labels_ ==
cluster_label]

    # Calculate means for x and y coordinates
    mean_x = np.mean(cluster_points[:, 0])
    mean_y = np.mean(cluster_points[:, 1])

    x_ret, y_ret, avg_width, avg_height = find_rectangle(cluster_points,
all_retangles, all_centroids)
    print(avg_width, avg_height)

    # Draw a rectangle around the centroid of the cluster
    x_rect = int(mean_x - avg_width / 2)
    y_rect = int(mean_y - avg_height / 2)
    w_rect = int(avg_width)
    h_rect = int(avg_height)

    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], label=f'Cluster
{cluster_label + 1}')
    plt.scatter(mean_x, mean_y, c='black', marker='X', s=100)
    rect = plt.Rectangle((x_rect, y_rect), w_rect, h_rect, fill=False,
edgecolor=(0, 1, 0), linewidth=1,
                         linestyle='-', alpha=1)
    plt.gca().add_patch(rect)

plt.imshow(som_mat, cmap='gray')
plt.colorbar()  # Add a colorbar to show the mapping of values
plt.legend()
plt.show()

# Visualize the original frame with cluster centroids and their areas after
processing all images
plt.figure(figsize=(10, 8))
plt.imshow(im_rgb)

# Plot cluster centroids on the original frame
for cluster_label in range(num_clusters):
    cluster_points = all_centroids_array[agg_clustering.labels_ ==
cluster_label]

    # Calculate means for x and y coordinates
    mean_x = np.mean(cluster_points[:, 0])
```

```
    mean_y = np.mean(cluster_points[:, 1])
    # Draw a rectangle around the centroid of the cluster
    x_rect = int(mean_x - avg_width / 2)
    y_rect = int(mean_y - avg_height / 2)
    w_rect = int(avg_width)
    h_rect = int(avg_height)

    plt.scatter(mean_x, mean_y, c='black', marker='X', s=100)
    rect = plt.Rectangle((x_rect, y_rect), w_rect, h_rect, fill=False,
edgecolor=(0, 1, 0), linewidth=1, linestyle='-', alpha=1)
    plt.gca().add_patch(rect)

plt.legend()  # Add legend based on labels in the previous scatter plots
plt.show()
```