UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE ENGENHARIA GEOGRÁFICA, GEOFÍSICA E ENERGIA

Redes neurais de convolução na classificação de edifícios em imagens de alta resolução espacial

Henrique Coutinho Varela da Silva

Mestrado em Engenharia Geoespacial

Dissertação orientada por:
Professor Doutor João Catalão Fernandes

2022

Agradecimentos

Antes de mais gostaria de aproveitar para agradecer a todas as pessoas que contribuíram e estiveram presentes ao longo do meu desafiante percurso académico e desenvolvimento deste trabalho final.

Em primeiro lugar, um agradecimento especial ao orientador Professor Doutor João Catalão Fernandes pela enorme disponibilidade e interesse em acompanhar e ajudar o processo de desenvolvimento desta tese. Gostaria também de agradecer, no geral, a todos os docentes do mestrado em Engenharia Geoespacial pelos vários conhecimentos transmitidos ao longo do curso, essenciais para o meu desenvolvimento pessoal e profissional.

Agradeço à minha família por todo o apoio e motivação académico e pessoal, especialmente ao meu Pai e à minha Mãe.

Finalmente, devo também um agradecimento a todas as outras pessoas que deram o seu contributo ao longo deste percurso, sobretudo à minha namorada.

Resumo

A presente tese tem como principal objetivo introduzir e explorar os conceitos de Inteligência Artificial e *Machine Learning*, com especial foco nas *Convolutional Neural Networks*. Nesta, descreve-se o processo de criação de um ambiente eficiente para treino, validação e teste de modelos profundos criados com capacidade de realizar a segmentação automática de edifícios em imagens aéreas e espaciais de alta resolução, alertando-se para a crescente importância e eficiência deste tipo de técnicas para propósitos e tarefas relacionadas com a Engenharia Geoespacial.

A informação extraída de imagens aéreas e espaciais apresenta aplicações significativas em várias disciplinas da Engenharia Geoespacial, como a deteção remota, análise espacial de dados, produção cartográfica, entre outras. Atualmente, grande parte da extração ainda é realizada por especialistas humanos, tornando o processo lento, caro e propenso a erros. O trabalho desenvolvido explora e desenvolve métodos para extrair automaticamente a localização e forma de entidades geoespaciais, nomeadamente edifícios.

Recorre-se à utilização da recente arquitetura *U-net*, que se tem vindo a destacar pela sua contribuição em tarefas de segmentação de imagens de alta resolução, obtidas utilizando técnicas espaciais e aéreas de aquisição de informação radiométrica e geoespacial. A programação em *Python* constituiu a base do trabalho desenvolvido, tendo sido necessário criar um conjunto de ferramentas, que foram publicadas num repositório *online GitHub*, de manipulação, processamento e testagem dos vários modelos profundos de *Machine Learning* e conjuntos de dados *big data.*

Após realizado o treino em nuvem dos vários modelos, estes foram analisados e comparados, com recurso a dois conjuntos de dados de validação criados para o efeito, com o objetivo de se avaliar a qualidade e influência das várias técnicas de treino realizadas, especialmente em termos de generalização e capacidade aplicacional. Algumas experiências demonstram resultados de precisão e revocação simultaneamente acima de 90% em certas amostras.

Palavras-chave: Artificial; Aprendizagem; Automatização; Geoespacial; Convolução.

Abstract

The main objective of this thesis is to introduce and explore the concepts of Artificial Intelligence and Machine Learning, with a special focus on Convolutional Neural Networks. The process of creating an efficient environment for training, validation and testing of deep models created, with the ability to perform the automatic segmentation of buildings in high resolution aerial and spatial images, is described, alerting to the growing importance and efficiency of this type of techniques for purposes and tasks related to Geospatial Engineering.

The information extracted from aerial and space images has significant applications in several disciplines of Geospatial Engineering, such as remote sensing, spatial data analysis, cartographic production, among others. Currently, much of the extraction is still performed by human experts, making the process slow, expensive, and error-prone. The work developed explores and develops methods to automatically extract the location and shape of geospatial entities, namely buildings.

The recent U-net architecture is used, which has been highlighted for its contribution in high resolution image segmentation tasks, obtained using spatial and aerial techniques for the acquisition of radiometric and geospatial information. Python programming formed the basis of the work developed, and it was necessary to create a set of tools, which were published in an online GitHub repository, for the manipulation, processing and testing of the various deep models of Machine Learning and big data datasets.

After performing the cloud training of the various models, they were analyzed and compared, using two sets of validation data created for this purpose, in order to assess the quality and influence of the various training techniques performed, especially in terms of generalization and application capability. Some experiments demonstrate accuracy and recall results simultaneously above 90% in certain samples.

Keywords: Artificial; Learning; Automation; Geospatial; Convolution.

Índice

Índice de Tabelas

Índice de Figuras

Acrónimos

VANT	Veículo Aéreo Não Tripulado
CNN	Rede Neural Convolucional (*Convolutional Neural Network*)
DL	Aprendizagem Profunda (*Deep Learning*)
PME	Pequenas e Médias Empresas
UE	União Europeia
IA	Inteligência Artificial
ML	Aprendizagem Automática (*Machine Learning*)
ONU	Organização das Nações Unidas
ESA	Agência Espacial Europeia (*European Spatial Agency*)
SVM	Máquina de Vetor de Suporte (*Support Vector Machine*)
RF	Floresta Aleatória (*Random Forest*)
FCN	Redes Totalmente Conectadas (*Fully Connected Networks*)
3D	Três Dimensões
SGD	Gradiente Estocástico Descendente (*Stochastic Gradient Descent*)
VRAM	Memória de Vídeo de Acesso Aleatório (*Video Random Access Memory*)
RAM	Memória de Acesso Aleatório (*Random Access Memory*)
Tanh	Tangente Hiperbólica
ReLU	Unidade Retificada Linearmente (*Rectified Linear Unit*)
API	Interface de Programação de Aplicações (*Application Programming Interface*)
GPU	Unidade Gráfica de Processamento (*Graphic Processing Unit*)
SIG	Sistema de Informação Geográfica
SNIG	Sistema Nacional de Informação Geográfica
DGT	Direção Geral do Território

1 Introdução

1.1 Enquadramento

Os sensores espaciais, aéreos e baseados em Veículos Aéreos Não Tripulados (VANT), que utilizam tecnologias avançadas de observação da Terra, têm a possibilidade de obter grandes quantidades e diferentes tipos de imagens de alta resolução. Estas imagens, com elevada resolução temporal e espacial, podem ser uma mais valia para disciplinas relacionadas com a Engenharia Geoespacial como é o caso da produção cartográfica, análise espacial de dados, deteção remota, entre outras. Podem fornecer, por exemplo, dados sobre a dimensão e extensão da destruição de infraestruturas e bens de uma zona afetada por um certo desastre natural, ou até mesmo contribuir para uma maior completagem de sistemas baseados na localização. Neste sentido, é desejável o desenvolvimento de ferramentas automáticas (ou semiautomáticas) que possibilitem a identificação e extração rápida de entidades geoespaciais (p.e. estradas, edifícios, cobertura do solo) de imagens aéreas de alta resolução para elaboração de diversas análises espaciais.

Uma técnica recente que pode promover amplamente a precisão da extração automatizada de entidades geoespaciais são as redes neurais convolucionais (*Convolutional Neural Networks*, *CNN*), um ramo importante da aprendizagem profunda (*Deep Learning*, *DL)* que melhorou substancialmente o desempenho da última geração da segmentação de imagens.

A quantidade de dados de imagem na *web* aumentou rapidamente nos últimos anos, a par com os avanços da *internet* e dos terminais multimédia. Para resolver problemas visuais de grande escala, com recurso a esta enorme quantidade de dados, o processamento automático de dados é realizado com a ajuda de computadores. A tecnologia automática de processamento de imagem torna-se mais significativa para deteção, classificação e segmentação de variados objetos [1]. À medida que avançamos para uma compreensão mais completa da imagem, a identificação mais precisa, detalhada e automatizada dos objetos/entidades torna-se crucial para praticamente qualquer área.

Recentemente, com a prevalência de métodos de *DL*, que chegam a alcançar níveis de desempenho impressionantes em muitas aplicações que envolvem imagens, incluindo classificação [2], deteção de objetos [3] e segmentação semântica [4], pode-se afirmar que a representação e aquisição deste tipo de informações entrou numa nova fase. Ao contrário dos recursos de nível baixo e médio, os modelos de *DL* podem aprender recursos mais abstratos e discriminativos das imagens com recurso às *CNN*, explicadas em detalhe neste documento. Os métodos automáticos, no contexto de deteção remota, abrangem uma vasta gama de aplicações, incluindo classificação de uso e cobertura do solo [5], deteção de mudanças [6], seleção e extração de entidades [7], entre outros.

Na Engenharia Geoespacial existem imensas aplicações que podem advir do uso de *CNN*, nomeadamente na disciplina de deteção remota. O próprio acompanhamento evolutivo de plataformas de partilha de dados, bem como a recente capacidade de processamento em nuvem, têm vindo a contribuir para o crescimento exponencial destas no geral. Enquanto um permite a realização de um treino completo com elevado número de dados em formato imagem, o outro permite uma grande capacidade de processamento rápido desta aprendizagem.

Alguns projetos europeus relacionados com a deteção remota podem ser referenciados e expõem a grande necessidade de haver informação geoespacial normalizada e aberta, sendo, por vezes, apontada a necessidade da posterior automatização da extração e organização dos dados que advêm desta.

O projeto *PARSEC* [8], por exemplo, focava-se na criação de serviços sustentáveis para ajudar as PME europeias a aumentar o seu perfil competitivo global, através do acesso a uma plataforma com capacidade de fornecer informação geoespacial, aproveitando a base de dados *Copernicus* da UE e outros recursos. Assim, o sistema fornecia às empresas de deteção remota acesso a suporte, tecnologia, capital e mercados. Este sistema era facilitado por meio da colaboração entre *clusters* a nível europeu, incentivando a inovação e criando novos mercados globais. O objetivo passou por ajudar as PME inovadoras a fazer melhor uso da vasta informação geoespacial de Observação da Terra disponível, criando-lhes um ambiente propício à maior produtividade, menor pegada de carbono e à capacidade de expansão para mais setores e países, através da utilização mais aberta deste tipo de dados.

Por outro lado, a enorme quantidade de informação ambiental disponível, necessária à tomada de decisão, pode ser esmagadora dentro da União Europeia. Recentemente o projeto *Landsupport* [9], financiado pela UE, desenvolveu uma plataforma *web* que compila e processa dados de sensores espaciais, aéreos e baseados em VANT, associando-os a ferramentas *web*, muitas delas baseadas em modelos de *DL*. Usando estas capacidades, agricultores, formuladores de políticas e até cidadãos podem facilmente encontrar informação relevante e realizar escolhas ambientais informadas. Muitos setores, como agricultura e silvicultura, precisam não só de acesso à informação, como também precisam de suporte operacional robusto para garantir que os dados brutos adquiridos são facilmente interpretados, podendo ser posteriormente traduzidos em ações positivas por estes utilizadores. Este foi o ponto de partida do projeto *Landsupport*, financiado pela UE, que reuniu 19 parceiros de 10 países diferentes da Europa, Médio Oriente e Ásia. Desta forma, o projeto visou desenvolver um sistema de apoio à decisão geoespacial totalmente gratuito e de acesso aberto, baseado na *web*, dedicando-se especialmente ao apoio à agricultura e silvicultura sustentáveis, avaliação dos benefícios entre usos do solo e à contribuição para o desenvolvimento e implementação de políticas de uso do solo na Europa [10].

É importante também referir o projeto *ExtremeEarth* [11], financiado pelos mesmos instrumentos que os projetos até aqui expostos, e acabando por ser um dos únicos relacionados simultaneamente com a organização e categorização em grande escala de dados, utilizando algoritmos de IA, juntamente com uma base de dados europeia de imagens satélite – o *Copernicus*. O *Copernicus* é o programa europeu de monitorização da Terra. Os dados geoespaciais produzidos pelos satélites *Sentinel* colocam-no na vanguarda do paradigma do *big data*. Assim, o *ExtremeEarth* concentrava-se no desenvolvimento das tecnologias que tornaram a Europa pioneira na área de *"Extreme Earth Analytics"*, referente às técnicas de deteção remota e IA que são necessárias para extrair informação e conhecimento dos *petabytes* de dados de plataformas abertas como o *Copernicus*. As atividades de pesquisa e inovação realizadas no *ExtremeEarth* permitiram o avanço significativo das fronteiras do *big data*, *Earth Analytics* e *Deep Learning*.

Os três projetos referidos (*PARSEC*, *Landsupport* e *ExtremeEarth*) tiveram como objetivo geral a criação de um maior e melhor acesso a dados relativos a informação geoespacial. Além disso, foram projetos que alertaram para a importância de serem criados métodos automáticos e semiautomáticos que auxiliem na gestão de dados e na consequente simplificação da interpretação dos mesmos. Estes foram financiados pelo programa *Horizon 2020*, que constituiu o programa de financiamento de pesquisa e inovação da UE para o período de 2014-2020, com um orçamento de quase 80 biliões de euros [12]. Mais recentemente o projeto foi renomeado *Horizon Europe*, com financiamento e objetivos delineados até 2027. Os seus objetivos essenciais são combater as alterações climáticas, ajudar a alcançar os objetivos de desenvolvimento sustentável da ONU e impulsionar a competitividade e o crescimento da UE.

É possível constatar que o processo de disponibilização dos dados (*Open Data*), proveniente de técnicas da deteção remota, começa a demonstrar um grande crescimento, mas ainda se encontra constrangido

devido a uma grande necessidade primária de organização dos enormes conjuntos de dados dispersos e que aumentam diariamente. De facto, a *DL* aplicada à extração de entidades geoespaciais utilizando *CNN* acaba por ser bastante afetada por estes atrasos, sendo os recursos de treino, necessários à criação precisa de modelos automáticos deste género, por vezes bastante escassos ou desorganizados e difíceis de encontrar ou adquirir. Devido às suas imensas aplicações, a motivação e financiamento de projetos que pretendam tentar resolver este tipo de problemáticas são considerados de extrema importância para o desenvolvimento eficiente de modelos automáticos de extração de entidades geoespaciais. A própria organização dos dados poderia ser otimizada com maior recurso a algoritmos de *ML* que os classificassem e organizassem de forma semiautomática.

Os vários avanços tecnológicos resultaram na crescente disponibilidade de dados provenientes da Observação da Terra (aérea ou espacial). As novas plataformas de aquisição de dados, como os Microssatélites ou os VANT, facilitam a observação da superfície da Terra com detalhes espaciais crescentes. Além disso, há uma tendência contínua de aumento da partilha de dados e acesso aberto como é o caso do *website OpenAerialMap*, o programa *NEON* da *US National Science Foundation* e o programa *Copernicus Open Access Hub* da UE e da *ESA*, entre outros. Assim, a automatização da aquisição de informação destes enormes conjuntos de dados torna-se decisiva, de modo a serem criadas análises rápidas que podem ser bastantes úteis em deteção remota, além de permitirem a dispensa de certos processos manuais morosos. A capacidade de uma máquina conseguir extrair recursos com grandes velocidades que aumentam de ano para ano e sem qualquer necessidade de descanso, cria um ambiente propício para a exploração de algoritmos que automatizem tarefas manuais deste género.

1.2. Motivação e Objetivos

Os avanços na área do processamento de imagem e na própria partilha de dados potenciam novas tentativas de deteção, construção e segmentação de entidades geoespaciais em imagens aéreas e espaciais, de forma a facilitar a criação de cartografia digital e de apoio à análise espacial. Assim sendo, nesta tese, propõe-se o desenvolvimento de ferramentas que possibilitam a experimentação de uma arquitetura *CNN* para classificação automática de edifícios em imagens de alta resolução espacial, com o objetivo de se elaborar cartografia de forma automatizada e com várias aplicações que também serão aqui expostas.

O treino deste classificador de *DL* requer um elevado número de amostras pelo que, neste trabalho, serão exploradas as bases de dados disponíveis na *web*, sendo também avaliado o contributo destas na exatidão do resultado final. É importante salientar o uso de uma técnica recente, de computação e programação *Python* em nuvem, que permite o processamento com desempenhos equivalentes a investimentos em *hardware* computacional de milhares de euros - o *Google Colab* [13]. Foi assim possível criar um ambiente propício ao treino de milhares de imagens, provenientes das bases de dados *online* - os chamados *big data* - aliado a velocidades de processamento elevadas. Será também explorada a abordagem de transferência do conhecimento (*Transfer Learning*) como estratégia de mitigação do sobre ajustamento e da redução da dimensão da amostra de treino do classificador.

O objetivo principal será então avaliar o contributo das bases de dados imagem públicas no processo de treino do classificador para a segmentação dos vários edifícios existentes em imagens aéreas, neste caso imagens ortoretificadas adquiridas por uma aeronave (como proxy de imagens de satélite de muito alta resolução) e imagens obtidas por satélite. Será analisada a precisão dos vários modelos treinados em conjuntos de dados compilados e normalizados para o efeito, de forma a ser possível tirar conclusões e realizar comparações baseadas em estatística e comparação de resultados empíricos, com base em várias

métricas - a fase de validação. Serão também apresentadas e testadas algumas aplicações que podem tirar proveito desta aquisição automática de dados e que confirmam os resultados validados - a fase de teste.

Para concretizar este objetivo será utilizada a arquitetura *U-net* e dados de treino de *big data* provenientes de múltiplos *datasets* existentes na *web*. A compilação e tratamento dos vários *datasets* de treino *big data*, provenientes de conjuntos enormes de dados rotulados criados e partilhados na *internet* por outros autores, foi um dos passos mais importantes para a criação de um ambiente favorável à segmentação generalizada de edifícios. Os modelos criados com estes foram aplicados a um conjunto de dados de validação que pode apresentar caraterísticas de textura e forma de edifício desafiadoras em termos de segmentação automática, sendo procuradas soluções que resolvam esta problemática.

1.3 Estrutura do trabalho

O trabalho desenvolvido apresenta-se neste documento dividido por cinco capítulos: Introdução, Estado da Arte, Metodologia, Resultados e Discussão e Conclusão.

O presente capítulo refere-se à Introdução e foram aqui expostos os objetivos principais da tese bem como uma apresentação do tema no geral e, nesta seção, uma breve descrição da estrutura do trabalho.

No capítulo Estado da Arte são analisados e sintetizados os principais desenvolvimentos na matéria de aprendizagem automática (*Machine Learning, ML*), com especial foco nas *CNN*, explicitando em detalhe o seu funcionamento e principais aplicações. De seguida, apresenta-se o capítulo Metodologia que descreve todo o procedimento realizado para treinar, validar e testar modelos de segmentação automática próprios e relacionados com a extração de entidades geoespaciais.

No final, no capítulo Resultados e Discussão, são analisados e discutidos os resultados obtidos, através da validação e testagem dos modelos criados, e explorados alguns exemplos aplicacionais relacionados com disciplinas ligadas à Engenharia Geoespacial. Além disso, na Conclusão, será feita uma análise geral do trabalho desenvolvido, ressalvando-se dificuldades, respostas e possíveis trabalhos que podem vir a tirar proveito desta tese.

2 Estado da Arte

2.1 Inteligência Artificial (IA) e Aprendizagem Automática (*ML*)

Menos de uma década depois de intercetar a máquina de criptografia nazi "*Enigma*" e ajudar as Forças Aliadas a vencer a Segunda Guerra Mundial, o matemático Alan Turing mudou a história pela segunda vez com uma pergunta simples: "As máquinas podem pensar?". O artigo *Computing Machinery and Intelligence* [14] e as experiências de Turing estabeleceram o objetivo e a visão fundamentais da IA. Na sua essência, é o ramo da ciência da computação que visa responder afirmativamente à pergunta de Turing. É o esforço de copiar ou simular a inteligência humana em máquinas. O objetivo expansivo da IA deu origem a muitas questões e debates, tanto que nenhuma definição singular é universalmente aceite [15].

Desde a revolução industrial, houve um grande desenvolvimento no campo da tecnologia, tendo esta contribuído para a substituição e otimização de muitos trabalhos manuais. A IA é uma das inovações tecnológicas que surgiu, ajudando a otimizar muitas tarefas que, sem as suas diversas aplicações, só poderiam ser realizadas com recurso a inteligência e esforço estritamente humano.

Esta pode usar dados externos, como *big data*, para obter um melhor desempenho para as tarefas fornecidas - *ML* - baseando-se na arte e ciência de programar computadores, de forma a que estes consigam aprender de forma autónoma, após lhe serem apresentados dados que os treinem para tal. A *ML* é um assunto simultaneamente interdisciplinar e multidisciplinar, envolvendo teoria da probabilidade, estatística, teoria da aproximação, análise convexa, teoria da complexidade de algoritmos e outras disciplinas [15]. É uma especialização da forma como os computadores simulam ou implementam o comportamento da aprendizagem humana, para adquirir novos conhecimentos ou habilidades.

Os dados que um sistema usa para aprender são chamados de conjunto de dados treino. Cada exemplo de treino é chamado a instância de treino (ou amostra). Neste trabalho, a tarefa é segmentar classes de edifícios quando estas surgem numa certa imagem, sendo a aprendizagem realizada através de dados de treino e no final avaliada por uma ou várias medidas de desempenho que precisam de ser posteriormente definidas. Por exemplo, o filtro de *spam* do *email* foi uma das primeiras aplicações de *ML* que se tornou padrão, aprendendo a marcar mensagens como sendo "lixo" após lhe serem apresentadas várias outras deste tipo e também normais, oferecendo-lhe capacidade de distinção [15].

A *ML* pode também servir de ferramenta à compreensão humana: os algoritmos podem ser inspecionados para se tentar estudar o que aprenderam do treino. Por exemplo, uma vez treinado o filtro de *spam*, pode ser facilmente inspecionado para revelar a lista de palavras e combinações de palavras que utiliza como melhores preditores de *spam*. Por vezes, revelam-se correlações insuspeitas ou novas tendências e, assim, é possível um humano ganhar uma melhor compreensão acerca do problema em questão. Além disso, as aplicações de técnicas de *ML* podem explorar grandes quantidades de dados, ajudando a descobrir padrões que não eram imediatamente aparentes - a chamada mineração de dados [15].

Por sua vez, *DL* pode ser apontada como um dos novos campos que constituem a *ML*. O conceito baseia-se em construir e simular a rede neural do cérebro humano, simulando artificialmente o mecanismo deste para interpretar dados. É um tipo de aprendizagem supervisionada cujas pesquisas se focam nas redes neurais artificiais [16].

É pertinente afirmar que a *ML*, no geral, pode trazer inúmeros benefícios, em situações como problemas para os quais as soluções existentes necessitam de um grande afinamento manual ou listas longas de regras, problemas complexos para os quais não existe uma solução tradicional, ambientes flutuantes ou até mesmo na aquisição de ideias acerca de problemas difíceis e com grandes quantidades de dados [15].

2.2 *ML* em Engenharia Geoespacial

O objetivo principal deste projeto final é treinar e testar vários modelos de *DL* com a capacidade de classificar, automaticamente e ao nível do píxel, entidades geoespaciais em diferentes tipos de imagens de alta resolução, com dados de treino que correspondem imagens e respetivas máscaras binárias (por exemplo, edifícios rotulados a branco e "não-edifícios" a preto). Nesta seção será abordada a importância de extração de dados geoespaciais com recurso a algoritmos de *ML*.

Dados geoespaciais são dados que descrevem objetos, eventos, ou outros recursos com localização na superfície da Terra ou perto dela. Este tipo de dados geralmente combina informação de localização, coordenadas tridimensionais respetivamente a um sistema de referência, informação temporal e atributos para cada classe [17]. Assim, uma classe geoespacial carateriza-se como sendo um dado geoespacial que reúne um conjunto de atributos caraterísticos e comuns entre os outros que a constituem, por exemplo, classe de edifícios, classe de vegetação, classe de estradas, entre muitas outras. Estes podem residir numa base de dados espacial como *PostGIS*, permitindo variadas análises. A capacidade da *ML* repetir processos e produzir resultados exaustivamente, com uma eficácia semelhante, e por vezes superior à capacidade humana, traz possibilidades que podem resultar numa grande melhoria em termos de desempenho e produtividade dos Sistemas de Informação Geográfica (SIG). Os SIG são amplamente utilizados em várias disciplinas da Engenharia Geoespacial como produção cartográfica e deteção remota, permitindo a modelação, planeamento e visualização espacial do território.

As técnicas de deteção remota associadas à *ML* melhoram a capacidade de a Terra ser estudada [18]. As empresas de tecnologia da informação dependem desta tecnologia para atualizar, por exemplo, os seus serviços baseados na localização (*Location Based Services, LBS*). É legítimo afirmar que os governos também já as começam a utilizar para uma variedade de serviços públicos. Nos Estados Unidos da América, por exemplo, a classificação de cobertura do solo do *National Land-Cover Database* (*NLCD*) de 2001, foi produzida usando árvores de decisão [19], um tipo de *ML* supervisionada onde os dados são divididos continuamente de acordo com um determinado parâmetro.

Mais recentemente, e neste contexto de *ML*, surgem as *CNN*, uma classe de rede neural artificial do tipo *feed-forward*, com uma melhor capacidade de processamento e análise de imagens digitais de alta resolução adquiridas, por exemplo, com recurso a sensores espaciais, aéreos e baseados em VANT, como será abordado mais à frente. A última década testemunhou um *boom* de satélites e VANT equipados com sensores capazes de adquirir informações de alta resolução, fornecendo pela primeira vez um número extremamente elevado de imagens de quase todos os cantos da superfície da Terra. Os *data warehouses* destas imagens aumentam diariamente, incluindo imagens com diferentes resoluções espectrais e espaciais, pelo que se torna de extrema importância a implementação de ferramentas automáticas como as *CNN*, que permitam organizar e extrair informações deste tipo de dados.

2.3. Tipos de Sistemas de *ML*

É pertinente afirmar que na *ML*, há possibilidade de seguir um de dois caminhos: a aprendizagem não supervisionada e a aprendizagem supervisionada [15]. A principal diferença entre a supervisionada e não supervisionada é que a primeira usa dados rotulados para treinar o algoritmo, enquanto que a outra não [20]. No entanto, existem algumas nuances entre as abordagens.

No método não supervisionado, os algoritmos de *ML* analisam e agrupam autonomamente dados de entrada não rotulados, sendo o seu principal objetivo encontrar a estrutura e os padrões destes. Os modelos de aprendizagem deste tipo são utilizados para três principais tarefas: agrupamento, associação e redução de dimensionalidade. O agrupamento é uma técnica que, tal como o nome indica, tem como principal objetivo agrupar dados não rotulados, recorrendo a técnicas como o agrupamento *k-means*; muito útil para compressão de imagem, separabilidade espetral de classes, entre outros. A associação é outra das abordagens deste tipo e recorre a várias regras de forma a descobrir relações entre várias variáveis de um certo conjunto de dados; frequentemente aplicadas em otimização de motores de recomendação, como anúncios *web*. Por sua vez, a redução de dimensionalidade tem como objetivo principal reduzir o número de dados de entrada, mantendo a integridade dos dados; uma técnica muito utilizada na fase de pré-processamento de dados, por exemplo, para remoção de ruído visual dos dados de forma a melhorar a qualidade das imagens [20].

Por outro lado, a aprendizagem supervisionada é um método no qual os modelos são treinados usando dados rotulados. Os modelos precisam de encontrar a função de mapeamento entre uma variável de entrada (X) e a variável de saída (Y), classificando dados ou prevendo possíveis resultados, com grande precisão. Relativamente aos resultados, este método, pode ser dividido em dois tipos de problemas: classificação e regressão [20]. Na classificação é utilizado um algoritmo para atribuir os dados a categorias específicas; no mundo real podem ser usados, por exemplo, para classificar e identificar fogos em zonas de difícil acesso, com recurso à utilização de VANT [21]. Os algoritmos de classificação mais tradicionais e com grande utilização em aplicações ligadas à deteção remota são os classificadores lineares, máquinas vetor de suporte, árvores de decisão e algoritmos de floresta aleatória. Para além destes, a regressão usa um algoritmo para analisar a relação entre variáveis dependentes e independentes. Os modelos de regressão são úteis para prever valores numéricos com base em diferentes conjuntos de dados e os mais comuns são regressão linear, regressão logística e regressão polinomial [20].

Na aprendizagem supervisionada, o algoritmo "aprende" com o conjunto de dados de treino, faz previsões iterativas sobre os dados, ajustando-os para uma classificação correta [15]. Embora estes tipos de modelos tenham tendência a ser mais precisos do que os não supervisionados, exigem intervenção humana inicial para rotular os dados adequadamente, uma tarefa por vezes morosa. Por exemplo, um modelo de aprendizagem supervisionado pode prever onde existem casas, estradas e outras entidades geoespaciais numa certa imagem, mas primeiro, é necessário treiná-lo e identificar, em várias imagens de treino, o que são estradas ou casas de forma a que o modelo aprenda a identificá-las autonomamente. Os modelos de aprendizagem não supervisionados, por outro lado, trabalham por conta própria para descobrir a estrutura inerente aos dados não rotulados, sendo apenas necessário alguma intervenção humana para validar as variáveis de saída.

Podem ainda ser classificados, dentro do campo da *ML*, os dois tipos de algoritmos menos genéricos e baseados nos acima referidos: aprendizagem semisupervisionada e aprendizagem de reforço [15].

A aprendizagem semisupervisionada tenta melhorar o desempenho numa das duas tarefas genéricas (aprendizagem supervisionada ou não supervisionada) utilizando informações geralmente associadas entre a que foi escolhida e a outra. Normalmente utiliza-se para problemas de classificação

supervisionada em que os dados de treino são escassos. Neste tipo de casos, pode ser difícil construir um classificador supervisionado confiável sendo necessário introduzirem-se dados de treino classificados num ambiente não supervisionado. Os conjuntos de dados adicionais obtidos por aprendizagem não supervisionada, para os quais o rótulo era inicialmente desconhecido, podem ser assim usados para auxiliar no processo de treino. Esta situação ocorre em domínios de aplicações em que os dados rotulados manualmente são caros ou difíceis de obter [22].

Finalmente, na aprendizagem de reforço, o treino de modelos é realizado de forma a que estes consigam produzir uma sequência de decisões. O computador aprende a atingir um objetivo num ambiente incerto e potencialmente complexo. Nesta aprendizagem, enfrenta-se uma situação semelhante a um jogo. O computador emprega tentativa e erro para encontrar uma solução para o problema. Para a máquina realizar o que o programador deseja, o modelo recebe recompensas ou penalizações pelas ações que executa. O objetivo é que o modelo maximize a recompensa total [23].

2.4 O Que "Veem" os Computadores?

Um píxel (abreviação de *"picture element"* ou "elemento de imagem") é um pequeno quadrado digital de cor. Uma imagem digital é composta por uma matriz ou conjunto de píxeis. Para se visualizar uma imagem num computador esta tem de existir em formato digital, podendo ser captada com uma câmara digital ou digitalizada utilizando um *scanner*. Uma imagem digital é composta por uma matriz bidimensional preenchida por píxeis, dispostos em colunas e linhas. Cada píxel pode representar uma área na superfície da Terra no caso da aquisição aérea e espacial. Este tem um valor de intensidade e coordenadas de localização na imagem bidimensional associadas. O valor de intensidade representa a quantidade física medida, como a refração solar em determinada faixa de comprimento de onda refletida do solo, radiação infravermelha emitida ou intensidade de radar retrodispersa. Este valor é normalmente um valor médio referente a toda a área do solo coberta pelo píxel [24].

Devido à capacidade de armazenamento finita dos sistemas de computação, um número digital é armazenado com um número finito de *bits*. O número de *bits* determina a resolução radiométrica da imagem. Por exemplo, um número digital de 8 bits varia de 0 a 255. Obtendo uma imagem calibrada radiometricamente, o valor real da intensidade pode ser derivado do número digital do píxel. A localização de um píxel é indicada pelas coordenadas linha e coluna na imagem bidimensional. Pode haver uma associação entre a coluna-linha de um píxel e as coordenadas geográficas (por exemplo, longitude, latitude) do local-píxel da imagem, caso esta se encontre georreferenciada utilizando, por exemplo, no mínimo três pontos com coordenadas conhecidas no terreno e na imagem. Esta pode também ser obtida utilizando, por exemplo, equações de colinearidade da fotogrametria que associam as coordenadas da foto às do terreno através do conhecimento das orientações externas e internas da câmara utilizada.

A representação digital do número "8" da Figura 2.1, em termos de visão computacional, é constituída por uma matriz de píxeis e, como neste caso é uma imagem binária, cada um desses píxeis pode ser representado por um único número. São este o tipo de imagens normalmente usadas para rotular outras na fase de treino de uma *CNN*, sendo que os seus valores de "1" existem nas mesmas localizações dos objetos a classificar das correspondentes imagens coloridas, ou seja, são imagens identificadoras do tipo de entidades que o algoritmo deve aprender. Estes dois tipos de imagens (colorida e correspondente binária) normalmente possuem o mesmo nome e dimensões, de formar a ser possível a sua correta associação. São normalmente guardadas em pastas diferentes e são utilizadas nas fases treino e validação em algoritmos deste género.

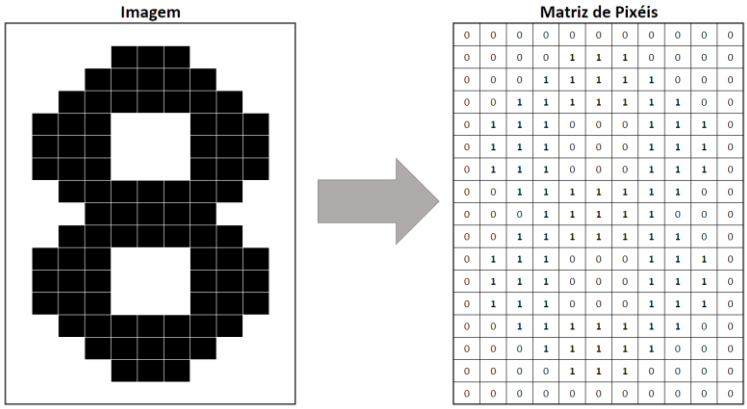


Figura 2.1 - Exemplo de imagem binária em formato de matriz

Caso seja uma imagem em escala de cinzentos, a interpretação é feita utilizando o mesmo método, mas os píxeis podem assumir vários valores, normalmente entre "0" e "255", que correspondem à intensidade da cor que cada píxel deve recrear.

Em termos de imagens multicor, cada píxel é normalmente definido dentro das três bandas de cor *RGB*: vermelho (*Red*), verde (*Green*) e azul *(Blue)*. Dentro de cada dimensão, tal como é o caso das imagens de escala cinza, são atribuídos valores aos píxeis, normalmente entre "0" e "255", de forma a que a composição destes represente uma imagem colorida (Figura 2.2).

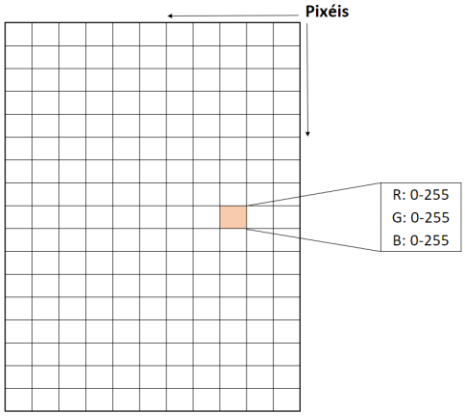


Figura 2.2 - Exemplo de imagem *RGB* em formato de matriz

Para um computador, em termos de IA, ser apenas capaz de ler uma imagem é inútil se não a conseguir interpretar, ou se não puder descrever o que retrata e contém. Um computador pode ser ensinado a classificar o conteúdo de uma imagem, um exemplo de *ML* em que se pode ensinar um computador a identificar e, por vezes, a descrever uma imagem. É semelhante à aprendizagem de um humano a identificar nomes de diferentes cores ou a diferenciar um gato de um cão, mostrando exemplos de cada caso, exatamente como um computador aprende a identificar objetos e caraterísticas numa imagem.

2.5 Algoritmos Tradicionais de Classificação de Imagens

Existem muitos algoritmos de classificação de imagens. Algoritmos diferentes têm resultados diferentes em problemas diferentes. Os algoritmos de classificação de imagens de *ML* tradicionais e os algoritmos de classificação de imagens de *DL* têm as suas próprias vantagens.

Sendo que este trabalho lida com um dos algoritmos de *DL* mais recentes de segmentação de imagem, torna-se importante que seja feita uma breve revisão de que outros precederam e de que forma conseguiram contribuir para aplicações ligadas à Engenharia Geoespacial. Para além das *CNN*, e no âmbito da classificação automática, podem ser apontados dois tipos de algoritmos deste tipo com bastante uso científico comprovado na área: as Florestas Aleatórias (*Random Forest*, *RF*) [25] e as Máquinas de Vetor de Suporte (*Support Vector Machines*, *SVM*) [26].

2.5.1 Máquinas de Vetor de Suporte (*SVM*)

As *SVM* são uma técnica de *ML* muito poderosa e versátil, capaz de realizar classificação linear ou não linear, regressão e até deteção de *outliers* [15]. É uma técnica de aprendizagem supervisionada, estatística e não paramétrica, não havendo qualquer tipo de suposições sobre a distribuição de dados subjacente [26].

O método é apresentado a um conjunto de instâncias de dados rotulados tendo depois como objetivo encontrar um hiperplano - limite de decisão que minimiza erros de classificação - que separe o conjunto de dados num número discreto e predefinido de classes, de forma a que as duas classes sejam separadas o mais longe possível das instâncias, consoante os exemplos de treino [27], como pode ser observado na Figura 2.3. Basicamente, pode-se afirmar que é um método onde a decisão sobre a associação de classe é feita usando apenas a similaridade entre as amostras de treino e teste. São classificadores binários lineares que atribuem a uma determinada amostra de teste uma classe de um de dois rótulos possíveis.

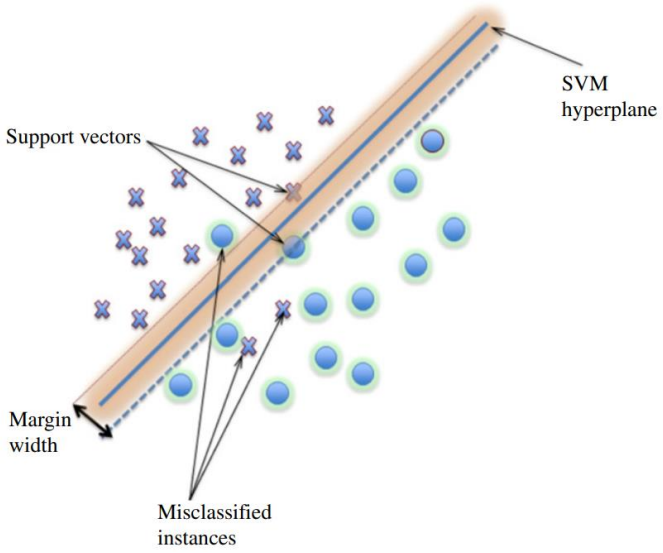


Figura 2.3 - Esquema base de *SVM* [26]

O seu grande uso no campo da deteção remota deve-se principalmente à sua capacidade de produzir bons resultados com pequenos conjuntos de dados de treino, muitas vezes produzindo maior precisão de classificação do que outros métodos tradicionais [27]. No entanto, a implementação de um *SVM*

assume que os dados de características multiespectrais são linearmente separáveis no espaço de entrada. Na prática, os pontos de dados de diferentes associações de classe podem sobrepor-se, o que dificulta a separabilidade linear, pois os limites básicos de decisão deste tipo geralmente não são suficientes para classificar padrões com alta precisão [26].

2.5.2 Floresta Aleatória (*RF*)

Por sua vez, o algoritmo de *RF* desenvolvido por Leo Breiman [28], é uma abordagem que se baseia num tipo de *ML* denominada de aprendizagem em conjunto, útil para resolver problemas de classificação e regressão [27]. A aprendizagem em conjunto refere-se à criação e combinação de múltiplos indutores para resolver uma tarefa de aprendizagem específica. A explicação intuitiva para a metodologia deste tipo de aprendizagem deriva da natureza humana e da tendência de reunir diferentes opiniões (indutores) e combiná-las para tomar uma decisão complexa. A ideia principal é que pesar e agregar várias opiniões individuais será melhor do que escolher a opinião de um único indivíduo [29].

Os métodos de aprendizagem em conjunto mais utilizados são o *Bagging* e o *Boosting* (Figura 2.4). O *Boosting* consiste num processo de construção de uma sequência de modelos de *ML*, onde cada modelo tenta corrigir o erro do anterior sequencialmente [27]. O *AdaBoost* foi a primeira abordagem deste género bem-sucedida, que foi desenvolvida para casos de classificação binária [30]. No entanto, o principal problema do *AdaBoost* era o *overfitting* do modelo [31], que significa que este pode apresentar um bom desempenho para os dados de treino, mas não generaliza bem pelo que a sua aplicação a outros conjuntos de dados exteriores apresentará maiores erros [15]. O *Bootstrap Aggregating*, conhecido como *Bagging*, é outro tipo de método de aprendizagem em conjunto, sendo projetado para melhorar a estabilidade e a precisão dos modelos em paralelo, enquanto reduz a variação [31]. Como tal, em termos de aprendizagem em conjunto, o *Bagging* é reconhecido por ser mais robusto contra o problema de *overfitting* em comparação com a abordagem de *Boosting* [27].

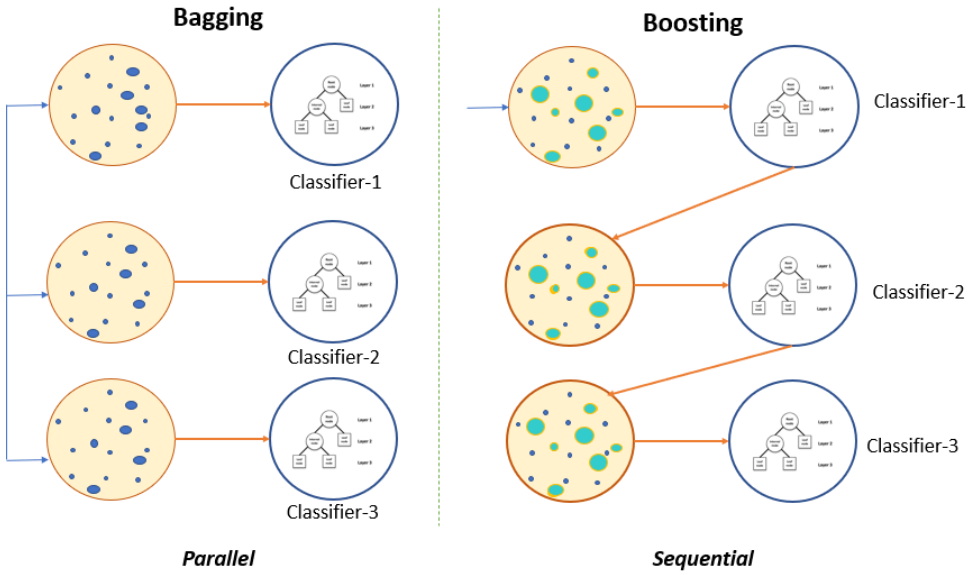


Figura 2.4 - Métodos de aprendizagem em conjunto *Boosting* e *Bagging* [30]

As *RF* foram a primeira abordagem bem-sucedida da utilização do *Bagging* [27]. O método combina a abordagem de *Bagging* [32], e a seleção aleatória de características, introduzidas independentemente por Ho [33, 34] e Amit e Geman [35], para construir uma coleção de árvores de decisão com variação

controlada. Cada árvore no conjunto atua como um classificador base para determinar o rótulo de classe de uma instância não rotulada.

Este processo é realizado via votação da maioria, onde cada classificador lança um voto correspondente ao rótulo de classe previsto, sendo o rótulo de classe com mais votos, usado para classificar a instância [35], como esquematizado na Figura 2.5.

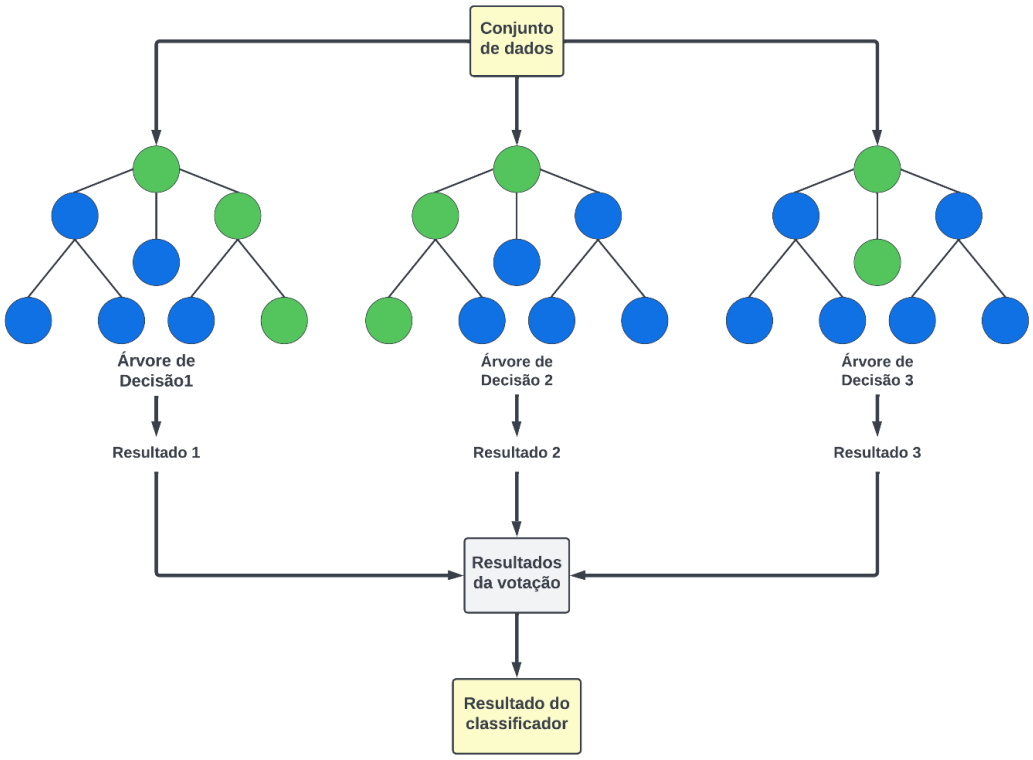


Figura 2.5 - Esquema base de *RF*

As *RF* ganharam popularidade na classificação de cobertura do solo devido ao seu processo de tomada de decisão claro e compreensível e excelentes resultados [36]. Outras vantagens do classificador *RF* podem ser resumidas da seguinte forma [27, 37]:

- Possibilidade de manipulação de milhares de variáveis de entrada sem exclusão de outras;

- Capacidade de reduzir a variância sem aumentar o erro das previsões;

- Capacidade de calcular proximidades entre pares de casos que podem ser posteriormente utilizados na determinação da localização de *outliers*;

- Capacidade de lidar facilmente com *outliers* e ruídos;

- Computacionalmente leve quando comparado com outros métodos de conjunto de árvores que utilizam *Boosting*.

Como tal, muitos trabalhos de pesquisa ilustraram a grande capacidade do classificador em deteção remota, por exemplo, para classificação de arquivos *Landsat* [27], classificação de imagens hiper e multiespectrais [38] e a classificação de imagens multiespectrais juntamente com dados de modelos digitais de elevação [39].

2.6. Redes Neurais Convolucionais (*CNN*)

As *CNN* são um conjunto de técnicas de aprendizagem profunda que têm sido aplicadas a tarefas visuais desde o final da década de 80. No entanto, apesar de algumas aplicações dispersas, estas permaneceram com pouca utilização até meados dos anos 2000, devido principalmente aos desenvolvimentos no poder de computação e ao surgimento de grandes quantidades de dados rotulados, que complementados com arquiteturas aprimoradas, contribuíram para o seu avanço que tem visto uma rápida progressão desde 2012, acompanhando essencialmente o rápido crescimento da componente gráfica computacional [40].

Desde o início, a ideia básica por detrás do funcionamento das redes neurais era imitar o funcionamento do cérebro humano ao mais alto nível possível. A *CNN* contribui para este objetivo, focando-se no funcionamento dos órgãos sensoriais visuais dos seres vivos e na sua capacidade de reconhecimento dos vários tipos de objeto visualizados, usando uma série de várias técnicas seguidas numa ordem específica [41]. David H. Hubel e Torsten Wiesel realizaram várias experiências em gatos em 1958 e 1959, de forma a tentarem perceber como funciona a estrutura geral do córtex visual [15]. Em particular, estes autores concluíram que muitos neurónios no córtex visual têm um pequeno campo recetivo local, o que significa que reagem apenas a estímulos visuais localizados numa região limitada do campo visual. Os campos recetivos de diferentes neurónios podem se sobrepor e, juntos, formam todo o campo visual. Além disso, os autores constataram que alguns neurónios reagem apenas a imagens de linhas horizontais, enquanto outros reagem apenas a linhas com diferentes orientações. Com as experiências realizadas também concluíram que alguns neurónios têm campos recetivos maiores e reagem a padrões mais complexos que são combinações dos padrões de nível inferior. Estes estudos do córtex visual inspiraram o *neocognitron*, introduzido em 1980, que evoluiu gradualmente para o que hoje chamamos de *CNN*.

Uma *CNN* pode ser dividida pelas várias camadas ou blocos de construção que em conjunto criam a rede como se pode observar no esquema exemplo da Figura 2.6. Estas são a camada de entrada, camada de convolução, camada de redução da amostra e camada totalmente conectada. Os neurónios entre as camadas são conectados por meio de pesos e vieses (*biases*).

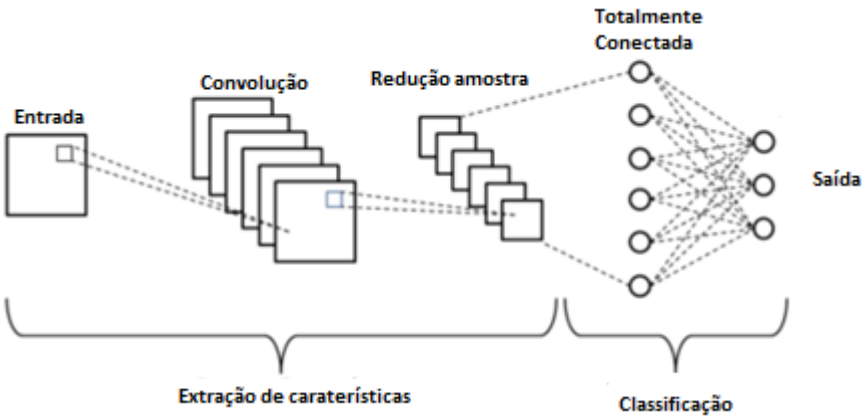


Figura 2.6 - Esquema base das *CNN*

Por outras palavras, uma *CNN* consiste em componentes empilhadas camada por camada [42] - a caraterística principal que lhe atribui o estatuto de *DL*. A camada de saída final gera resultados de classificação. A camada inicial é a camada de entrada, por exemplo dados de imagens aéreas, e a última é a camada totalmente conectada, com uma classificação prevista. No meio estão as camadas ocultas que transformam o espaço de recursos da entrada de uma maneira que corresponda à saída (camada de convolução e de redução da amostra) [43]. Assim a sequência base das *CNN* pode ser sintetizada em 4 etapas chave:

- A camada de entrada guarda os valores de píxel da imagem de entrada;

- A camada de convolução determinará a saída dos neurónios que estão conectados às regiões locais da entrada, através do cálculo do produto escalar entre os seus pesos e a região conectada ao volume de entrada;

- A camada de redução da amostra simplesmente realizará *downsampling* ao longo da dimensionalidade espacial da entrada fornecida, reduzindo ainda mais o número de parâmetros da ativação;

- As camadas totalmente conectadas tentam produzir pontuações para cada classe a partir das ativações, que são usadas para classificação.

Além do referido, pode-se afirmar que hoje em dia as *CNN*, como um tipo de *DL*, têm as seguintes vantagens sobre os modelos tradicionais de *ML* [42]:

- Aplicam diretamente uma operação de convolução aos píxeis de uma imagem para extrair recursos de dados abstratos, podendo ser aplicadas a vários cenários devido à sua capacidade poderosa de generalização;

- São capazes de representar a informação da imagem de maneira distribuída e adquirir rapidamente a informação da imagem de grandes volumes de dados, com a capacidade de resolver efetivamente problemas não lineares complexos (por exemplo, a rotação e a translação de uma imagem);

- São caracterizadas por conexões esparsas, compartilhamento de pesos e subamostragem espacial, o que resulta numa estrutura de rede mais simples e mais adaptável às variadas estruturas de imagem.

Em comparação com as imagens comuns, as imagens multiespectrais contêm informação espectral mais rica, bem como informação espacial que reflete a sua estrutura, forma e textura. A eficácia deste tipo de *DL* revolucionou as possibilidades de analisar padrões espaciais em dados de observação da Terra. Podem ser apresentados vários trabalhos que distinguem a grande capacidade para lidar com tarefas deste género. Uma revisão de Zhu et al. (2017) sobre os princípios gerais e potenciais de *DL* em deteção remota [44], Hoeser e Kuenzer, que realizam uma revisão geral aprofundada sobre arquiteturas para análise de dados de observação da Terra [45] e Reichstein et al., que fornecem perspetivas sobre como a *DL* pode avançar a ciência geográfica no geral [46].

Nas próximas seções, são exploradas as várias fases necessárias para a classificação de imagens utilizando *CNN*, nomeadamente a camada de entrada, camada de convolução, camada de redução da amostra e camada totalmente conectada. É também explicado de que forma se pode proceder à segmentação de imagem utilizando este tipo de *DL*. Inicialmente são abordados alguns projetos anteriores de maior sucesso na área da deteção e segmentação de imagens aéreas e/ou espaciais.

2.6.1 Exemplos de Aplicações em Engenharia Geoespacial

Nesta seção é feita uma revisão de alguns dos inúmeros projetos desenvolvidos que se consideram ser de maior interesse e relevância para a deteção remota e auxílio à produção cartográfica. Estes utilizam imagens essencialmente provenientes de VANT. As arquiteturas utilizadas em alguns projetos serão descritas em maior detalhe numa seção mais à frente, sendo necessário primeiro apresentar uma revisão mais completa do funcionamento das *CNN*.

2.6.1.1 Deteção

A deteção de objetos, a partir de imagens de obtidas por sistemas aéreos e espaciais, tem como principal objetivo localizar e discriminar os tipos de objetos existentes numa imagem (Seção 2.4.1). As aplicações deste género usam o método baseado em regiões candidatas. O método envolve três etapas: geração de regiões candidatas, extração de características pela *CNN* e classificação das regiões candidatas. As regiões candidatas são uma série de locais em que os objetos podem aparecer na imagem pré-gerada. Todos estes locais serão usados como entrada para a *CNN* para extração e classificação de recursos.

Para deteção aérea de forma a auxiliar na resposta a desastres Yalong, Nipun e Hamir [47] criaram um modelo com este objetivo, utilizando imagens aéreas captadas por VANT e com dados de treino adquiridos manualmente de vídeos de desastres, hospedados na plataforma *online Youtube*. Assim, os autores criaram um *dataset* que denominaram *Volan2018*, contendo oito vídeos dos furacões Harvey (Texas), Irma (Porto Rico), Maria (Flórida) e Michael (Flórida). Utilizando este *dataset* os modelos *CNN* foram então treinados, validados e testados em diferentes combinações de altitude do ponto de vista do sensor (baixa para drone *versus* alta para helicóptero) e o próprio equilíbrio do sistema de aquisição dos dados (equilibrado *versus* desequilibrado). Estes modelos foram posteriormente utilizados na deteção em tempo real, em vídeos obtidos por um VANT, *frame* a *frame*, das classes área inundada, resíduos, carros, vegetação, telhados danificados e telhados, utilizando a arquitetura *YOLO v2* para o propósito descrito e obtendo resultados como os apresentados na Figura 2.7. Em termos de validação de resultados, foram registadas precisões na ordem dos 70-80%.

Figura 2.7 - Exemplo de resultado de teste *YOLOv2* [47]

Além do referido, em termos de deteção utilizando VANT, pode ainda ser apontado um trabalho que utiliza uma arquitetura denominada *YOLO v3* para contabilização de automóveis em tempo real [48]. Desta forma, foi possível utilizar o modelo criado para monitorização de tráfego rodoviário em tempo real com precisões elevadas. Os autores deste artigo comparam também o desempenho da arquitetura escolhida com a *Fast R-CNN*, outro tipo de arquitetura utilizada para o mesmo âmbito, concluindo que a *YOLO v3* se destaca em termos de sensibilidade e processamento, apesar dos dois apresentarem precisões semelhantes. Um exemplo da sua testagem pode ser observado na Figura 2.8.

Figura 2.8 - Exemplo de resultado de teste *YOLOv3* [50]

2.6.1.2 Segmentação

A segmentação de imagem é também bastante utilizada para criar mapas temáticos das imagens, ou por vezes até como apoio à criação de mapas topográficos, dependendo da precisão dos dados. Para extrair objetos de uma imagem adquirida por um sensor aéreo ou espacial, é necessário segmentar os objetos de interesse na imagem e produzir um mapa de classificação de imagem a nível do píxel.

Uma arquitetura que se destaca pelo seu crescente uso em matéria de segmentação de imagens é a *U-net*. A tese de Sacadura [49] estuda esta arquitetura aplicando-a à segmentação de imagens multiespectrais de alta resolução para cartografia de uso do solo. O estudo teve como objetivo avaliar de uma forma geral a capacidade da arquitetura *U-net* para a segmentação de entidades geoespaciais que caracterizam o uso do solo nomeadamente em classes de telha vermelha, vias rodoviárias, edifícios industriais, culturas permanentes e caminhos agrícolas.

Por sua vez, existiu uma tentativa relacionada com a resposta a desastres, que passou pela segmentação de incêndios em imagens aéreas produzidas por VANT [50]. Foi tida em conta a importância de deteção de incêndios em zonas de difícil acesso utilizando os benefícios deste tipo de tecnologia aérea de aquisição remota de informação de terreno, de forma a que estes sejam efetivamente combatidos devido à rápida deteção, desenvolvendo de uma melhor capacidade de gestão do combate aos incêndios por órgãos de emergência como bombeiros. Os autores criaram um conjunto de dados de treino que denominaram *Flame Dataset* e que consiste na junção e organização de diferentes repositórios. Este inclui vídeos aéreos gravados por câmaras aéreas e também imagens térmicas adquiridas utilizando câmaras aéreas térmicas infravermelhas, por VANT durante a queima de várias pilhas de detritos numa floresta do Arizona, nos Estados Unidos da América. Para abordagens como *CNN*, este *dataset* é disponibilizado publicamente e, em termos de segmentação de incêndios, existem 39.375 imagens rotuladas ("Fogo" vs "Não Fogo") para a fase de treino. O treino deste foi realizado utilizando uma rede neural denominada *Xception*. Este método atingiu precisões de 92% com pontuações F1 de 88%. A Figura 2.9 demonstra quatro exemplos visuais de validação.

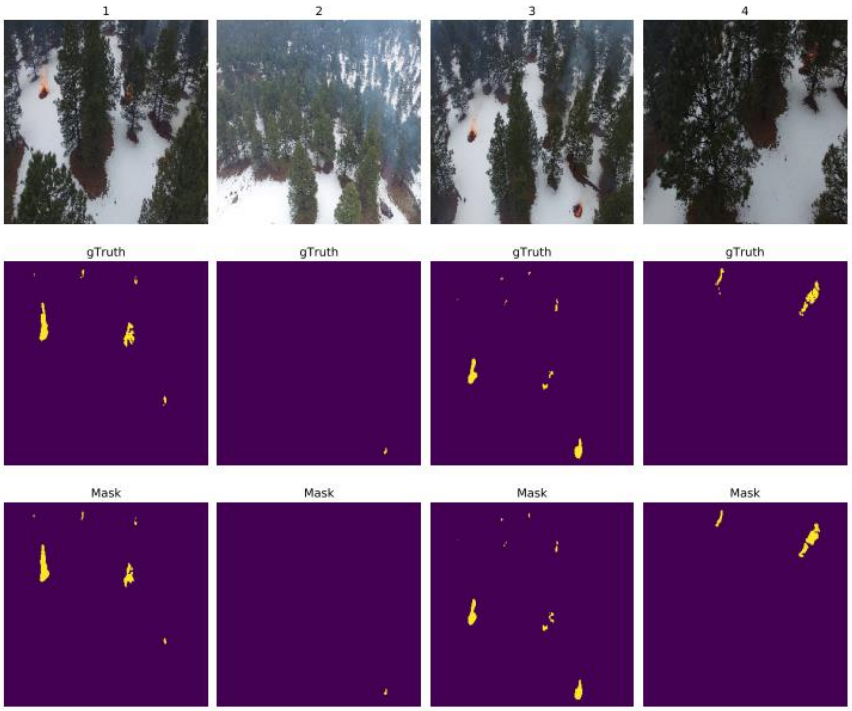


Figura 2.9 - Exemplo de resultado de validação *Xception* [50]

Existem várias outras arquiteturas que resultam da modificação da *U-net* e que por vezes acabam por atingir resultados melhores para determinados casos. Nomeadamente, um artigo que reporta o uso de uma Rede Convolucional denominada *Segnet* e utilizada pelos autores para segmentar edifícios de imagens aéreas de alta resolução. A experiência passou pela utilização de uma *nova* rede neural profunda chamada método *Seg–U-net*, que é uma composição das técnicas *Segnet* e *U-net*, para segmentar edifícios a partir de imagens aéreas de alta resolução. Os resultados obtiveram 92,73% de precisão no conjunto de dados de validação *Massachusetts*. Concluiu-se que a técnica proposta melhorou o desempenho ligeiramente para 0,44%, 1,17% e 0,14% em comparação com os métodos de rede neural totalmente convolucional (*FCN*), *Segnet* e *U-net*, respetivamente [51]. Recentemente o surgimento de outra arquitetura modificada *U-net* que também se pode apontar é a *C-U-net*, que melhora ligeiramente a extração de estradas de imagens de alta resolução [52].

2.6.2 Propagação dos dados na *CNN*

Antes de serem expostos termos mais específicos é importante referir de que forma se propagam os dados numa *CNN*, desde a camada de entrada até ao resultado final, também de forma a dar mais sentido à ordem a que se dispõem as seções seguintes. A propagação direta é a forma base das redes neurais criarem previsões. Os dados de entrada são "propagados em frente" pelas camadas ocultas da rede até a camada final, que gera uma previsão como se pode observar pelo esquema da Figura 2.10.

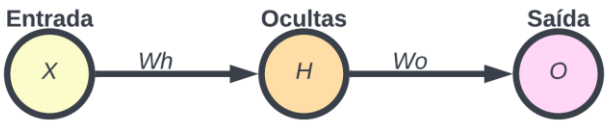


Figura 2.10 - Propagação direta

Para o exemplo básico rede neural da Figura 2.13, uma única passagem de propagação ou previsão (*O*) pode-se traduzir matematicamente em [53]:

$$O = A(A(X.Wh).Wo) \qquad [2.1]$$

Onde *A* é uma função de ativação como *ReLU*, *X* é a entrada e *Wh* e *Wo* são pesos. Nas próximas secções são abordados os conceitos de pesos e função de ativação.

2.6.3 Pesos e Vieses

O peso é o parâmetro dentro de uma rede neural que transforma os dados de entrada nas camadas ocultas da rede. Uma rede neural é composta por uma série de neurónios que recebem e produzem entradas de e para outros, sendo colecionados em, mapas de caraterísticas. À medida que uma entrada é recebida num neurónio, é multiplicada por um valor de peso e a saída resultante, influenciada por pesos, é observada ou passada para a próxima camada consoante a função de ativação. Muitas vezes, os pesos de uma rede neural estão contidos nas camadas ocultas da rede [54].

Por outro lado, o viés ou *bias* permite alterar a forma como o neurónio é interpretado pela função de ativação, resultando numa maior liberdade para o modelo atingir diferentes resultados. Enquanto os pesos permitem que uma rede neural artificial ajuste a força das conexões entre os neurónios, o viés pode ser usado para fazer ajustes dentro dos próprios neurónios [55]. O viés pode ser positivo ou negativo, aumentando ou diminuindo a saída de um neurónio. Um neurónio reúne e soma as entradas que recebe, adiciona o viés e só depois passa o resultado para a função de ativação (Figura 2.11).

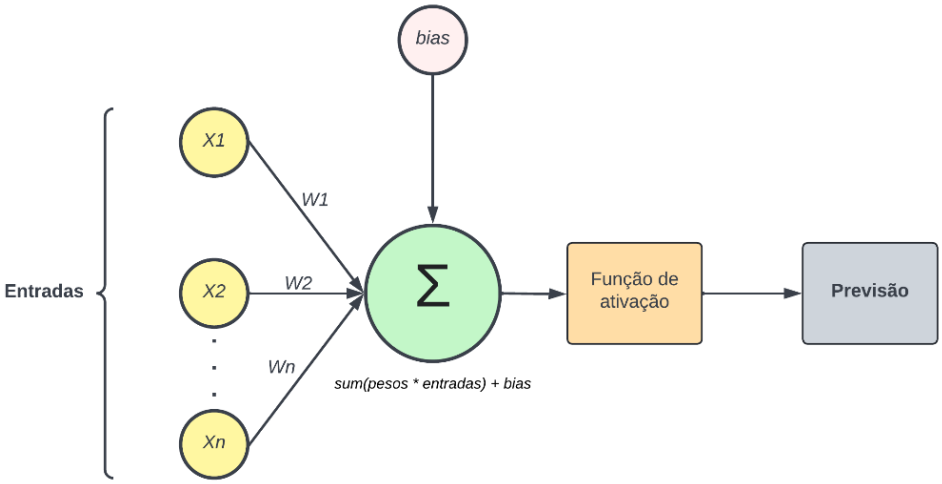


Figura 2.11 - Processamento realizado por um neurónio

O viés é um parâmetro adicional na rede neural que é usado para ajustar a saída juntamente com a soma ponderada das entradas para o neurónio. O processamento (*PN*) realizado por um neurónio pode ser representado pela equação:

$$PN = sum(pesos.entradas) + viés \qquad [2.2]$$

Uma rede neural seleciona aleatoriamente os valores de peso e de vieses antes que a aprendizagem comece. À medida que o treino avança, ambos os parâmetros são ajustados para os valores que melhor representam os dados de treino - utilizando o chamado gradiente descendente, descrito mais à frente. Os dois parâmetros diferem na extensão da sua influência sobre os dados de entrada. Os pesos podem ser

pensados como a força da conexão, afetando a quantidade de influência que uma mudança na entrada terá sobre o valor de saída. Por outro lado, o viés pode ser definido como o erro da previsão médio do modelo no conjunto de treino, pelo que um valor alto sugere que este tem uma baixa capacidade de previsão, acabando por influenciar/ajustar mais o resultado produzido por um neurónio quando comparado com um valor baixo.

2.6.4 Camada de Convolução

O bloco de construção mais importante de uma *CNN* é a chamada camada de convolução. Os neurónios na primeira camada desta estão conectados apenas aos píxeis respetivos aos seus campos recetivos [15], normalmente equivalente ao tamanho do *kernel* aplicado (Figura 2.12, *input layer*). Por sua vez, cada neurónio na segunda camada de convolução está conectado apenas a neurónios localizados dentro de um pequeno retângulo na primeira camada (Figura 2.12, *Convolutional layer 1*). Assim, esta arquitetura permite que a rede se concentre em pequenos recursos de baixo nível na primeira camada oculta, agrupando-os em recursos de maiores dimensões e de nível superior na camada oculta seguinte, e assim por diante [15]. Esta estrutura hierárquica, representada na Figura 2.12, é semelhante ao procedimento realizado pelo humano na identificação visual de objetos, que é uma das razões pelas quais as *CNN* apresentam bons resultados na extração de informação das imagens.

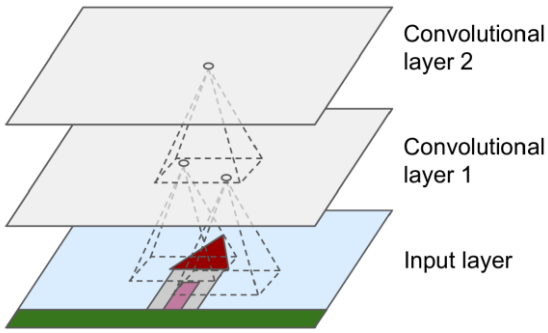


Figura 2.12 - Camada de convolução [15]

A operação de convolução funciona numa pequena área local da imagem com um certo tamanho de um *kernel* convolucional. O *kernel* convolucional é uma matriz de pesos que pode ser aprendida - o chamado filtro. A saída da camada de convolução passa por uma função de ativação, sendo obtido um mapa de características [56]. As camadas de convolução incluem vários filtros ou pesos otimizáveis que transformam a entrada ou as camadas ocultas anteriores. O número de filtros define a profundidade de uma camada de convolução. As transformações resultantes visam revelar padrões decisivos para o problema em questão, sendo aprendidos iterativamente através da convolução, que será explorada em detalhe na seção seguinte.

O resultado final é uma nova camada de produtos escalares para cada filtro, também chamada de mapa de características. Os primeiros mapas de caraterística de uma *CNN* podem incluir padrões relativos a partes da imagem da camada de entrada como cantos, círculos ou bordas. Assim, é possível concluir que nas camadas de convolução, os mapas de características da camada anterior são alvos da operação de convolução com recurso a *kernels* apreensíveis e submetidos a uma certa função de ativação para formar o mapa de características de saída.

2.6.4.1 Filtragem espacial e convolução

No processamento de imagens digitais, as operações de filtro são usualmente aplicadas a uma imagem executando uma operação espacial chamada convolução e recorrendo a uma matriz chamada *kernel*. Os valores armazenados no *kernel* estão diretamente relacionados aos resultados da aplicação do filtro, e os filtros são caracterizados apenas pela matriz de *kernel* associada. O caso bidimensional da operação de convolução sobre uma imagem digital pode ser definido como a ação de deslizamento de uma matriz *kernel* sobre uma matriz da imagem, uma porção de cada vez, com a soma dos produtos elementares das duas matrizes como resultado. Esta operação pode ser definida matematicamente:

$$y[i,j] = x[i,j] * h[i,j] = \sum_{j=1}^{M} \sum_{i=1}^{N} x[i,j].h[m-i,n-j] \qquad [2.3]$$

Onde, *x* representa a matriz da imagem de entrada, de dimensão (*M* x *N*), alvo de convolução com a matriz do *kernel h*, resultando numa matriz *y*, a imagem de saída. Os índices *i* e *j* estão relacionados com as matrizes de imagem enquanto os de *m* e *n* com o *kernel*.

As definições matemáticas de convolução são indicadas entre as matrizes pelo operador "*", como pode ser observado na Equação 2.3. Basicamente, numa ótica de processamento digital de imagem e *DL*, a convolução pode ser vista como a aplicação de vários filtros, que são usados para alterar a imagem, mapeando as várias caraterísticas que a constituem e definem, para mais tarde serem usadas em processos de classificação de imagem. No entanto, é importante denotar que no caso de uma imagem composta por três bandas, como é o caso das imagens *RGB*, por exemplo, a operação funciona da mesma maneira, mas com a diferença de ter de ser executada três vezes, uma para cada banda.

2.6.4.2 Preenchimento e passo na operação de convolução

Nesta seção serão explorados dois termos que podem condicionar os resultados das camadas de convolução.

Um problema desafiante ao aplicar este tipo de camadas é que existe a tendência de se perderem píxeis no perímetro de imagem alvo da operação. Como normalmente são utilizados *kernels* de pequenas dimensões, podem-se perder apenas alguns píxeis, mas esta discrepância pode aumentar à medida que são aplicadas camadas convolucionais sucessivas e cada vez mais profundas. Uma solução direta para este problema passa por adicionar píxeis com valor "0" ao redor da imagem (Figura 2.13), aumentando assim as dimensões da mesma - o chamado preenchimento. Este pode ser ainda dividido em preenchimento *SAME* - quando o tamanho de saída da operação é o mesmo que o da entrada, requerendo que o filtro tenha capacidade de ser aplicado para além da imagem com recurso a preenchimento - e *VALID* - quando o filtro se mantém numa posição valida na imagem diminuindo as dimensões da imagem e sem recurso a preenchimento do exterior desta [57].

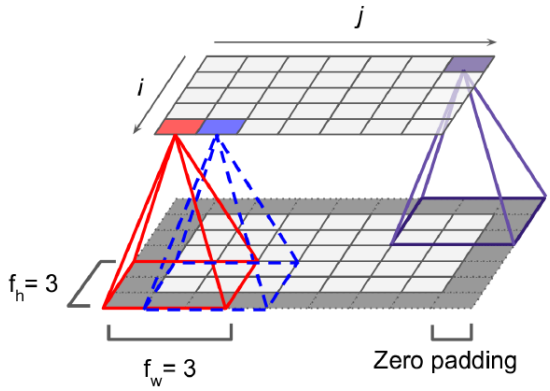


Figura 2.13 - Preenchimento na operação de convolução [15]

Por outro lado, o passo ou *stride*, é um parâmetro do filtro da rede neural que modifica a intensidade da quantidade de movimento sobre a imagem. Por exemplo, se o passo de uma rede neural for definido como "2", o filtro move-se à distância de dois píxeis entre os dois centros sucessivamente (Figura 2.14). O tamanho deste parâmetro afeta também o volume de saída codificado, sendo geralmente definido como um número inteiro. [57]

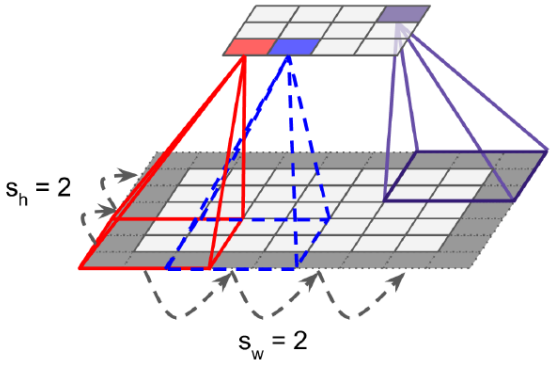


Figura 2.14 - Passo na operação de convolução [15]

2.6.4.3 Mapas de caraterísticas

Os mapas de caraterísticas são os produtos escalares, dispostos em matriz, resultantes das várias operações de convolução aos dados de entrada.

Uma camada de convolução tem vários filtros que são aplicados à camada de entrada, criando um mapa de caraterísticas por cada um, para que seja representado com mais precisão em 3D. Para fazer isso, existe um neurónio por píxel em cada mapa de características, e todos os neurónios dentro de um determinado mapa de características compartilham os mesmos parâmetros de pesos e vieses. [15]

Em suma, uma camada de convolução aplica simultaneamente vários filtros de treino à sua camada de entrada, tornando-a capaz de detetar várias caraterísticas que ficam registadas nos chamados mapas de caraterísticas. Ficam assim disponíveis os resultados a ser processados na camada de redução da amostra.

Matematicamente, as dimensões (*o* x *o*) do mapa de caraterísticas'resultante, para as dimensões de um certo *kernel (k x k)*, preenchimento *p*, passo *s* e dimensões de entrada *(i x i)* pode ser definido como:

$$o = \frac{i+2p-k}{s} + 1 \qquad [2.4]$$

2.6.5 Função de Ativação

Imaginando que uma certa pessoa tem de decidir se vai sair de sua casa apenas baseando-se em certas condições ou valores de entrada, por exemplo, o tempo, a hora do dia e a disposição, por ordem de importância. Esta ordem de importância pode ser vista como os pesos, em que "3" seria o mais alto e "1" o mais baixo para o fator da disposição, por exemplo. No caso de uma classificação binária se o tempo e a hora estivessem como desejado seria marcado com "1" e se a disposição não estivesse boa seria marcada com "0", sendo que os pesos atribuídos a cada fator são multiplicados pelo valor binário decidido correspondente. Assim, após somados os valores obtidos de cada um dos produtos, obter-se-ia um valor que representava a vontade da pessoa sair de casa com base nos fatores influentes (tempo, hora e disposição) e nas suas respetivas implicações (pesos ou filtros) para a decisão final. Seria depois definido uma função que indicasse se o resultado era suficientemente significativo para levar a pessoa sair ou não de casa (ativar ou não o neurónio). Em tal caso, o valor de saída seria "1", ativando o neurónio, no sentido de propagação de informação entre neurónios de diferentes camadas.

Assim, a função de ativação refere-se aos nós que são colocados entre as redes neurais e ajudam a decidir que neurónios devem ser ativados, podendo ser usada para resolver problemas não lineares ou abstratos. Basicamente, uma função de ativação é usada para aumentar a capacidade de expressão e abstração do modelo de rede neural, ajudando a melhorar a precisão dos seus resultados, sendo normalmente colocada entre camadas de convolução. É uma espécie de limiar que determina um valor mínimo para se considerar aquando da determinação de caraterísticas a uma camada de entrada, determinando a possibilidade de cada neurónio continuar a fazer parte do processo de classificação, sendo a sua intensidade determinada pelos seus pesos e vieses [43].

Esta função é o núcleo da estrutura de uma rede neural profunda e as mais comuns são: *sigmoid, Tanh, ReLU* e *softplus.* [58]

2.6.5.1 Sigmoid

A função de ativação *sigmoid*, representada no gráfico da Figura 2.15, traduz o intervalo de entrada para o intervalo [0; 1].

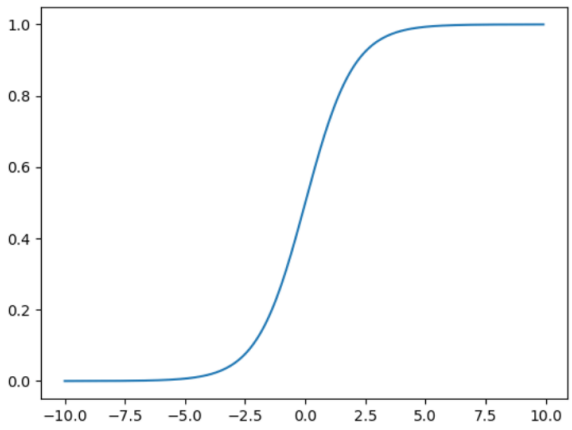


Figura 2.15 - Função *sigmoid*

É não linear por natureza e tem uma derivada suave. Devido à faixa de saída do *sigmoid* [0;1], a saída de cada unidade de entrada *x* é reduzida, fazendo com que o gradiente se anule especialmente em redes de *DL*. Pode ser definida matematicamente como:

$$f(x) = \frac{1}{1+e^{-x}} \qquad [2.5]$$

Onde *e* é o número de *Euler*, uma constante matemática.

2.6.5.2 Tanh

A função *Tanh* representada no gráfico da Figura 2.16 traduz o valor de entrada para o intervalo [-1; 1]. Quanto maior a entrada, mais próximo o valor de saída estará de "1", e quanto menor, mais próxima a saída estará de "-1".

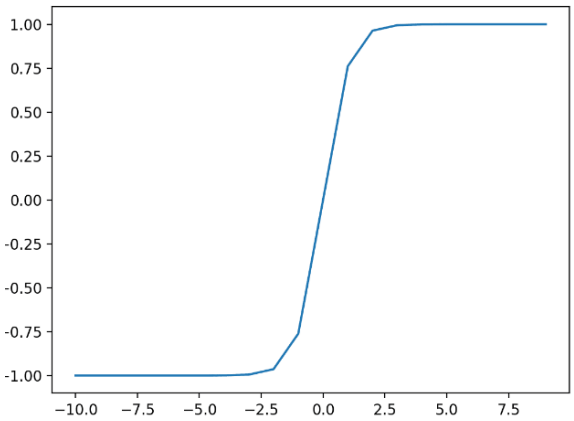


Figura 2.16 - Função *Tanh*

A função *Tanh* é idêntica à função *sigmoid* mas com diferente contradomínio, sendo uma função simétrica centrada em zero. Os valores podem ser calculados utilizando a seguinte fórmula:

$$Tanh(x) = \frac{(1-e^{-2x})}{(1+e^{-2x})} \qquad [2.6]$$

2.6.5.3 ReLU

O *ReLU* é a função de ativação mais usada em termos de *DL*. O sucesso é baseado nos relatos do seu desempenho de treino superior [59] em relação a outras funções de ativação, como *sigmoid* e *tanh.* A função pode ser matematicamente definida da seguinte forma:

$$ReLU(x) = \max(0, x) \qquad [2.7]$$

A função é linear para todos os valores positivos e zero para todos os valores negativos, como representado no gráfico da Figura 2.17. No entanto, os valores de saída relativos a valores de entrada positivos podem variar de zero a infinito. Uma das vantagens de usar o *ReLU* é a sua simplicidade.

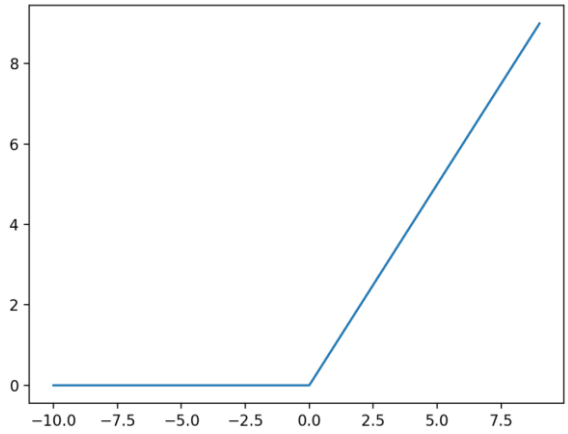


Figura 2.17 - Função *ReLU*

As arquiteturas *CNN* mais conhecidas empilham algumas camadas de convolução, cada uma geralmente seguida por uma camada *ReLU*, depois uma camada de redução da amostra, depois mais algumas camadas de convolução também acompanhadas individualmente com *ReLU*, e depois outra camada de redução da amostra, e assim por diante. A desvantagem do *ReLU* é que não permite aprendizagem em exemplos para os quais a ativação resulta em zero. Pode acontecer, por exemplo, quando há um grande gradiente e um neurónio com *ReLU* pode acabar com um peso e viés negativo. Os pesos deste neurónio nunca mais são atualizados pelo que se costuma denominar neurónio morto [60].

2.6.5.4 Softplus

A função *softplus*, é semelhante à função *ReLu*. A diferença entre a função *softplus* e a função *ReLu* pode ser vista claramente na Figura 2.18, especialmente nos valores de entrada negativos e próximos de zero. Em comparação, é mais dispendiosa computacionalmente, mas ajuda a resolver o problema dos neurónios mortos.

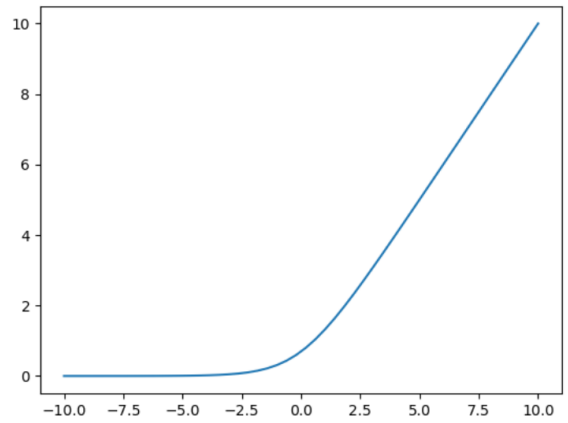


Figura 2.18 - Função *softplus*

Pode ser formulada matematicamente da seguinte forma:

$$Softplus(x) = \ln(1 + e^{x}) \qquad [2.8]$$

2.6.6 Camada de Redução da Amostra

As camadas de redução da amostra de uma *CNN* são também bastante importantes e utilizadas para transformar os dados provenientes da camada de entrada, tal como é o caso das camadas de convolução. Fazem estas parte, juntamente com as de convolução, das camadas ocultas da rede que transformam o espaço de recursos da entrada de uma maneira que corresponda à saída, como já foi referido.

A redução da amostra é um passo importante para reduzir ainda mais as dimensões do mapa de caraterísticas, mantendo apenas os recursos importantes e reduzindo a invariância espacial. Isso, por sua vez, reduz o número de recursos que podem ser aprendidos [40]. O objetivo principal é a criação de uma subamostra da saída da convolução, para reduzir a carga computacional, o uso de memória e o número de parâmetros, limitando assim o risco de *overfitting* [15].

A técnica de redução da amostra *Max Pooling*, por exemplo, seleciona os valores mais altos, resultantes da operação de convolução de uma submatriz, normalmente com dimensões (2 x 2), sobre o mapa de caraterísticas (Figura 2.19). Desta forma é criada uma matriz separada a partir da utilização desta técnica, onde residem os valores mais altos registados em cada sobreposição da submatriz com o mapa de caraterísticas. Assim, garante-se que os recursos apreensíveis permanecem limitados em número, preservando também os principais recursos de qualquer imagem independentemente da sua localização [41]. Existe também o *Mean Pooling* que atribui o valor médio de todos os elementos do mapa de caraterísticas que calham dentro da passagem da submatriz.

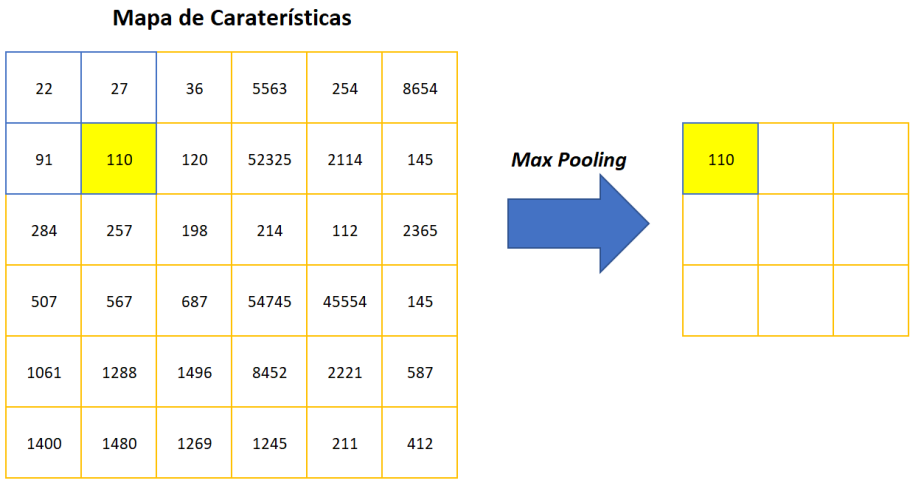


Figura 2.19 - Operação *Max Pooling*

As camadas de convolução e de redução da amostra são assim responsáveis pela extração de recursos a diferentes níveis de abstração. Por exemplo, os filtros na primeira camada podem detetar bordas horizontais, verticais e diagonais. Os filtros na próxima camada podem detetar formas, e os filtros na última camada podem detetar uma coleção de formas. Os valores de filtro das *CNN* são inicializados aleatoriamente e são aprendidos pelo algoritmo. Estas nuances tornam a *CNN* muito poderosa, porque além de realizarem a classificação, concede-lhes também a capacidade de extração automática de recursos. Assim distingue-se de outras técnicas de classificação, como *SVM*, que não podem, por exemplo, realizar extração de recursos.

2.6.7 Camada Totalmente Conectada

Até aqui foi possível constatar que as camadas de convolução e de redução da amostra, geralmente são empilhadas umas sobre as outras de forma a tornar possível a extração de representações de recursos mais abstratas, à medida que as camadas de entrada se movem pela rede.

As camadas totalmente conectadas, que seguem estas camadas ocultas na hierarquia da rede, interpretam estas representações de recursos [40]. Após a extração de recursos, é necessário classificar os dados em várias classes. Esta ação é habitualmente realizada utilizando estas camadas, sendo também possível, mas menos usual, utilizar-se um classificador convencional como *SVM* (Seção 2.5.1) [60].

As camadas totalmente conectadas aprendem uma função, normalmente não linear, entre os recursos de alto nível fornecidos como saída das camadas ocultas [60]. Esta é a camada final que faz parte da *CNN* comum. Geralmente a matriz da camada anterior, que corresponde a uma matriz 2D, é transformada para o formato 1D antes de ser passada para os neurónios. Todo o raciocínio e computação em dados é normalmente realizado nesta camada, sendo o seu principal objetivo a classificação destes.

2.6.8 Processo de Treino das *CNN*

O processo de treino de uma *CNN* pode ser bastante complexo devido à necessidade de existirem enormes quantidades de conjuntos de dados com boa qualidade, necessários para um algoritmo aprender caraterísticas e conseguir posteriormente obter bons resultados, especialmente nas fases de teste e validação. Assim, nesta seção são exploradas as técnicas mais importantes em termos de controlo de qualidade e garantia do melhor treino possível, tanto em termos de eficiência como em termos de resultados produzidos.

2.6.8.1 Função de perda

A função de perda de uma *CNN* quantifica a diferença entre o resultado esperado e o resultado produzido pelo modelo de *ML*. A partir desta função, é possível serem derivados os gradientes que são usados para atualizar os pesos. A média sobre todas as perdas constitui o chamado custo [60]. Em termos de treino, esta métrica é bastante importante e pode também ser divida em perda em treino e perda em validação.

A perda em treino é uma métrica usada para avaliar como o modelo se ajusta aos dados de treino, ou seja, avalia o erro das previsões do modelo no conjunto de treino e consiste na aplicação de uma função de perda ao longo dos vários lotes que dividem os *epochs*. Por sua vez, a perda em validação é uma métrica usada para avaliar o desempenho de um modelo num conjunto de validação [61]. O conjunto de validação é normalmente definido como sendo uma parte do conjunto de dados de treino, mas que não entra na aprendizagem, e à qual é aplicada uma função de perda ao longo dos vários *epochs* que constituem a fase de treino do modelo (deixando de fazer parte do conjunto inicial de dados de treino).

A comparação de resultados da perda em treino com resultados da perda em validação pode ser bastante útil na determinação de *overfit* ou *underfit* do modelo. Quando os valores da perda em validação e os valores da perda em treino se encontram simultaneamente altos e distantes, pode-se concluir que o modelo se encontra *underfit*, apresentando maus resultados de previsão no conjunto de treino, indicando que o modelo precisa de mais *epochs* ou de mais dados de treino. Por outro lado, uma perda em validação bastante superior a uma perda em treino pode indicar *overfit*, pelo que o modelo prevê bem no conjunto de dados de treino, mas em dados exteriores apresenta dificuldades (conjunto de dados de validação),

podendo ser utilizadas técnicas de regularização que ajudam a atenuar este problema como será abordado mais à frente.

Algumas das funções de perda que se consideram de maior importância na fase de treino de algoritmos de classificação são: *binary cross entropy*, *categorical cross entropy* e *hinge loss*.

A perda de entropia cruzada, ou *cross entropy*, mede o desempenho de um modelo de classificação através de um valor de probabilidade que varia entre zero e um. Este valor de perda de entropia cruzada aumenta à medida que a probabilidade prevista diverge do rótulo real [62]. As tarefas de classificação que têm apenas dois rótulos para a variável de saída são chamadas de problemas de classificação binária, enquanto os problemas com mais de dois rótulos são chamados de problemas de classificação categórica ou multiclasse [63]. Na classificação binária, onde o número de classes *M* é igual a "2", a entropia cruzada *b (binary cross entropy)* pode ser calculada como:

$$b = -(ylog(p) + (1-y)\log(1-p)) \qquad [2.9]$$

Se *M>2*, ou seja, classificação multiclasse *(categorical cross entropy)*, calcula-se uma perda separada para cada rótulo de classe e somamos o resultado:

$$c = -\sum_{c=1}^{M} y_{o,c}\log(p_{o,c}) \qquad [2.10]$$

Existe ainda a chamada *sparse cross entropy*, que se pode distinguir da *categorical* pelo facto de usar rótulos inteiros para descrever cada classe, ao invés de um conjunto de números. É definida pela mesma Equação 2.10, pelo que a única diferença passa pela maneira como os rótulos são definidos. [64]

Outra função de perda comumente usada para classificação é a *hinge loss*, sendo principalmente aplicada em *SVM* para calcular a margem máxima do hiperplano para a divisão das classes. A dobra da função penaliza amostras mal classificadas e classifica corretamente aquelas que estão dentro da margem definida ao longo do limite de decisão. Esta margem é a margem máxima do hiperplano para os pontos de dados, razão pela qual a *hinge loss* é preferida para *SVM* [65].

2.6.8.2 Gradiente Estocástico Descendente (SGD)

Os pesos e vieses da rede são geralmente otimizados usando o algoritmo de gradiente descendente como já foi referido anteriormente. O termo gradiente descendente implica a minimização progressiva de erros ao longo de uma inclinação (gradiente).

A descida do gradiente é realizada em iterações ao longo do treino, nas quais as previsões de um modelo com parametrização imediata são comparadas com as anotações dos dados de treino usando uma função de perda (*categorical cross entropy*, *binary cross entropy*, entre outras). As convoluções são baseadas em filtros inicializados aleatoriamente, a atribuição de observações em lotes é aleatória e o gradiente descendente tem uma natureza aleatória, portanto, neste caso é conhecido como gradiente estocástico descendente (*Stochastic Gradient Descent, SGD*) [42].

O *SGD* é derivado usando o algoritmo de retropropagação. Dada uma rede neural com uma camada de entrada, uma camada de saída e *n* camadas ocultas, entre elas o algoritmo de retropropagação calcula o gradiente da função de perda em relação aos pesos e vieses das camadas ocultas. Este gradiente é então usado para avaliar e atualizar os pesos e vieses do modelo através do gradiente descendente, ou seja, tentando encontrar um mínimo global no espaço de características de alta dimensão, como pode ser visualizado na Figura 2.20.

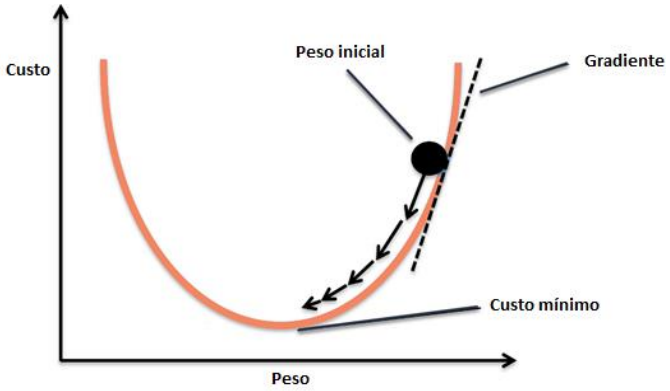


Figura 2.20 - Descida do gradiente

O procedimento de descida de gradiente é realizado para várias amostras, seguido pela média dos pesos e desvios calculados das camadas ocultas [66]. Os pesos e vieses atualizados podem posteriormente ativar ou não os neurónios consoante os seus valores e função de ativação escolhida, como já foi referido anteriormente (Seção 2.6.5), deixando de ser ativados os que vão sendo identificados como causa dos erros de classificação ao longo das várias iterações - a chamada otimização utilizando o *SGD*.

2.6.8.3 Lotes e epochs

A fase de treino de uma *CNN* é computacionalmente muito intensa, pois as variáveis explicativas são ricas em dimensões, resultando numa infinidade de mapas de caraterísticas que retratam diferentes especificidades espaciais e contextos em escalas variadas. Estas variáveis (imagens) resultam em quantidades enormes de dados a serem processados durante esta fase, especialmente considerando que o treino do modelo pode exigir milhares de amostras para memorizar os recursos decisivos da classe ou valor alvo. Estes volumes de dados podem não ser compatíveis com a memória do nosso sistema, um problema que surgiu neste projeto e que foi suavizado com recurso à computação em nuvem, como será descrito mais à frente.

Além de técnicas que melhorem o *hardware* onde se treinam os dados, esta ação de treino pode também ser realizada sequencialmente em lotes que compreendem apenas uma parte de todo o conjunto de dados. Os pesos e tendências do modelo são atualizados com base num gradiente médio para todo o lote. Separar o conjunto de dados em lotes permite treinar o modelo iterativamente até que ele tenha visto todas as amostras, o que é chamado de *epoch*. O número de iterações para terminar um *epoch* é, portanto, o número total de observações dividido pelo tamanho do lote.

O número de lotes e *epoch* é definido antes de se dar início ao treino do algoritmo, sendo que o primeiro pode causar grandes variações de uso de memória *VRAM* e *RAM* do computador, consoante o valor utilizado.

2.6.8.4 Regularização do treino

O *overfitting* é um problema comum de treino como já foi já foi explorado na Seção 2.6.8.1. Nesta seção será explorado o conceito de regularização e de que forma as técnicas associadas a esta podem resolver este problema, nomeadamente o *dropout* e *early stopping*.

A regularização é uma técnica que faz pequenas modificações no algoritmo de aprendizagem de modo que o modelo generalize melhor. Através da sua aplicação, o desempenho do modelo em dados exteriores ao treino melhora - dados de teste ou validação. As redes neurais profundas, normalmente têm dezenas de milhares de parâmetros, às vezes até milhões e, por esta razão, a rede acaba por ganhar uma liberdade incrível, tendo a capacidade de acomodar uma enorme variedade de conjuntos de dados complexos. Apesar disso, esta grande flexibilidade cria um ambiente propício ao *overfitting* do conjunto de treino [15].

O *dropout* é uma técnica de regularização que foi proposta por Geoffrey Hinton em 2012, e é explicada em grande detalhe por Srivastava et al. [67]. Esta apresenta-se como sendo uma das mais interessantes técnicas de regularização, produzindo excelentes resultados e destacando-se pela sua grande popularidade. Basicamente, em cada etapa de treino, cada neurónio têm uma probabilidade *p* de ser temporariamente excluído do processamento (Figura 2.21), o que significa que será totalmente ignorado durante o treino, mas pode voltar a ser ativado noutra etapa [68]. O hiperparâmetro *p* é chamado de taxa de abandono e representa a probabilidade de um neurónio ser removido na fase de treino. Ao utilizar o *dropout*, a mesma camada altera a sua conectividade, procurando assim caminhos alternativos para transmitir as informações à próxima camada [69]. Esta técnica é bastante utilizada em arquiteturas de segmentação como é o caso da *U-net*, que será abordada em detalhe mais é frente.

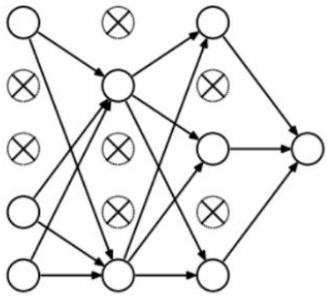


Figura 2.21 - *Dropout* de neurónios

Outra técnica de controlo de *overfitting* que pode ser apontada é o *early stopping*. Uma vez selecionado o esquema de avaliação do modelo, um gatilho que seja capaz de parar o processo de treino deve ser escolhido. Este usa uma métrica de desempenho, como uma função de perda, que decide quando interromper o treino. Geralmente é monitorizado o desempenho do modelo no conjunto de dados de validação utilizando umas das funções de perda referida Seção 2.6.8.1. No caso mais simples, o treino é interrompido assim que o desempenho no conjunto de dados de validação diminui em comparação com o desempenho no conjunto de dados de validação no *epoch* anterior. Este pode também ser definido para ter uma certa paciência, que corresponde ao número de *epochs* com valores inferiores aos anteriores que este deve ter em conta antes de parar o treino.

2.6.8.5 Função Softmax

A regressão logística produz um valor decimal entre “0” e “1”. Por exemplo, uma saída de regressão logística de 0,8 de um classificador de email sugere 80% de chance de um email ser *spam* e 20% de probabilidade de não o ser. Claramente, a soma das probabilidades de um e-mail ser spam ou não é “1”, mas em termos de problemas multiclasse a soma poderá não ser um valor normalizado. Assim, a função *softmax* atribui probabilidades decimais a cada classe em problemas multiclasse, de forma a que

a sua soma seja igual a “1”. A restrição imposta por esta função ajuda o treino a convergir mais rapidamente.

A *softmax* transforma um vetor de valores reais - normalmente o vetor 1D resultante da camada totalmente conectada - num vetor de valores reais cuja soma é igual a “1”. Os valores de entrada são transformados pela *softmax* em valores entre “0” e “1”, para que possam ser interpretados como probabilidades (Figura 2.22). Se uma das entradas tiver valores pequenos ou negativos, é transformada numa probabilidade baixa e, se tiver valores grandes, é transformada numa probabilidade alta [15].

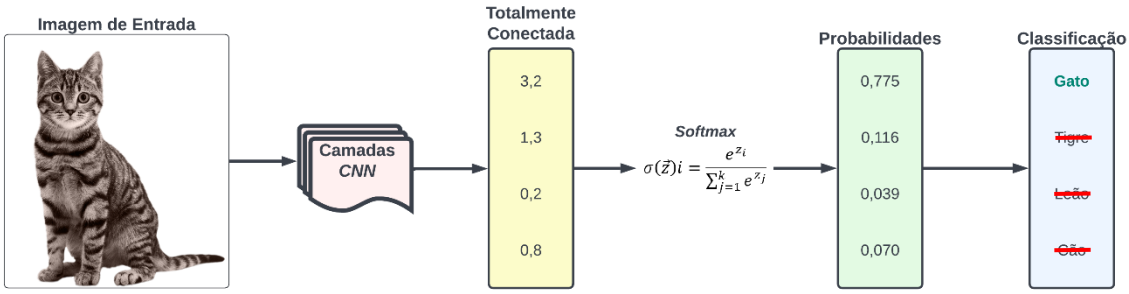


Figura 2.22 - Esquema da aplicação da função *Softmax*

A Equação 2.11 define a formulação matemática da *softmax*.

$$\sigma(\vec{z}_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \qquad [2.11]$$

Nesta, *z* é o vetor de entrada da função, *K* representa o número de classes do classificador multiclasse, z_i são os elementos do vetor de entrada *K* e e^{zi} é uma função exponencial aplicada a todos os elementos de forma a que estes sejam sempre positivos. Finalmente o termo do denominador $\sum_{j=1}^{k} e^{z_j}$ é o chamado termo de normalização que garante que a soma dos todos os valores de saída da função seja igual a “1”, constituindo assim uma distribuição de probabilidade válida para problemas de classificação multiclasse.

2.7 Segmentação Semântica com *CNN*

A primeira tentativa de segmentação semântica utilizando *CNN* pode ser apontada para o ano 2015 [70]. Os autores mostraram que uma rede totalmente convolucional (*FCN*), em tarefas de segmentação semântica, atinge resultados impressionantes. Este é o primeiro trabalho a treinar *CNN* ponta a ponta para previsão *pixelwise* e a partir de aprendizagem supervisionada. Permitiu também avanços na deteção de objetos, previsão de partes e pontos-chave [71] e correspondência local [72].

Neste artigo, os autores afirmam também a possibilidade de substituir as camadas densas - camadas que já receberam entradas de outras - no topo de uma *CNN* por camadas de convolução para realizar previsões a nível do píxel (Figura 2.23) - daí o nome “totalmente convolucionais”.

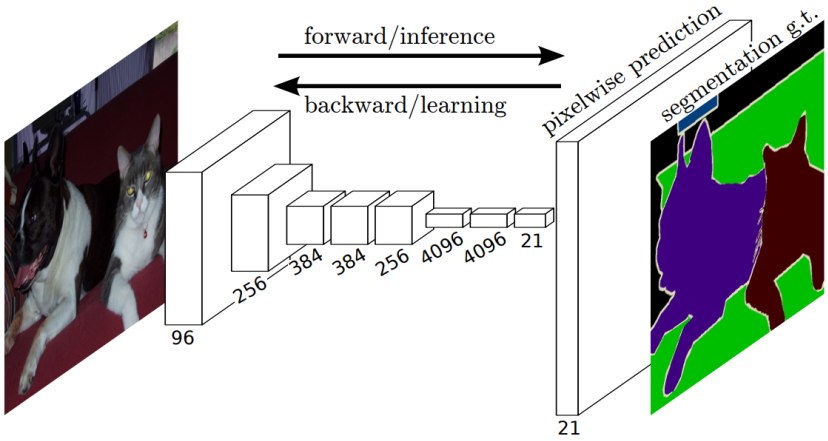


Figura 2.23 - *CNN* para segmentação de imagem [70]

Esta afirmação pode ser esclarecida com recurso ao exemplo exposto em [15]: supondo que uma camada densa com duzentos neurónios fique no topo de uma camada de convolução, que gera cem mapas de caraterísticas, cada um de tamanho (7 × 7). Cada neurónio calculará uma soma ponderada de todas as (100 × 7 × 7) ativações da camada de convolução. Se a camada densa for substituída por uma camada de convolução usando duzentos filtros, cada um (7 × 7) (mesmas dimensões do mapa de caraterísticas de entrada) e com preenchimento *VALID*, serão também produzidos duzentos mapas de caraterísticas, cada um (1 × 1). Por outras palavras, a camada de convolução produzirá duzentos números, assim como produz a camada densa. Este facto é apontado como sendo de extrema importância porque enquanto uma camada densa espera um tamanho de entrada específico, uma camada de convolução processará com prazer imagens de qualquer tamanho, mas com o número específico de canais ou bandas. Como uma *FCN* contém apenas camadas de convolução e camadas de redução da amostra, que têm a mesma propriedade, pode ser treinada e executada em imagens de qualquer tamanho.

No entanto, os resultados inicialmente obtidos após a segmentação utilizando *FCN* não eram suficientemente bons, relativamente distorcidos e suaves, não sendo totalmente sensível a detalhes na imagem [73]. Mais tarde, tentativas como a de Ronneberger et al. [74] que propôs uma arquitetura denominada *U-Net*, inicialmente com o objetivo de segmentar imagens biomédicas, mostraram grandes desenvolvimentos e melhorias nesta área, como será abordado.

2.7.1 Camada de Convolução Transposta

Nesta parte apresenta-se brevemente o conceito de convolução transposta, uma camada que pode ser utilizada para reversão de operação de redução espacial da camada de entrada - camadas ocultas de convolução e redução da amostra - que se tornou bastante importante em arquiteturas de segmentação mais recentes, especialmente na que se utiliza nesta tese.

As camadas *CNN* que vimos até agora, como camadas de convolução e camadas de redução da amostra, normalmente reduzem as dimensões espaciais (altura e largura) da entrada ou mantêm-nas inalteradas. Na segmentação semântica, que classifica a imagem a nível do píxel como já foi abordado, será conveniente que as dimensões espaciais de entrada e saída sejam as mesmas, por exemplo, a capacidade de um píxel de saída conter os resultados de classificação para o píxel de entrada na mesma posição espacial, sendo usualmente utilizadas convoluções transpostas para o efeito.

Considerando como exemplo o mapa de caraterísticas de entrada de tamanho (2 x 2) exposto na Figura 2.24, que precisa de ver as suas dimensões aumentadas para (3 x 3) utilizando este método, a operação utilizada por esta camada para atingir este objetivo pode ser descrita pelos seguintes passos [75]:

1) Cria-se um *kernel* (2 x 2) com as mesmas dimensões e elementos que o mapa de caraterísticas, e não se associa qualquer tipo de *padding* ao mapa de caraterísticas original;
2) Multiplicam-se todos os elementos do *kernel* pelo elemento superior esquerdo do mapa de caraterísticas, preenchendo os quatro elementos resultantes numa matriz com tamanho equivalente ao mapa de saída desejado (3 x 3) e na posição correspondente (neste caso canto superior esquerdo);
3) Utilizando a mesma lógica realiza-se a mesma operação para os restantes elementos do mapa de caraterísticas, multiplicando-os sempre por todos os elementos do *kernel* e tendo em conta a posição de cada um na matriz (3 x 3) resultante da operação;
4) No final resultam quatro matrizes (3 x 3), uma para cada elemento do mapa de caraterísticas, pelo que os elementos que se sobrepõem entre elas são adicionados, ficando assim criado um mapa de caraterísticas com as dimensões aumentadas, como exemplificado na Figura 2.24.

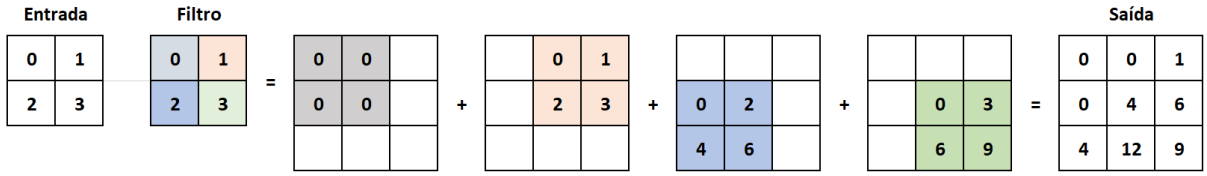


Figura 2.24 - Operação de convolução transposta

O termo deconvolução e convolução transposta são frequentemente confundidos um com o outro. Muitas fontes usam os dois de forma intercambiável e, embora existam deconvoluções, não são utilizadas com o mesmo objetivo que as convoluções transpostas. Uma deconvolução é uma operação matemática que reverte o efeito da convolução, pelo que valores de entrada numa convolução regressam ao seu estado inicial. Por outro lado, uma camada de convolução transposta apenas reconstrói as dimensões espaciais da entrada. Na teoria, favorável em termos de *DL*, pois pode aprender os seus próprios parâmetros por meio do *SGD*, não fornecendo, no entanto, a mesma saída que a entrada.

2.8 Arquiteturas *CNN*: Aplicações e Caraterísticas

Nesta seção serão exploradas algumas arquiteturas *CNN* que podem ser úteis para os variados propósitos de aquisição de informações provenientes de imagens, nomeadamente: classificação, deteção e segmentação. Assim, o objetivo passa por disponibilizar exemplos materializados de arquiteturas *CNN* que tem várias aplicações, algumas já referidas na Seção 2.6.1 e com ligação a disciplinas como a deteção remota, especialmente no que toca à deteção e segmentação de entidades geoespaciais.

2.8.1 Arquitetura *LeNet-5*

A arquitetura *LeNet-5* é talvez a arquitetura *CNN* mais reconhecida. Foi criada por LeCun et al. em 1998 e exposta no artigo *Gradient-Based Learning Applied to Document* Recognition [76], sendo amplamente utilizada para reconhecimento de dígitos manuscritos.

A rede possui 5 camadas (Figura 2.25) com parâmetros que podem ser aprendidos (camadas ocultas), daí o nome *Lenet-5*. Possui três conjuntos de camadas de convolução com duas camadas de redução da

amostra média entre si. Após as camadas ocultas, existem ainda duas camadas totalmente conectadas. Ainda de referir a implementação da *softmax* na última camada de redução da amostra que classifica as imagens nas respetivas classes e a utilização da função de ativação *tanh* na outra e nas de convolução [77].

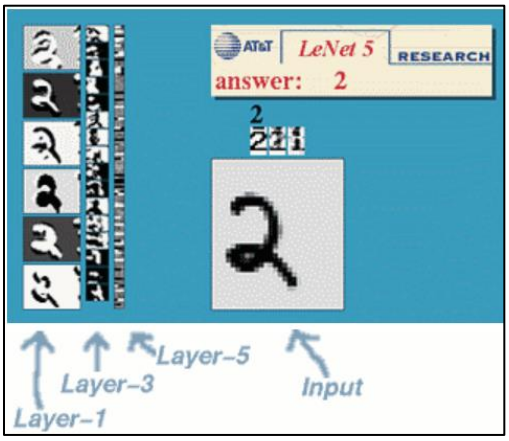


Figura 2.25 - Arquitetura *LeNet-5* [76]

Assim, esta arquitetura pode ver o seu procedimento sintetizado em 8 passos consecutivos [78]:

1) Camada de entrada de dimensões (32 x 32) e com apenas uma banda;
2) Camada de convolução com seis filtros de tamanho (5 x 5) e *stride* de um. A função de ativação utilizada é a *tanh*. O mapa de caraterísticas de saída é (28 x 28 x 6);
3) Camada de redução da amostra média com tamanho de filtro (2 x 2) e passo um. O mapa de caraterísticas resultante é (14 x 14 x 6);
4) Camada de convolução com 16 filtros de (5 x 5) e passo um. Além disso, a função de ativação é *tanh*. O tamanho de saída é (10 x 10 x 16);
5) Camada de redução da amostra média de (2 x 2) com passo dois. Como resultado, o tamanho do mapa de caraterísticas foi reduzido para (5 x 5 x 16);
6) Camada final de convolução possui cento e vinte filtros de (5 x 5) com passo um e função de ativação *tanh*. O tamanho de saída é 120;
7) Camada totalmente conectada com oitenta e quatro neurónios que resultam na saída para 84 valores e a função de ativação usada aqui é novamente *tanh*;
8) Camada de saída com dez neurónios e função *softmax*.

A camada de saída desta arquitetura é pouco usual sendo importante referi-la. Em vez de calcular a multiplicação da matriz das entradas e o vetor de peso, cada neurónio emite o quadrado da distância euclidiana entre o seu vetor de entrada e o seu vetor de peso. Cada saída mede o quanto a imagem pertence a uma determinada classe de dígitos. A função de custo de entropia cruzada é utilizada, penalizando muito mais as previsões erradas, produzindo gradientes maiores e convergindo mais rapidamente para os melhores valores [15].

2.8.2 Arquitetura *Xception*

Uma variante da arquitetura *GoogLeNet* que merece destaque é a *Xception*, proposta em 2016 por François Chollet [79], que substitui os módulos *inception* por um tipo especial de camada chamada convolução separável em profundidade.

Enquanto uma camada de convolução usa filtros que tentam capturar simultaneamente padrões espaciais e padrões entre as várias bandas de uma imagem, uma camada de convolução separável supõe que padrões espaciais e bandas da imagem podem ser modelados separadamente, como pode ser visualizado na Figura 2.26.

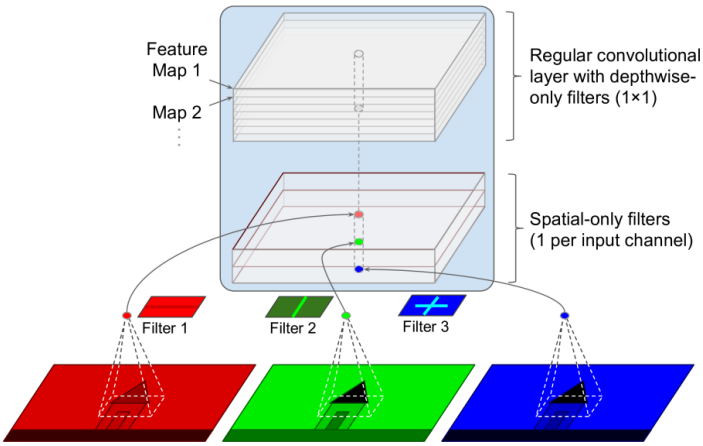


Figura 2.26 - Camada de convolução separável [15]

A arquitetura *Xception* possui trinta e seis camadas de convolução formando a base de extração de recursos da rede como pode ser visualizado na Figura 2.27. Destas, duas camadas de convolução são regulares, mas as restantes são convoluções separáveis, além de algumas camadas de redução da amostra máxima. A classificação de imagem pode ser realizada utilizando uma camada final de regressão logística e uma camada totalmente conectada, após a aplicação de uma camada de redução da amostra média.

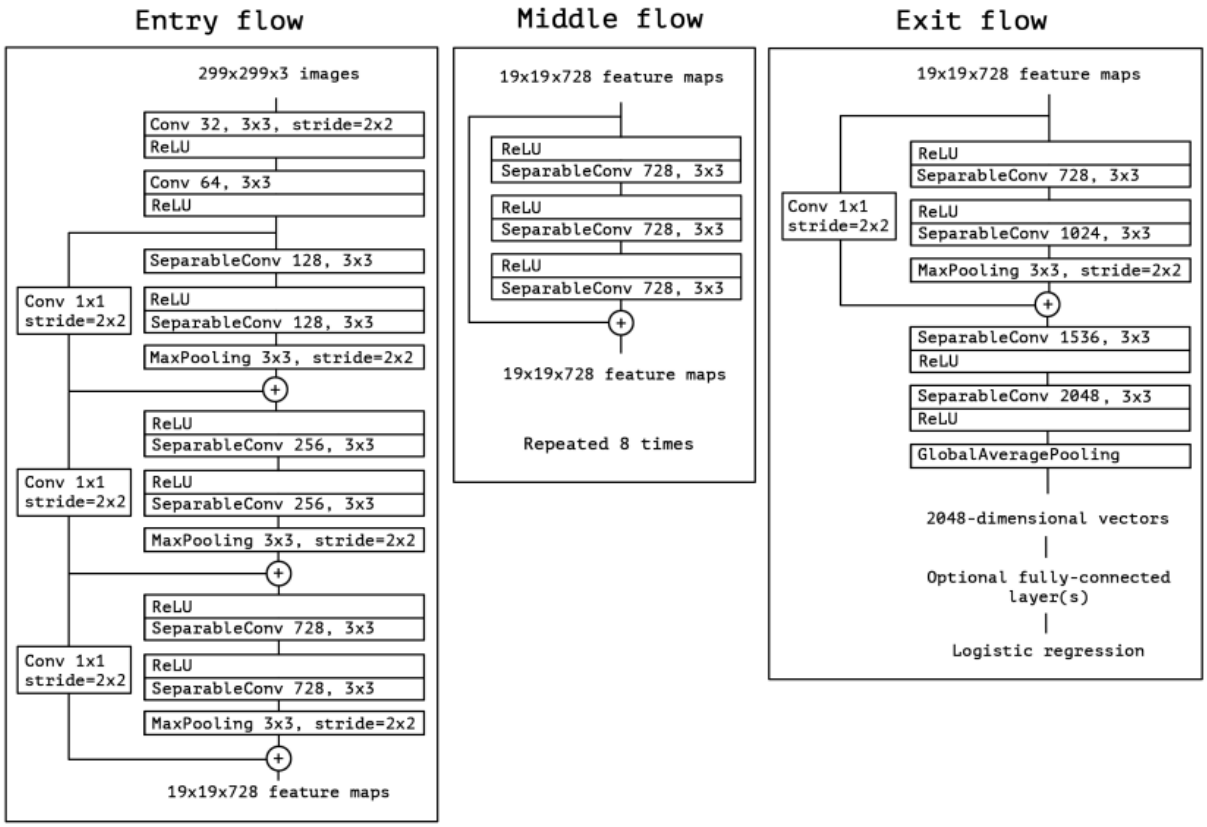


Figura 2.27 - Arquitetura *Xception* [79]

2.8.3 Arquitetura *Segnet*

Em 2015 Badrinarayanan et al. apresentaram uma nova arquitetura *CNN* para segmentação de imagens a nível do píxel, a *Segnet* [80].

Na arquitetura *Segnet* não há camadas totalmente conectadas, ou seja, faz parte da família das *FCN*. Um descodificador realiza o *upsample* da entrada usando índices de redução da amostra transferidos de um codificador, produzindo um mapa de caraterísticas esparso. De seguida, é realizada uma convolução com um conjunto de filtros treinável para densificar o mapa de caraterísticas.

A arquitetura possui uma rede codificadora e uma rede descodificadora correspondente (Figura 2.28), seguida por uma camada final de classificação píxel a píxel. A primeira consiste em treze camadas de convolução que correspondem às treze primeiras camadas de convolução da rede *VGG16* [81] projetada para classificação de objetos. São descartadas camadas totalmente conectadas em favor da retenção de mapas de caraterísticas de maior resolução na saída do codificador, reduzindo também o número de parâmetros da *SegNet*. Cada camada do codificador tem uma camada descodificadora correspondente e, portanto, a rede do descodificador tem treze camadas tal como a outra. A saída final do descodificador é alimentada a um classificador *softmax* para produzir probabilidades de classe para cada píxel independentemente (Seção 2.6.8.5).

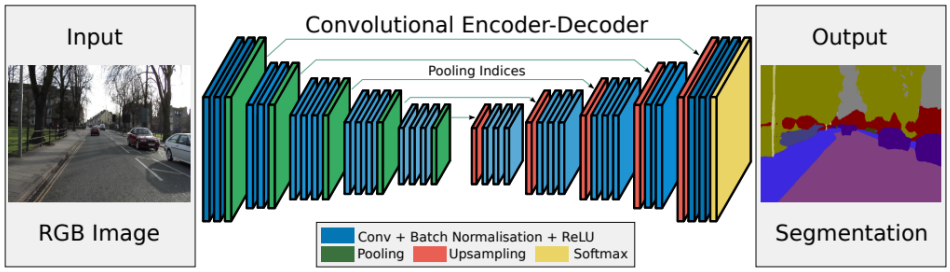


Figura 2.28 - Arquitetura *Segnet* [80]

2.8.4 Arquitetura *U-net*

A arquitetura *U-net* é uma das principais arquiteturas utilizada para classificação de imagens a nível do píxel - segmentação semântica. A arquitetura foi proposta por Ronneberger et al. em 2015 [74] e é constituída por um caminho de contração que captura o contexto das imagens e um caminho de expansão simétrico que permite a localização precisa de elementos na imagem (Figura 2.30). Foi provado pelos autores que tal arquitetura pode ser treinada ponta a ponta a partir de muito poucas imagens, superando, na competição ISBI para segmentação microscópica de imagens biomédicas, um dos melhores métodos anteriores, as redes de convolução de janela deslizante.

A arquitetura pode ser representada por um esquema em formato “U”, tal como se pode observar pela Figura 2.29. Esta é constituída por um caminho de contração (lado esquerdo) e por um caminho expansivo (lado direito), sendo que o primeiro segue a arquitetura típica da *FCN*.

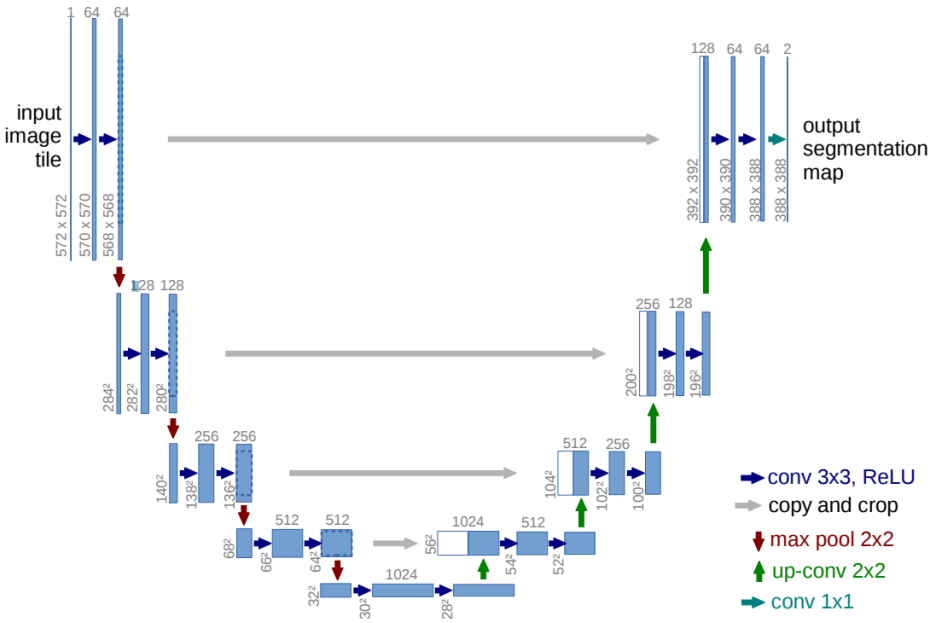


Figura 2.29 - Arquitetura *U-net* [74]

A contração consiste na aplicação repetida de duas convoluções (3 x 3), sem qualquer preenchimento associado à imagem de entrada, cada uma seguida por *ReLU* e uma operação de redução da amostra máxima (2 x 2) com passo "2". Em cada etapa de redução da amostra, os mapas de caraterísticas são duplicados.

Por outro lado, cada passo no caminho expansivo consiste no *upsampling* do mapa de caraterísticas utilizando uma convolução transposta (2 x 2) que reduz em metade o número de mapas de caraterísticas, uma concatenação com o mapa de caraterísticas correspondentemente recortado do caminho de contratação e duas convoluções (3 x 3), cada uma seguida por *ReLU*. O recorte é necessário devido à perda de píxeis ao longo das bordas dos mapas de caraterísticas em cada convolução. Na camada final, uma convolução (1 x 1) é usada para mapear cada um dos sessenta e quatro vetores de características produzidos para o número desejado de classes. No total, a rede possui vinte e três camadas de convolução [74].

3 Metodologia

Neste capítulo é apresentada a metodologia proposta para a segmentação da entidade geoespacial "edifício" em imagens de alta resolução. Assim, é importante apresentar neste a origem e especificidades dos conjuntos de dados adquiridos na *internet* bem como as técnicas desenvolvidas para tratamento, criação e compilação dos mesmos. É também descrito de que forma se procedeu à implementação das três fases de treino, validação e teste dos modelos criados, expondo e descrevendo os vários conjuntos de dados utilizados, programas *Python* criados e experiências de treino e validação realizadas.

3.1 Dados de Treino

O principal objetivo deste trabalho é estudar o impacto do uso de dados com caraterísticas *big data*, disponíveis e partilhados por outros autores na *internet*, para segmentação automática de edifícios utilizando *CNN*. Nesta seção é realizada uma descrição dos *datasets* que foram utilizados, nomeadamente o *Inria Aerial Image Labeling Dataset*, *Waterloo Building Dataset* e o *Massachusetts Buildings Dataset*.

O conjunto de dados *Inria Aerial Image Labeling* foi publicado por Maggiori et al. em 2017 [82] num projeto que pretendeu responder à pergunta "Os métodos de segmentação semântica podem ser generalizados para qualquer cidade?". Possui uma cobertura de 405km², imagens aéreas ortorretificadas com uma resolução espacial de 0.3m e imagens *ground truth* (máscaras) dos limites dos edifícios correspondentes como se pode observar no exemplo da Figura 3.1. As imagens que constituem este conjunto de treino correspondem a cinco localizações distintas com diferentes densidades populacionais que podem ajudar na generalização do modelo: Tyrol e Vienna na Austria e Kitsap, Chicago e Austin nos Estados Unidos da América. Foi dado o nome *Inria* às experiências que recorreram apenas a este conjunto ao longo do trabalho, como será abordado em detalhe na próxima seção. Os dados podem ser descarregados gratuitamente na página *web* do projeto [83].

Figura 3.1 - Exemplo de imagens de treino do conjunto de dados *Inria Aerial Image Labeling* [83]

Por sua vez, o *Waterloo Building Dataset*, que foi publicado em 2017 por He et al. [84], contem 117000 imagens (256 x 256 x 3) que cobrem parte da área de Waterloo no Canadá com uma resolução espacial de 0,12m e uma cobertura de cerca de 206 km2. Este é disponibilizado *online* e gratuitamente na plataforma de partilha de dados *Harvard Dataverse* [85].

Finalmente, o *Massachusetts Buildings Dataset* [86] contem 151 imagens (1500 x 1500 x 3) da área de Boston nos Estados Unidos da América e pode ser descarregado da plataforma *Kaggle* [87]. A totalidade das imagens cobre uma área de cerca de 340 km2. A grande quantidade de dados rotulados disponibilizados pelos autores deve-se ao facto de a administração da cidade de Boston contribuir com a regular atualização dos polígonos relativos a edifícios no projeto aberto *OpenStreetMap*. O conjunto de dados abrange principalmente áreas urbanas e suburbanas e edifícios de variados tamanhos e faz uso de imagens divulgadas pelo estado de *Massachusetts*. Todas as imagens são redimensionadas para uma resolução de "1" píxel por metro quadrado.

3.2 Tratamento de Dados e Experiências

Nesta seção são apresentados e descritos os quatro modelos experimentais criados com os dados de treino adquiridos. É também descrito o processo de tratamento dos dados utilizados para os treinar e de que forma se criou um conjunto de dados de treino utilizando imagens de zonas do território nacional.

3.2.1 Experiências

O treino utilizando a arquitetura *U-net* foi realizado com recurso a várias abordagens diferentes ou experiências, pelo que se torna importante descrever em pormenor nesta seção quais as suas principais diferenças. Para isso, as informações que se seguem têm como principal objetivo sintetizá-las e disponibilizar acesso aos seus recursos.

Como se pode observar na Tabela 3.1, as primeiras experiências de modelos de segmentação de edifícios passaram pela utilização individual do *Inria Aerial Image Labeling Dataset* - experiência *Inria* - e pela combinação deste com os outros dois descritos anteriormente: *Waterloo Building Dataset* e *Massachusetts Buildings Dataset* - experiência *Full*. Desta forma, pretende-se avaliar a contribuição da mistura de *datasets* com caraterísticas diferentes, tanto em termos de resolução como em termos de escala e textura, de forma a ser criado um conjunto mais abrangente e com maior diversidade de dados.

Tabela 3.1 - Síntese das experiências realizadas

Nome da experiência	**Descrição**	**Detalhes**	**Objetivo**	***Links***
Inria	Treino com imagens recortadas com dimensões (256 x 256) provenientes do *Inria Aerial Image Labeling*.	- Treino com arquitetura *U-net*; - 44928 imagens normalizadas e respetivas máscaras; - Imagens guardadas em 18 partes de 2496 imagens.	Avaliar o desempenho do modelo na segmentação de imagens com localizações diferentes daquelas para as quais foi treinado.	*Drive*
Full	Treino com combinação dos três *datasets* adquiridos *online* com dimensões (256 x 256) normalizadas.	- Treino com arquitetura *U-net*; - 106848 imagens normalizadas e respetivas máscaras; - Imagens guardadas em 42 partes de 2544 imagens.	Avaliar o desempenho da mistura de *datasets* com várias caraterísticas para a segmentação de imagens com localizações diferentes daquelas para as quais foi treinado.	*Drive*
Inria+	Treino utilizando conjunto nacional criado manualmente e treinado com pesos da experiência *Inria* inicializados.	- Treino com arquitetura *U-net*; - 200 imagens criadas manualmente com recurso ao *ArcGIS* e *OpenStreetMap*; - Imagens de zonas de Lisboa; - Treino com pesos da experiência Inria inicializados.	Avaliar o desempenho do treino com pesos inicializados da experiência Inria e o ajuste do modelo na segmentação de imagens com caraterísticas semelhantes às de treino.	*Drive*
Full+	Treino utilizando conjunto nacional criado manualmente e treinado com pesos da experiência *Full* inicializados.	- Treino com arquitetura *U-net*; - 200 imagens criadas manualmente com recurso ao *ArcGIS* e *OpenStreetMap*; - Imagens de zonas de Lisboa; - Treino com pesos da experiência Full inicializados.	Avaliar o desempenho do treino com pesos inicializados da experiência *Full* e o ajuste do modelo na segmentação de imagens com caraterísticas semelhantes às de treino.	

Por outro lado, também se tentou melhorar os modelos criados, com a conceção de um conjunto de treino nacional, que foi utilizado para retreinar os modelos com os pesos inicializados das experiências *Full* e *Inria* - as experiências *Full+* e *Inria+*. Assim, esperam-se obter melhores resultados em território nacional devido às novas caraterísticas de edifício apresentadas ao modelo.

3.2.2 Tratamento dos Dados de Treino

Para se poder recorrer à mistura de imagens de diferentes fontes como técnica de treino (experiência *Full*), foi necessário primeiro realizar uma normalização de forma a que todas elas ficassem com as mesmas dimensões e nomes diferentes. Para isso, foi criado um programa em *Python*, o *recorte.py*.

As imagens e respetivas máscaras de treino *big data* foram recortadas através deste programa *Python*, de forma a que cada uma ficasse partida em mosaicos de dimensões normalizadas (256 x 256 x 3) e (256 x 256 x 1). Esta ferramenta necessita da indicação manual de cinco variáveis: as dimensões de recorte (um número inteiro que corresponde simultaneamente a altura e largura de recorte), o *step* ou sobreposição vertical e horizontal entre imagens do recorte (por exemplo 256 para não haver e 128 para haver uma sobreposição de 50% entre imagens), o caminho da pasta raiz que contém a pasta *images* e a pasta *masks*, o caminho para a pasta de saída das imagens recortadas e o caminho para a pasta de saída das máscaras recortadas. Após estas indicações, o programa realiza o recorte de todas as imagens existentes na pasta *images* e *masks* consoante o valor indicado (neste caso "256" de forma a serem obtidas imagens (256 x 256) correspondentes a mosaicos das imagens originais), guarda as imagens recortadas noutras duas pastas de saída e, no final, renomeia-as (imagens e máscaras) utilizando a junção do número respetivo aos seus índices com o nome do *dataset* em causa como denominação, de forma a ser inequívoco para que se possam combinar vários *datasets* na mesma pasta.

Após realizado o recorte das imagens em vários mosaicos, pode acontecer que alguns não contenham informação de rótulo relevante, pelo que foi também desenvolvido um programa *Python* que permite a remoção de máscaras sem rótulos de edifícios (imagens binárias totalmente pretas) e respetivas imagens *RGB* - o *cleaner.py*. Esta ferramenta de tratamento de dados funciona, numa primeira fase, através da iteração pela pasta das máscaras, de forma a realizar a computação da soma de todos os elementos de cada imagem binária da pasta através de uma função criada com este propósito. Com esta operação eliminam-se todas as imagens que não contivessem informação, ou seja, aquelas cuja a soma dos elementos das suas matrizes resulta-se em zero. A pasta de máscaras "limpa" resultante era de seguida comparada com a pasta original das imagens, através de outra função *Python* do programa *cleaner.py*, e são eliminadas desta última as imagens que não apresentem associação, através do seu nome, com qualquer outra máscara binária contida na pasta "limpa".

No final da preparação dos dados obteve-se um total de 44928 imagens e respetivas máscaras para o conjunto de dados *Inria* e 106848 imagens e máscaras para o *Full* - descritos à frente com maior detalhe.

3.2.3 Criação do Conjunto de Treino Nacional

Tendo em conta que os telhados de um edifício podem ser caraterísticos de uma certa região, os modelos foram ainda retreinados com um conjunto de duzentas imagens de zonas dos arredores de Lisboa em Portugal, com objetivo de serem esperados e analisados melhores resultados em cada um dos dois modelos até aqui mencionados (*Full* e *Inria*), especialmente em imagens de Portugal com caraterísticas um pouco diferentes das utilizadas para treinar o modelo, normalmente devido à localização geográfica a que se referem.

Estas duzentas imagens de treino (256 x 256 x 3) foram obtidas com recurso a duas técnicas: digitalização manual das classes sobre uma imagem em ambiente *ArcMap* e um método semiautomático que combina o uso do *QGIS*, *ArcMap* e do *OpenStreetMaps* para obter cento e cinquenta das duzentas imagens que constituem este pequeno conjunto de otimização.

O primeiro método consistiu na criação de um projeto em *ArcMap* e na extração de imagens satélite de alta resolução e não georreferenciadas da plataforma *Google Earth*. Assim foi criada uma *layer* de polígonos de cor branca e foram digitalizadas quatro imagens satélite (1024 x 1024) manualmente. Feito isto, foi criada uma *layer* de cor preta onde se sobrepuseram os polígonos digitalizados com cor branca e exportou-se cada uma das três máscaras resultantes, com as mesmas dimensões que a imagem original.

As restantes amostras deste conjunto de dados foram obtidas com recurso ao *OpenStreetMap* e à possibilidade de serem exportados os polígonos correspondentes a edifícios (Figura 3.2), disponíveis nesta plataforma, em formato *osm*.

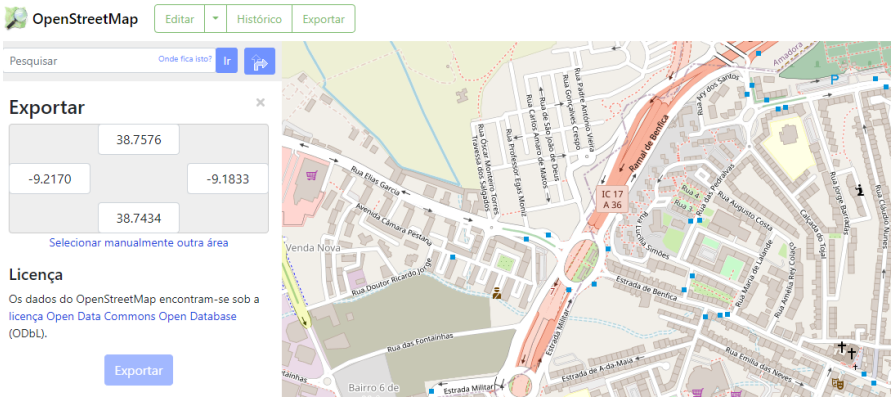


Figura 3.2 - Exportação *OpenStreetMap*

Este formato *osm* pode ser aberto em *QGIS* e exportado como *shapefile*, sendo neste caso exportados apenas os polígonos correspondentes à entidade geoespacial "edifício". Desta forma, foi possível, em seguida, abrir esta *shapefile* no *ArcMap* de forma a se sobreporem estes com, por exemplo, os mapas base de satélite da *ArcGis*. Com o *ArcMap* e a ferramenta *editor*, tornou-se possível o ajuste dos polígonos às imagens, que por vezes sofrem de distorção, e a exportação de imagens e respetivas máscaras - criadas através da sobreposição de uma *layer* de polígonos, relativa aos edifícios e de cor branca, com uma *layer* de fundo de cor preta. As imagens e máscaras resultantes destas técnicas foram também submetidas ao mesmo processo de tratamento que as outras, redimensionando em (256 x 256 x 3) e eliminando as máscaras sem informação.

Assim, assume-se transferência de conhecimento, adquirido pelos modelos, a partir do qual, devido à grande quantidade de dados disponíveis para treino, se obtém uma generalização de pesos e características (modelos *Full e Inria*). Estes modelos, após retreinados com este conjunto nacional, podem ver o seu desempenho melhorado (modelos *Full+* e *Inria+*) com este pequeno conjunto de novos dados de treino, mais precisos, devido à maior semelhança de caraterísticas e texturas de edifício, em relação à generalidade dos locais onde se deseja realizar a segmentação de edifícios.

3.3 Ambiente de Desenvolvimento e *Software*

Nesta tese a maioria do trabalho prático foi realizado utilizando linguagem de programação *Python*. Esta permitiu o desenvolvimento de múltiplas ferramentas automáticas de processamento e tratamento de dados como será explorado em detalhe nesta seção. É importante referir a utilização de módulos de programação *Tensorflow* que permitem uma implementação eficiente em *Python* da rede neural *U-net* descrita em detalhe na próxima seção, nomeadamente com recurso aos módulos da biblioteca *Keras*.

Como pode ser observado na Figura 3.3, o primeiro e mais antigo módulo adequado para o desenvolvimento e treino de Redes Neurais Profundas é o *Torch*, lançado em 2002 [88]. O *Torch* consistia originalmente numa implementação e interface *C++*. Hoje em dia, é implementado em *C/CUDA*, expondo uma interface na linguagem de programação *Lua*. Mais recentemente, em 2015, a empresa globalmente conhecida como *Google* disponibilizou ao público um *software open-source* de *DL* - o *Tensorflow* [89] - com o principal foco de fornecer um ambiente baseado em *Python* simples e flexível para a criação de modelos de aprendizagem máquina profunda para computadores, dispositivos móveis, *web* e nuvem. Entre as datas de lançamento do *Torch* e do *Tensorflow*, outros como o *Caffe* [90] e o *DL4J* [91] já tinham começado a pavimentar o caminho para a criação mais facilitada de modelos profundos, tendo sido os primeiros a permitir o treino de *CNN* e a disponibilizar interfaces em *Python, MATLAB* e *Java*. Além destes, é importante referir o *kit* de desenvolvimento de *software Nvidia Deep Learning* [92] que veio permitir a maximização dos desempenhos de treino em *GPU Nvidia*, nomeadamente com a disponibilização do módulo *cuDNN*.

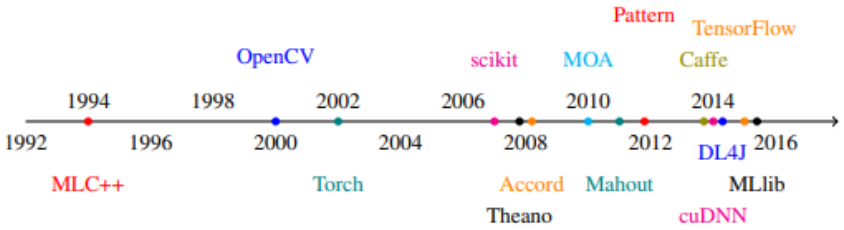


Figura 3.3 - Cronologia de bibliotecas de redes neurais profundas [93]

O *Tensorflow* é baseado na linguagem de programação *Python*. Neste, os algoritmos de Aprendizagem de Automática são representados como grafos computacionais. Um grafo computacional ou de fluxo de dados é uma forma de gráfico direcionado onde vértices, ou nós, descrevem as operações e arestas representam os dados que formulam essas operações [94]. Basicamente e como se pode observar na Figura 3.4, são desenhadas arestas direcionadas de *x* e *y* para um nó de saída representando *z*, sendo o nó anotado com um rótulo que descreve a computação realizada - semelhante à estrutura básica de uma rede neural. Por outro lado, *Keras* é a *API* de alto nível do *TensorFlow*, programada também em *Python*, com o objetivo de fornecer experimentação rápida e consiste na principal ferramenta utilizada no desenvolvimento prático desta tese. Fornece abstrações essenciais e blocos de construção para desenvolver e enviar soluções de *ML* com altas velocidades de iteração, sendo marcada pela sua simplicidade, flexibilidade e poder de computação [95].

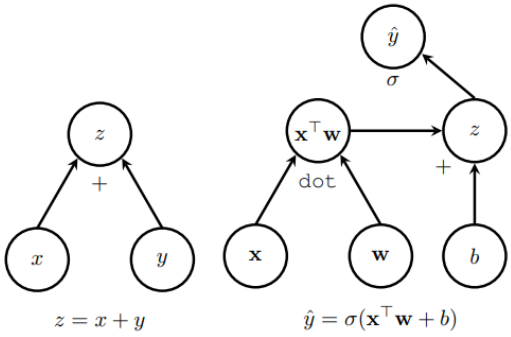


Figura 3.4 - Grafo de fluxo de dados [93]

Assim, com estes avanços que sustentam uma melhor e mais fácil criação de modelos de *ML*, torna-se interessante explorar as várias ferramentas que, em conjunto com os avanços na partilha de informação

referidos na Seção 1.1, permitem a criação de modelos com diversas aplicações, nomeadamente no campo da ciência geoespacial.

3.4 Implementação da Arquitetura *U-net*

Os modelos foram treinados utilizando a arquitetura *U-net* (Seção 2.8.4), pelo que foram implementadas todas as camadas que a constituem, com recurso ao *Tensorflow* e *Keras*, utilizando a programação *Python*. Estes módulos, além de utilizados na implementação da arquitetura, foram também necessários para criar programas relativos ao treino, validação e teste deste algoritmo de *DL*.

É importante notar que a *U-net* foi implementada como sendo uma função (*modelo_U-net* da Figura 3.5), pelo que pode ser reutilizada e iniciada em qualquer outro programa como um módulo, tendo em conta que foi guardada como um ficheiro *.py* individual (*unet.py*). Por isso, nas fases de treino, validação e teste foi apenas necessário utilizar o comando "*from unet import modelo_unet*" para se aplicar esta arquitetura. Nesta fase são apresentados os vários módulos *Keras* e variáveis necessários para recriar a *U-net* e que se encontram expostos na Figura 3.5.

Antes de se proceder é necessário esclarecer que no final de cada linha relativa a uma operação (convolução, redução da amostra, *dropout*, concatenação e camada de saída) existe a indicação de qual a camada a que esta se aplica. Por exemplo, na Figura 3.5 a primeira convolução *c1* (linha 9) é aplicada à variável de entrada *s* (linha 6), pelo que no final da sua linha existe "*(s)",* e o *dropout* (linha 10) que se segue é aplicado à camada de saída de *c1* (linha 9) pelo que no final da linha é necessário a indicação *"(c1)"*.

```
1  from keras.models import Model
2  from keras.layers import Input, Conv2D, MaxPooling2D, concatenate, Conv2DTranspose , Dropout
3
4  def modelo_unet(IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS):
5      inputs = Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
6      s = inputs
7
8      #Descida
9      c1 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(s)
10     c1 = Dropout(0.1)(c1)
11     c1 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c1)
12     p1 = MaxPooling2D((2, 2))(c1)
13
14     c2 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p1)
15     c2 = Dropout(0.1)(c2)
16     c2 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c2)
17     p2 = MaxPooling2D((2, 2))(c2)
18
19     c3 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p2)
20     c3 = Dropout(0.2)(c3)
21     c3 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c3)
22     p3 = MaxPooling2D((2, 2))(c3)
23
24     c4 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p3)
25     c4 = Dropout(0.2)(c4)
26     c4 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c4)
27     p4 = MaxPooling2D((2, 2))(c4)
28     #Fundo
29     c5 = Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p4)
30     c5 = Dropout(0.3)(c5)
31     c5 = Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c5)
32     #Subida
33     u6 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(c5)
34     u6 = concatenate([u6, c4])
35     c6 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u6)
36     c6 = Dropout(0.2)(c6)
37     c6 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c6)
38
39     u7 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(c6)
40     u7 = concatenate([u7, c3])
41     c7 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u7)
42     c7 = Dropout(0.2)(c7)
43     c7 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c7)
44
45     u8 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same')(c7)
46     u8 = concatenate([u8, c2])
47     c8 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u8)
48     c8 = Dropout(0.1)(c8)
49     c8 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c8)
50
51     u9 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same')(c8)
52     u9 = concatenate([u9, c1], axis=3)
53     c9 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u9)
54     c9 = Dropout(0.1)(c9)
55     c9 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c9)
56
57     outputs = Conv2D(1, (1, 1), activation='sigmoid')(c9)
58     model = Model(inputs=[inputs], outputs=[outputs])
59     model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
60
61     return model
```

Figura 3.5 - Implementação *U-net* em *Python*

3.4.1 Módulo *Input*

O módulo *Input* (Figura 3.5, linha 5) contém as variáveis locais da função *modelo_unet*. Estas são as dimensões da imagem de entrada (*IMG_HEIGHT e IMG_WIDHT*) e o número de bandas da mesma (*IMG_CHANNELS*). Desta forma, o modelo fica informado acerca das dimensões de entrada que deve aceitar, neste caso (256 x 256 x 3) – imagens *RGB* com 256 pixéis de altura e largura. Estes valores foram mantidos constantes ao longo das três principais fases da experiência (treino, teste e validação).

3.4.2 Módulo *Conv2D*

O *Conv2D* corresponde à implementação das camadas de convolução. Através de variáveis locais foi também definido o número de filtros, o tamanho do *kernel*, a função de ativação, o inicializador de *kernel* e o preenchimento da operação. Estas camadas são sempre seguidas por um módulo *Dropout* que contém uma variável local relativa à probabilidade de excluir um neurónio.

3.4.3 Módulo *Conv2DTranspose*

A convolução transposta na subida da arquitetura é realizada pelo módulo *Conv2DTranspose*, tendo sido apenas necessário explicitar, neste caso, as variáveis locais relativas ao número de filtros, ao tamanho do *kernel* e ao passo e preenchimento da operação de convolução.

3.4.4 Módulo *MaxPooling2D*

O *MaxPooling2D* é o módulo relativo à camada de redução da amostra máxima. Nesta arquitetura esta é sempre realizada com uma submatriz de tamanho (2 x 2).

3.4.5 Módulo *concatenate*

Como se pode observar, as camadas de subida são constantemente concatenadas com as paralelas camadas de descida utilizando o módulo *concatenate*. Este módulo necessita apenas da indicação das duas camadas a concatenar, por exemplo, a concatenação *u6* da primeira camada de convolução transposta com a última camada de convolução da descida (Figura 3.5, linha 34).

3.4.6 Variável *outputs*

A variável *outputs*, definida na linha 57 da Figura 3.5, tem como principal objetivo classificar os resultados de saída da última camada da arquitetura, utilizando uma função de ativação *Sigmoid*.

3.4.7 Módulo *compile*

O *compile* é um módulo que configura o modelo para o treino. Basicamente as variáveis locais deste correspondem à indicação do otimizador, da função de perda e da métrica a ser avaliada ao longo do treino do modelo.

3.5 Fase de Treino

O treino dos modelos de *ML* foi realizado numa plataforma *online* desenvolvida pela mesma empresa fundadora do *Tensorflow*, a *Google*. O *Colab* é uma poderosa ferramenta de processamento *online* que, quando combinada com uma boa quantidade de armazenamento em nuvem providenciado pela ferramenta *Google Drive*, permite atingir resultados impressionantes, comparáveis com os obtidos por supercomputadores de custo elevado. Viabiliza, desta forma, a escrita e execução de código *Python* em qualquer dispositivo com capacidade de aceder a um *browser* de *internet*. Além disso, é uma plataforma de acesso gratuito, vem com a maioria dos módulos de *ML Python* necessários pré-instalados e permite uma fácil partilha de trabalhos, por exemplo, caso seja um trabalho em equipa ou se pretenda publicar resultados e ferramentas *online* com maior facilidade - da mesma forma que os documentos criados e editados no *Google Docs* são partilhados. É importante também referir que este é implementado com recurso à interface *web Jupyter Notebooks*.

Logo de início, um dos principais obstáculos com o qual este projeto se deparou foi a falta de processamento gráfico existente. Além de ser de elevado custo, a maioria das *GPU* utilizadas para este propósito não são disponibilizadas ao consumidor comum. As suas aplicações são, normalmente, estritamente relacionadas com *ML* pelo que para se realizar um treino local eficiente seria necessário um grande investimento numa máquina que, além do uso restrito, poderia ser facilmente ultrapassada num futuro próximo, devido essencialmente à evolução exponencial do poder de processamento gráfico computacional. Assim, o processamento em nuvem surgiu como uma resolução para este problema, sendo utilizado na fase de treino do algoritmo de *ML*. Os dados de treino resultantes da Seção 3.2.2 e da Seção 3.2.3 foram então exportados para o *Google Drive* e treinados por partes, de forma a não ser atingido o limite de *RAM* da versão gratuita do *Colab*.

Os conjuntos de dados divididos por partes tiveram de ser exportados para o *Google Drive* que pode ser posteriormente acedido pelo *Colab*. Para isso, foi necessário criar uma pasta *zip* para cada uma das pastas correspondentes ao treino por partes. Assim foram exportadas dezoito pastas *zip* relativas ao treino *Inria* e quarenta e duas pastas *zip* relativas ao treino *Full*.

Organizaram-se as imagens por partes de no máximo 2600 imagens e máscaras (256 x 256) de forma a que fosse possível treinar um modelo sem problemas de memória *RAM* ou *VRAM* utilizando o *Google Colab*. Dividindo as imagens do *Inria* em dezoito partes obtiveram-se grupos de treino de 2496 imagens e respetivas máscaras, que foram utilizados sucessivamente para treinar o modelo. O *Full* foi dividido em quarenta e duas partes e cada parte continha 2544 imagens de treino e respetivas máscaras. A organização e divisão em pastas procedeu-se automaticamente também com recurso ao *Python*.

3.5.1 Implementação do Treino

Uma das principais caraterísticas que fazem o treino *online* diferir do local é o facto dos dados que são utilizados terem de residir numa plataforma de armazenamento *online* como é o caso da *Google Drive*.

Para isso, foi necessário utilizarem-se os módulos *Python* "*drive*" e "*zipfile*" do *Google.Colab.* O primeiro assegura a ligação ao armazenamento do *Google Drive*, enquanto que o segundo garante a extração da pasta *zip* que se deseja usar para treino.

De seguida, tornou-se necessário proceder à listagem de todas as imagens e máscaras para a memória do sistema em nuvem. Para isso, foi elaborado um programa que agrupa todas as imagens e respetivas máscaras, que se encontram armazenadas em nuvem no *Google Drive*, em duas listas que são posteriormente utilizadas para treinar o modelo. De notar, a necessidade de as imagens serem convertidas para *RGB* devido ao facto do módulo *cv2* as ler em formato *BGR*. É também necessário realizar uma normalização das máscaras e imagens antes da passagem para a aprendizagem.

Finalmente, o último passo consiste na realização do treino do modelo. Para isso, é necessário indicar na função do modelo *U-net* as dimensões de entrada como explicado na Seção 3.4.1. Inicialmente cria-se um corte dos dados de treino de forma a que seja possível utilizar parte destes para validação, com uma percentagem definida em 20% da totalidade de dados de treino. Foram também definidas técnicas de monitorização do treino, os *callbacks*, nomeadamente o *earlystopping*, que termina o treino quando a perda em validação não melhora em dez *epochs* (uma paciência ou *patience* de "10"), o *checkpoint* que guarda automaticamente modelos que apresentam a precisão de validação máxima, quando comparados com os dos *epochs* anteriores, e o *CSVlogger* que guarda num ficheiro *csv* todas as informações estatísticas registadas ao longo do processo de treino, permitindo a escolha do melhor modelo guardado pelo *checkpoint*. Com todas estas variáveis definidas pode ser realizado o encaixe do modelo, onde se indicam os dados de treino, a dimensão dos lotes de treino, os *callbacks* ou técnicas de monitorização do treino utilizadas, o número máximo de *epochs* e os dados de validação a ser monitorizados (neste caso os que foram excluídos em 20% do conjunto de dados de treino), para se dar início ao treino do modelo.

De notar que este treino foi apenas realizado para a primeira parte de cada *dataset*, sendo que as seguintes partes constituintes do mesmo foram treinadas da mesma forma só que com a inicialização dos pesos de treino resultantes da parte anterior, ou seja, o modelo final é resultante de um treino sucessivo e por partes. O modelo escolhido para retreino entre cada parte foi sempre o que apresentasse a melhor combinação de valores de perda em validação e precisão em validação, com recurso ao uso da técnica *early stopping*, *checkpoint* e à própria análise dos valores produzidos em cada *epoch* com o *CSVlogger*.

3.6 Fase de Validação

Em termos de validação, ambos os conjuntos foram validados para os mesmos tipos de imagens de forma a que os resultados fossem comparáveis estatisticamente. Como se pode observar na Tabela 3.2, um dos *dataset* de validação consistiu na seleção de cinquenta de algumas das imagens mais percetíveis (em termos de digitalização e até mesmo qualidade de imagem) do conjunto de dados de treino utilizado na tese realizada por João Sacadura no ano 2021 [49], de forma a se aferir o impacto na previsão realizada em dados que, por natureza da sua localização geográfica, podem apresentar algumas caraterísticas diferentes do treino - o conjunto *nac*. Além deste, foi ainda criado um conjunto de validação composto por duzentas imagens, com caraterísticas distintas entre si, deixadas de fora do treino do modelo *Full*, para avaliar os desempenhos dos modelos - o conjunto *int.* Assim sendo, um dos objetivos principais passa por tentar perceber quais as diferenças em termos de qualidade de previsão, para imagens com caraterísticas um pouco diferentes ou variáveis das aprendidas pelos modelos - a capacidade de generalização dos modelos.

Tabela 3.2 - Síntese dos conjuntos de validação utilizados

Nome	Descrição	Objetivo
int	Conjunto de dados de validação composto por 200 imagens e respetivas máscaras, adquiridas e colocadas de parte do treino do conjunto compilado de dados *Full*.	Avaliar o desempenho dos modelos num conjunto de dados com caraterísticas distintas e aferir as diferenças observadas quando se aplicam os modelos retreinados com o conjunto de treino nacional.
nac	Conjunto de dados de validação constituído por 50 imagens e respetivas máscaras selecionadas do conjunto de dados de treino de Sacadura [49].	Avaliar as melhorias de desempenho das previsões antes e após o treino dos modelos com o conjunto de dados de treino nacional.

Na Figura 3.6 apresentam-se alguns exemplos de amostras (imagens *RGB* e respetivas máscaras verdadeiras ou *groud truth*) dos dois conjuntos de validação utilizados.

Figura 3.6 - Exemplos de imagens usadas na validação dos modelos

De forma a serem avaliados os resultados previstos por cada modelo foi necessário selecionar métricas que possibilitassem a correta comparação de qualidade. Foram assim selecionadas três métricas que possibilitam a comparação dos resultados previstos pelos modelos com as respetivas máscaras *ground truth* de forma a que seja possível pontuar e comparar cada um:

1) Precisão (*Precision*)

A precisão é intuitivamente a capacidade de o classificador não rotular como positiva uma amostra que é negativa. O melhor valor é "1" e o pior valor é "0". Esta métrica pode ser definida matematicamente como:

$$Precisão = \frac{TP}{TP+FP} \qquad [3.1]$$

Onde *TP* é o número de verdadeiros positivos, ou seja, edifícios corretamente identificados e *FP* o número de falsos positivos, ou seja, edifícios que foram identificados pelo modelo e não existem.

2) Revocação *(Recall)*

A revocação é intuitivamente a habilidade do classificador em encontrar amostras positivas. O melhor valor é 1 e o pior valor é 0. Pode ser definido como:

$$Revocação = \frac{TP}{TP+FN} \qquad [3.2]$$

Onde *TP* é o número de verdadeiros positivos e *FN* o número de falsos negativos, ou seja, edifícios que não foram identificados.

3) Pontuação F1

A pontuação F1 pode ser interpretada como uma média harmónica da precisão e revocação, onde o seu melhor valor é "1" e o pior "0". A contribuição relativa da precisão e revocação são iguais. A fórmula para a pontuação F1 pode ser definida matematicamente:

$$F1 = 2 * \frac{(precisão * revocação)}{(precisão + revocação)} \qquad [3.3]$$

3.6.1 Implementação da Validação

De forma a ser possível calcular estas métricas para múltiplas imagens utilizando um modelo foi criado um *script Python* que compara as máscaras verdadeiras com as máscaras estimadas, para um certo intervalo de valores de corte, calculando a média das métricas para cada intervalo (denominado *valid.py*). Este tem também a capacidade de gerar um ficheiro em formato *.csv* muito útil para a criação dos gráficos que se seguem. Além do exposto, para cada modelo e conjunto de validação escolhido este programa gera quatro imagens binárias (256 x 256 x 1) correspondentes às estimativas com valores de corte dos quatro primeiros múltiplos de 0,2 (0,2; 0,4; 0,6 e 0,8). Além deste foi também criado um outro programa (*valid256.py)* que consiste numa aplicação simples de um modelo a uma imagem com as dimensões (256 x 256 x 3), ou seja, uma imagem com as mesmas dimensões daquelas utilizadas para treinar o modelo. Resulta um ficheiro de saída (256 x 256 x 1) com a previsão realizada e a informação acerca das métricas de validação referidas.

Assim, o processo de validação torna-se muito mais eficiente e com capacidade de validar modelos em conjuntos de dados de grande dimensão, ao longo de intervalos de valor de corte.

3.7 Fase de Teste

O teste dos modelos de *DL*, refere-se à análise das possíveis aplicações que podem advir destes. A validação constituiu uma fase mais comparativa e de análise estatística que indicou qual seria o melhor modelo a aplicar, através das métricas de análise de resultados referidas na seção anterior. Por outro lado, nesta fase, testam-se os modelos em dados novos que não têm nenhuma imagem de referência segmentada e que apenas confirmam os bons resultados registados na validação. São dados brutos que podem ser transformados em produtos segmentados, neste caso edifícios, que podem ter várias aplicações.

Tal como a preparação, validação e treino, a fase de teste dos modelos foi realizada utilizando programação *Python*. Para esta tarefa foram desenvolvidos dois programas: o *teste256.py* e o *reconst.py*. Estes dois programas necessitavam sempre do carregamento do modelo resultante do treino, para realizar as previsões, que podia ser descarregado do *Google Drive*, depois de processado pelo *Colab*, e ser utilizado localmente devido ao facto de previsões em imagens (256 x 256 x 3) não pesarem tanto na memória gráfica do sistema como pesa a operação de treino.

O programa *teste 256.py* consiste numa aplicação simples do modelo a uma imagem com as dimensões (256 x 256 x 3), ou seja, uma imagem com as mesmas dimensões daquelas utilizadas para treinar o modelo. Esta aplicação pode ser bastante útil para uma análise visual da qualidade das previsões de cada modelo. Resulta um ficheiro de saída (256 x 256 x 1).

As imagens provenientes de sistemas equipados em sistemas aéreos, VANT e satélites podem ser de variadas resoluções pelo que se tornou necessário criar uma solução que permitisse a previsão neste tipo de dados. Utilizando a lógica do programa *recorte.py*, criou-se outro que recorta uma imagem original em partes com dimensões equivalentes às dos dados de treino, neste caso (256 x 256 x 3), realiza uma previsão por cada parte utilizando o modelo desejado e no final reconstrói a imagem segmentada com dimensões correspondentes à original de entrada, através da junção dos vários mosaicos de previsão - o *reconst.py*. Esta operação torna-se necessária porque a aplicação de um modelo *U-net* treinado irá sempre obter previsões mais precisas em imagens de teste com as mesmas dimensões que as utilizadas na fase de treino.

É importante referir que as imagens foram recortadas de forma diferente das de treino de forma a evitar distorções de costura causadas quando os objetos a segmentar aparecem recortados. Para isso, este programa vem preparado com um conjunto de iterações que recortam a imagem *RGB* original, com um passo de, por exemplo, 128 píxeis de modo a que haja uma sobreposição lateral e longitudinal de 50% entre imagens, realizam as previsões em cada imagem de recorte (256 x 256 x 3), produzem as respetivas previsões (256 x 256 x 1) para cada e reconstroem uma imagem final resultante da concatenação, nas várias sobreposições horizontais e verticais, dos elementos binários matriz de cada imagem prevista, garantindo uma segmentação mais completa de entidades.

3.8 Síntese do Projeto

Nesta seção será feita uma síntese geral do projeto bem como os principais *softwares* utilizados e principais ferramentas próprias desenvolvidas com recurso à programação em *Python.*

Como se pode constatar, o projeto baseou-se na criação dum conjunto de ferramentas que permitiram o manuseamento e criação de dados, nomeadamente imagens e modelos de *ML*. Tanto o *software* como os programas próprios desenvolvidos em *Python* tiveram grande destaque no desenvolvimento da tese. Assim as Tabelas 3.3 e 3.4 que se seguem têm como principal objetivo expor e descrever de uma forma organizada todas as ferramentas desta natureza que foram expostas no decurso deste capítulo. Na Tabela 3.4 são também apresentados *links* com ligação às páginas publicadas no *GitHub* para cada programa desenvolvido.

Tabela 3.3 - Síntese das ferramentas utilizadas

Software / Ferramenta	**Descrição**	**Utilidade**
Spyder	O *Spyder* é um ambiente científico gratuito e de código aberto escrito em *Python*, para *Python* e projetado por e para cientistas, engenheiros e analistas de dados.	Criação de ferramentas próprias para tratamento, validação e teste de dados.
Google Drive	O *Google Drive* permite o armazenamento, partilha e colaboração em ficheiros e pastas a partir dum dispositivo móvel, tablet ou computador.	Armazenamento em nuvem de todos os conjuntos de dados utilizados no projeto.
Google Colab	O *Colaboratory* é um produto da *Google* que permite a qualquer pessoa a escrita e execução código *Python* por meio dum navegador *web*, sendo especialmente adequado para aprendizagem automática, análise de dados e educação.	Utilização das capacidades gratuitas de processamento em nuvem para treino dos modelos de *DL*.
OpenStreetMap	*OpenStreetMap* é um mapa gratuito e editável de todo o mundo, construído por voluntários e lançado com uma licença de conteúdo aberto.	Aquisição de ficheiros *osm* de entidades geoespaciaias para criação de máscaras binárias próprias.
QGIS	*QGIS* é um Sistema de Informação Geográfica de Código Aberto, sendo um projeto oficial da *Open Source Geospatial* Foundation (*OSGeo*). É compatível com *Linux, Unix, Mac OSX*, *Windows* e *Android.*	Exportação do ficheiro *osm* para o formato *shapefile* para carregamento em *ArcMap.*
ArcMap	O *ArcMap* é um programa usado para exibir e explorar conjuntos de dados SIG para uma certa área de estudo, onde se atribuem símbolos e se criam *layouts* de mapas para impressão ou publicação.	Associação dos polígonos do *OpenStreetMap* com imagens de base de satélite *ArcGIS* para criação de imagens e máscaras de treino. Criação manual de dados de treino através da digitalização de imagens satélite adquiridas.

Tabela 3.4 - Síntese e apresentação dos programas desenvolvidos em *Python*

Nome	**Descrição**	***Link***
unet.py	Programa desenvolvido no *Spyder que* corresponde à implementação *Python* da arquitetura *U-net*, tendo sido frequentemente utilizado por outros como módulo.	*GitHub*
recorte.py	Programa *Python* desenvolvido no *Spyder* com o objetivo de ser realizada partição de uma ou mais imagens e correspondentes máscaras em mosaicos de tamanho pré-definido, muito útil para normalização de imagens de diversos conjuntos para um só treino.	*GitHub*
cleaner.py	Programa Python desenvolvido no Spyder para limpeza de imagens resultantes do programa *recorte.py*. Analisa uma pasta de máscaras e elimina as que não contêm informação, bem como as imagens *RGB* de uma pasta de imagens correspondente.	*GitHub*
treino.ipynb	*Jupyter Notebook* executado no *Google Colab* para realizar o treino da primeira parte de cada conjunto de dados, criando o modelo a ser introduzido no retreino das outras partes.	*GitHub*
retreino.ipynb	*Jupyter Notebook* executado no *Google Colab* com o objetivo de realizar o retreino das várias partes constituintes dum conjunto de dados, sendo que cada parte foi sempre treinada sob os pesos inicializados do melhor modelo gerado pela parte que lhe antecedeu.	*GitHub*
valid256.py	Programa *Python* desenvolvido no *Spyder* com o objetivo de ser realizada a validação numa imagem e máscara de dimensões (256 x 256).	*GitHub*
valid.py	Programa *Python* desenvolvido no *Spyder* que realiza a validação numa pasta com várias imagens e respetivas máscaras.	*GitHub*
teste256.py	Programa *Python* desenvolvido no *Spyder* que prevê numa imagem de dimensões iguais às usadas para treinar o modelo.	*GitHub*
reconst.py	Programa *Python* desenvolvido no *Spyder* com o objetivo de ser realizada uma previsão suave em imagens de diversas dimensões.	*GitHub*

3.8.3 Esquema Síntese do Projeto

Finalmente, com o esquema exposto na Figura 3.7 torna-se bastante percetível o procedimento utilizado e a interligação entre os vários elementos expostos nas Tabelas 3.3 e 3.4.

O projeto inicia-se com a aquisição de dados para treino através de duas formas: utilizando *datasets* publicados *online* e criando dados de treino manualmente utilizando o *ArcMap, QGIS e OpenStreetMap*. Os dados adquiridos na *internet* foram submetidos a dois processos automáticos de tratamento. De seguida, os dados foram armazenados na plataforma *Google Drive* e foram treinados recorrendo ao *Google Colab* e á arquitetura *U-net* materializada no *unet.py*. Os modelos resultantes deste treino foram validados e o melhor foi usado para a fase final de teste que se encontra descrita em detalhe no próximo capítulo, juntamente com a análise dos resultados de validação que permitiram decidir qual o melhor modelo.

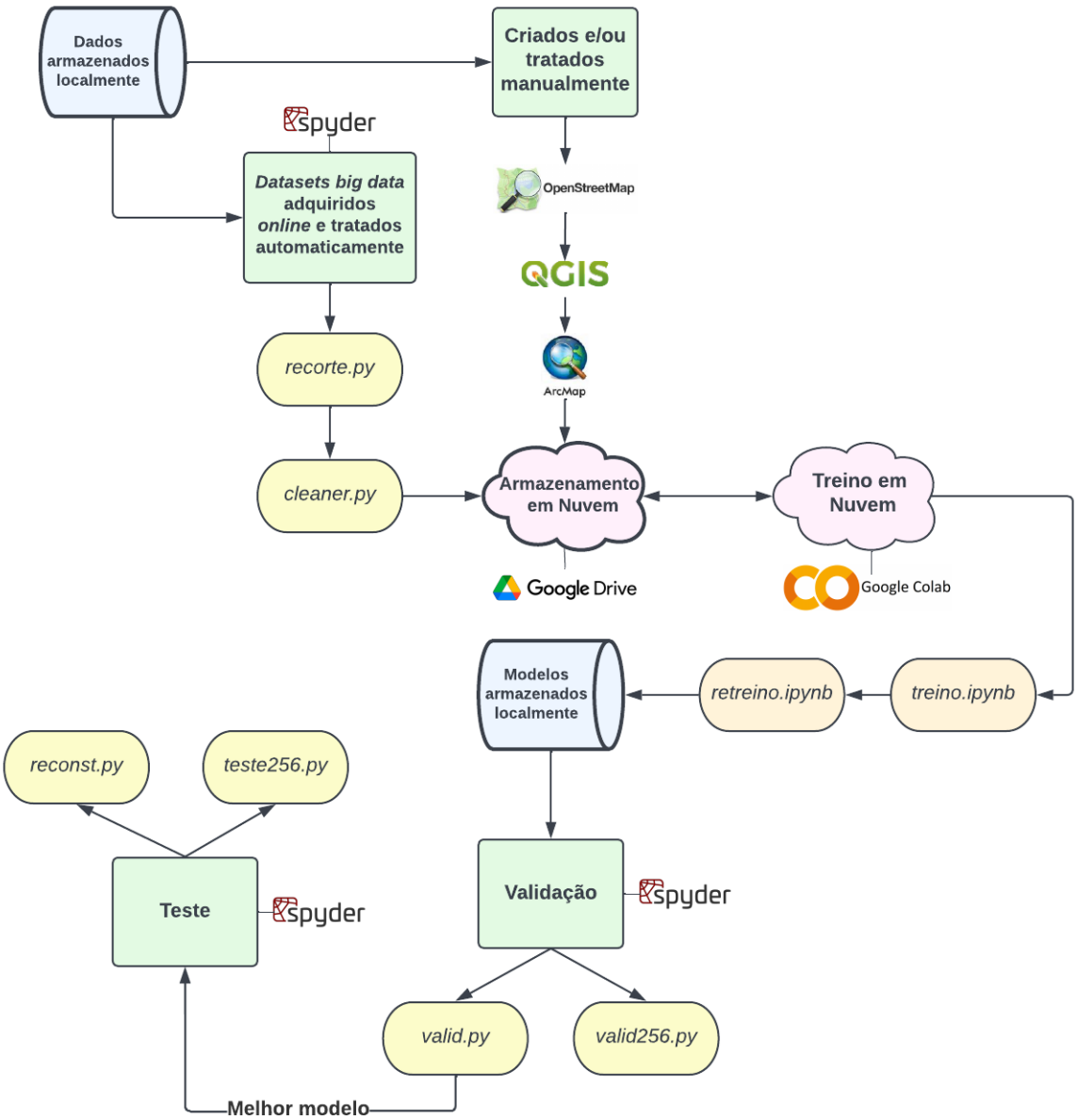


Figura 3.7 - Esquema síntese do projeto

4 Resultados e Discussão

4.1 Fase de Validação

Nesta seção são analisados os resultados obtidos por cada modelo, *Inria* e *Full* (Tabela 3.1) quando validados com os dados *int* e *nac* descritos na Tabela 3.2. As métricas usadas na validação foram descritas na Seção 3.6: pontuação F1, revocação e precisão. Assim, torna-se percetível quais são os que beneficiaram mais ou menos das várias técnicas de treino utilizadas. A validação é também importante na determinação de qual o modelo que consegue generalizar melhor o seu treino a dados nunca antes apresentados.

É importante referir que os gráficos que são apresentados foram resultado dos valores produzidos e guardados em formato *csv* pelo programa *valid.py* como referido na Seção 3.6.1. Além disso são apresentados exemplos de imagens binárias validadas e convertidas em formato *shapefile*, explorados em maior detalhe na seção relativa ao teste dos modelos. Assim criaram-se figuras que permitem uma melhor visualização e comparação de resultados através da sobreposição de polígonos de previsão validados com respetivas imagens *RGB*.

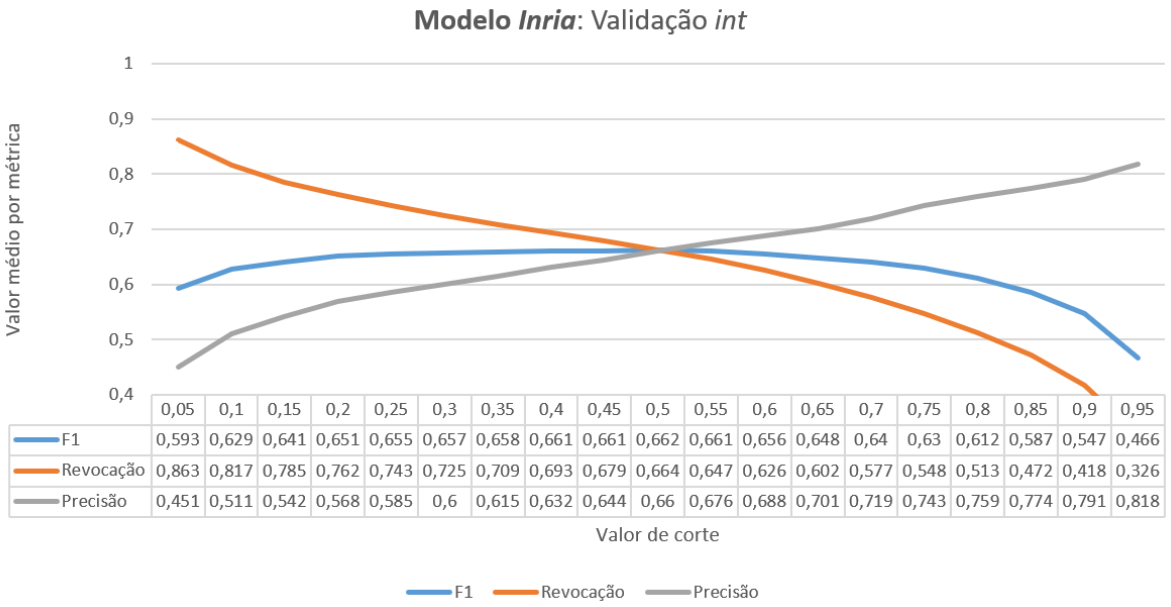
4.1.1 Validação *int*

A validação i*nt* é composta por duzentas imagens de alta resolução com caraterísticas bastantes diferentes em termos de escala e até texturas de telhado devido ao facto de ter sido criada através da extração de imagens do conjunto de treino *Full*, que mistura três *datasets* disponíveis *online* (Tabela 3.1). Por isso, o objetivo principal desta seção é avaliar e comparar o desempenho dos modelos base *Full* e *Inria*.

Os valores esperados não são muito elevados devido ao facto de as previsões neste conjunto de validação serem bastantes difíceis de realizar e também pelo facto de ser bastante difícil uma máscara corresponder exatamente ao resultado previsto. As máscaras de algumas imagens são bastante grosseiras na sua classificação de edifício e por vezes apresentam erros que podem influenciar os resultados obtidos nas métricas. Isto acontece principalmente quando as imagens foram criadas utilizando polígonos que não correspondem totalmente ao edifício representado na imagem, normalmente quando são utilizados polígonos do *OpenStreetMap* para criar as máscaras e não existe retificação da qualidade. De qualquer das maneiras, este *dataset* permite uma comparação e avaliação da capacidade dos modelos realizarem previsões em imagens de várias localizações, com vários tipos e texturas de telhados e de diferentes escalas e valores radiométricos. Permite também um ponto de comparação com os modelos avaliados com o conjunto *nac*.

O modelo *Inria* foi o que recorreu a menos dados de treino. Como se pode observar pelos resultados da Figura 4.1, o valor de cruzamento da revocação com a precisão origina um valor médio de pontuação F1 de 0,662, com um valor de corte da imagem prevista de “0,5”, que pode ser visto como o melhor desempenho médio do modelo, porque é onde existe uma maior aproximação entre a precisão e revocação, ou seja, um valor ótimo. Neste caso, um resultado razoável que demonstra a fraca capacidade do *Inria* ser generalizado para outro tipo de imagens com diferentes caraterísticas. A revocação demonstra a dificuldade de as previsões serem completas e englobarem todos os edifícios esperados pela máscara de validação, e a precisão indica que o modelo classifica erradamente bastantes pixéis na

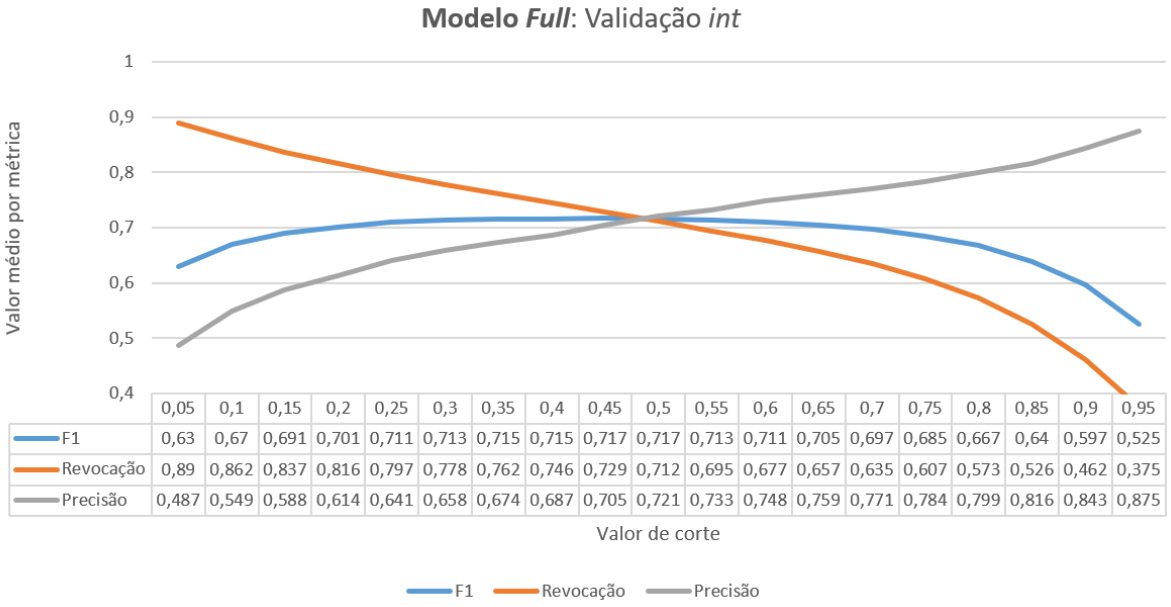
imagem como edifício. Estes resultados podem derivar do facto do modelo ser validado em algumas imagens de escalas muito elevadas comparativamente às imagens usadas para o treinar.



	0,05	0,1	0,15	0,2	0,25	0,3	0,35	0,4	0,45	0,5	0,55	0,6	0,65	0,7	0,75	0,8	0,85	0,9	0,95
F1	0,593	0,629	0,641	0,651	0,655	0,657	0,658	0,661	0,661	0,662	0,661	0,656	0,648	0,64	0,63	0,612	0,587	0,547	0,466
Revocação	0,863	0,817	0,785	0,762	0,743	0,725	0,709	0,693	0,679	0,664	0,647	0,626	0,602	0,577	0,548	0,513	0,472	0,418	0,326
Precisão	0,451	0,511	0,542	0,568	0,585	0,6	0,615	0,632	0,644	0,66	0,676	0,688	0,701	0,719	0,743	0,759	0,774	0,791	0,818

Figura 4.1 - Resultados *Inria* na validação *int*

Por outro lado, o modelo *Full*, que foi treinado com um conjunto de dados de maior dimensão, apresenta resultados bastante mais promissores. Como se pode constatar da Figura 4.2, no cruzamento da revocação com a precisão, registam-se valores F1 de 0,717. Este resultado já pode ser considerado bom e pode ser dado destaque a capacidade deste para segmentar imagens com maior resolução e escala quando comparado com o *Inria.*



	0,05	0,1	0,15	0,2	0,25	0,3	0,35	0,4	0,45	0,5	0,55	0,6	0,65	0,7	0,75	0,8	0,85	0,9	0,95
F1	0,63	0,67	0,691	0,701	0,711	0,713	0,715	0,715	0,717	0,717	0,713	0,711	0,705	0,697	0,685	0,667	0,64	0,597	0,525
Revocação	0,89	0,862	0,837	0,816	0,797	0,778	0,762	0,746	0,729	0,712	0,695	0,677	0,657	0,635	0,607	0,573	0,526	0,462	0,375
Precisão	0,487	0,549	0,588	0,614	0,641	0,658	0,674	0,687	0,705	0,721	0,733	0,748	0,759	0,771	0,784	0,799	0,816	0,843	0,875

Figura 4.2 - Resultados *Full* na validação *int*

Na Figura 4.3 é apresentada a comparação de três previsões interessantes (256 x 256 x 1), transformadas em polígonos e sobrepostas em *ArcMap*, realizadas neste conjunto de validação que suportam alguns factos expostos como a dificuldade de previsão do modelo *Inria* e *Full* a diferentes resoluções e escalas.

É claro que as métricas resultantes das experiências desta seção acabam por ser influenciadas por este fator podendo ser apontado como uma das razões pelas quais pode ser necessário um ajuste de pesos via retreino, consoante o tipo de dados de entrada nos quais se pretendam realizar previsões.

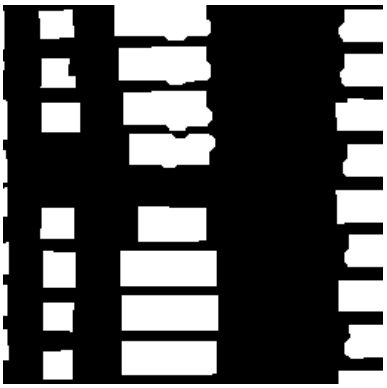
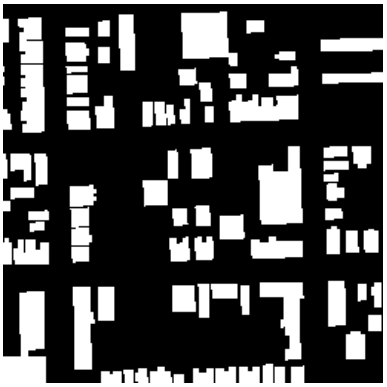
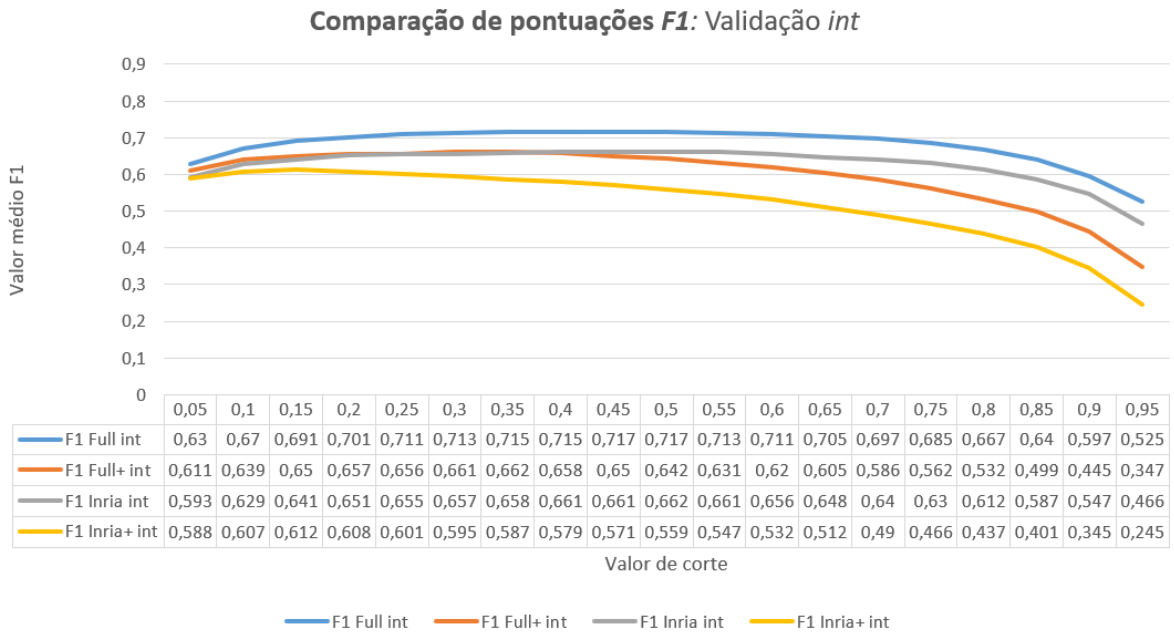
Exemplo de previsão *Inria* em *shp*	**Exemplo de previsão *Full* em *shp***	**Máscara verdadeira**
		
Precisão: 0.834 **Revocação:** 0.928 **Pontuação F1:** 0.878	**Precisão:** 0,996 **Revocação:** 0,983 **Pontuação F1:** 0,989	
		
Precisão: 0,668 **Revocação:** 0,759 **Pontuação F1:** 0,710	**Precisão:** 0,711 **Revocação:** 0,802 **Pontuação F1:** 0,754	
		
Precisão: 0,666 **Revocação:** 0,338 **Pontuação F1:** 0,449	**Precisão:** 0,513 **Revocação:** 0,537 **Pontuação F1:** 0,525	

Figura 4.3 - Exemplos de previsão com experiência *Inria* e *Full* no conjunto de validação *int*

A Figura 4.3 em conjunto com as análises realizadas às Figuras 4.1 e 4.2, sustentam também a afirmação da maior capacidade de generalização do modelo *Full*, quando comparado com o *Inria*, para segmentação em regiões com diferentes caraterísticas. Além do referido é possível também constatar-

se a falta de sensibilidade de algumas máscaras verdadeiras que identificam, por exemplo, telhados de edifícios cobertos por árvores, que são impossíveis de segmentar pelo modelo e influenciam negativamente as métricas.

Quanto aos modelos *Full+* e *Inria+*, estes também foram validados com este conjunto de dados, mas como era de esperar os valores registados são sempre inferiores quando comparados com as suas bases *Full* e *Inria*. Neste caso previa-se que da mesma maneira que os pesos dos modelos precisaram de ser ajustados para prever melhor em território nacional, o contrário se manifestaria neste conjunto porque os modelos se afastam das caraterísticas (pesos) que aprenderam para se ajustarem às novas introduzidas. Este facto pode ser visualizado percetivelmente na Figura 4.4 que compara as pontuações médias F1 dos quatro modelos, aplicados ao conjunto *int*, ao longo de valores de corte da imagem binária prevista. Pode-se observar que nenhum modelo retreinado regista valores médios mais elevados que as suas bases de treino *Inria* ou *Full*.



	0,05	0,1	0,15	0,2	0,25	0,3	0,35	0,4	0,45	0,5	0,55	0,6	0,65	0,7	0,75	0,8	0,85	0,9	0,95
F1 Full int	0,63	0,67	0,691	0,701	0,711	0,713	0,715	0,715	0,717	0,717	0,713	0,711	0,705	0,697	0,685	0,667	0,64	0,597	0,525
F1 Full+ int	0,611	0,639	0,65	0,657	0,656	0,661	0,662	0,658	0,65	0,642	0,631	0,62	0,605	0,586	0,562	0,532	0,499	0,445	0,347
F1 Inria int	0,593	0,629	0,641	0,651	0,655	0,657	0,658	0,661	0,661	0,662	0,661	0,656	0,648	0,64	0,63	0,612	0,587	0,547	0,466
F1 Inria+ int	0,588	0,607	0,612	0,608	0,601	0,595	0,587	0,579	0,571	0,559	0,547	0,532	0,512	0,49	0,466	0,437	0,401	0,345	0,245

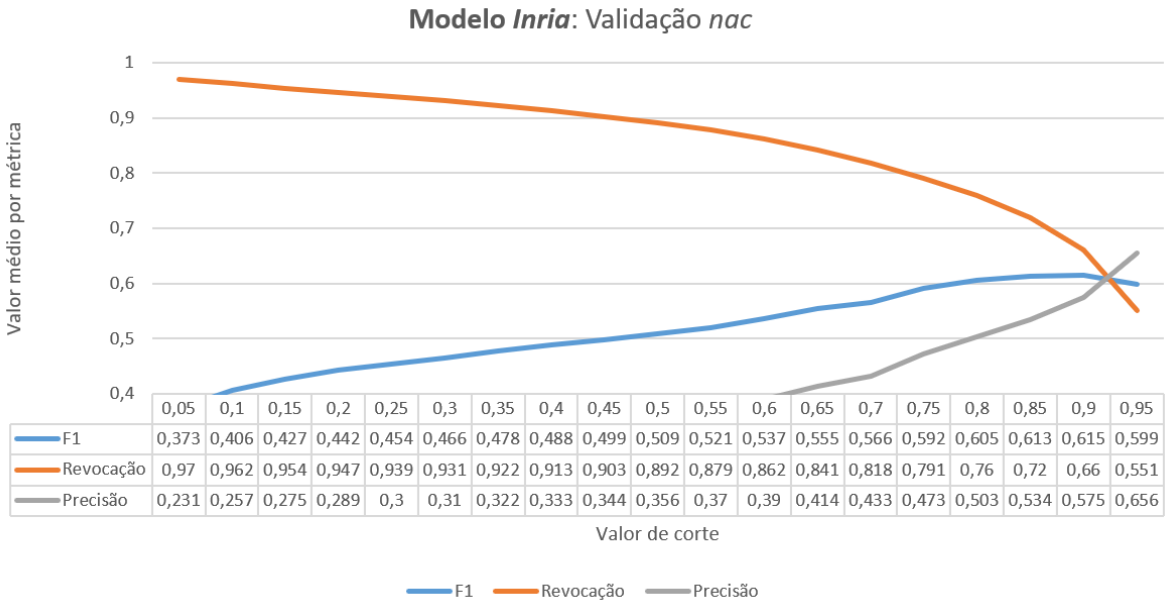
Figura 4.4 - Comparação de pontuações F1 na validação *int*

4.1.2 Validação *nac*

O conjunto de validação *nac* foi descrito em detalhe na Seção 3.6. É composto por cinquenta imagens de alta resolução e respetivas máscaras de edifícios de uma zona de Samora Correia em Portugal. Estas imagens apresentam caraterísticas de telhados desafiadoras para os modelos *Full* e *Inria* devido ao facto de se referirem a uma localização geográfica que apresenta caraterísticas totalmente diferentes das utilizadas para treino.

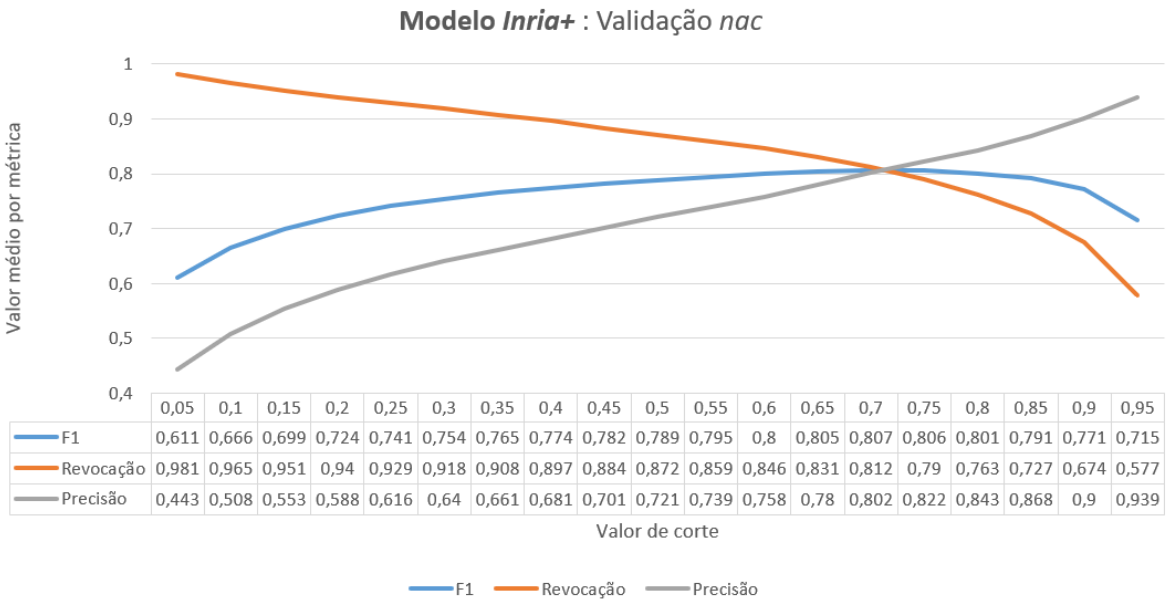
Ao contrário da validação *int* nenhuma destas imagens foi colocada de parte de qualquer conjunto de treino. Assim, o objetivo principal desta seção foi analisar os resultados de validação para os modelos *Full+* e *Inria+* de forma a ser analisada a capacidade de generalização dos modelos e também a contribuição do retreino dos modelos *Full* e *Inria* com o pequeno conjunto de treino nacional descrito na Seção 3.2.3.

As Figuras 4.5 e 4.6 apresentam gráficos que sumarizam o resultado das três métricas para o modelo *Inria* e *Inria+*. O *Inria*, como era de se esperar, apresenta valores abaixo dos resultantes da validação *int* com um valor máximo de F1 de 0,615, menos 0,047 quando comparado, demonstrando uma fraca capacidade de generalização. É também notável a falta de precisão do modelo que começa com um valor médio de 0,231. Por outro lado, e após retreinado o modelo *Inria* com o conjunto nacional, obteve-se o modelo *Inria+* que apresentou grandes melhorias neste contexto (Figura 4.5). O valor mais elevado de F1 é de 0,807, com destaque para a evolução de 0,227 do valor ótimo de precisão deste modelo quando comparado com o *Inria* (Figura 4.5).



	0,05	0,1	0,15	0,2	0,25	0,3	0,35	0,4	0,45	0,5	0,55	0,6	0,65	0,7	0,75	0,8	0,85	0,9	0,95
F1	0,373	0,406	0,427	0,442	0,454	0,466	0,478	0,488	0,499	0,509	0,521	0,537	0,555	0,566	0,592	0,605	0,613	0,615	0,599
Revocação	0,97	0,962	0,954	0,947	0,939	0,931	0,922	0,913	0,903	0,892	0,879	0,862	0,841	0,818	0,791	0,76	0,72	0,66	0,551
Precisão	0,231	0,257	0,275	0,289	0,3	0,31	0,322	0,333	0,344	0,356	0,37	0,39	0,414	0,433	0,473	0,503	0,534	0,575	0,656

Figura 4.5 - Resultados *Inria* na validação *nac*



	0,05	0,1	0,15	0,2	0,25	0,3	0,35	0,4	0,45	0,5	0,55	0,6	0,65	0,7	0,75	0,8	0,85	0,9	0,95
F1	0,611	0,666	0,699	0,724	0,741	0,754	0,765	0,774	0,782	0,789	0,795	0,8	0,805	0,807	0,806	0,801	0,791	0,771	0,715
Revocação	0,981	0,965	0,951	0,94	0,929	0,918	0,908	0,897	0,884	0,872	0,859	0,846	0,831	0,812	0,79	0,763	0,727	0,674	0,577
Precisão	0,443	0,508	0,553	0,588	0,616	0,64	0,661	0,681	0,701	0,721	0,739	0,758	0,78	0,802	0,822	0,843	0,868	0,9	0,939

Figura 4.6 - Resultados *Inria+* na validação *nac*

Na Figura 4.7 apresentam-se dois exemplos validados desta evolução notória entre o modelo *Inria* e o modelo *Inria+*. O modelo *Inria* apresenta-se bastante defeituoso e tem bastante dificuldade em manter uma precisão alta. No entanto, os resultados previstos pelo *Inria+*, demonstram uma grande melhoria,

identificando quase todos pixéis da imagem correspondentes a edifícios. É importante ter em conta que as máscaras nem sempre correspondem estritamente a todos os pixéis de edifício na imagem, podendo condicionar melhores pontuações, especialmente devido à capacidade de deteção do modelo independentemente da cor do telhado do edifício.

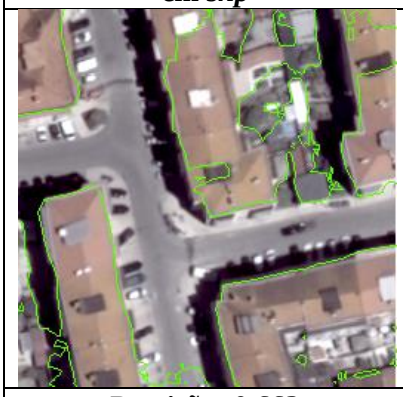
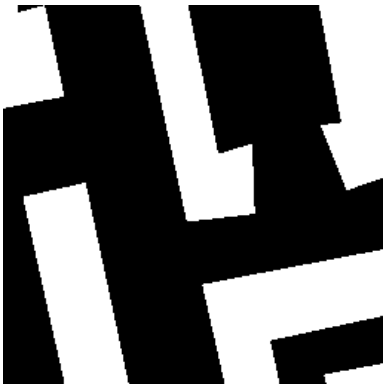
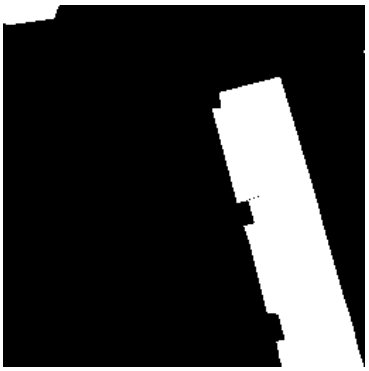
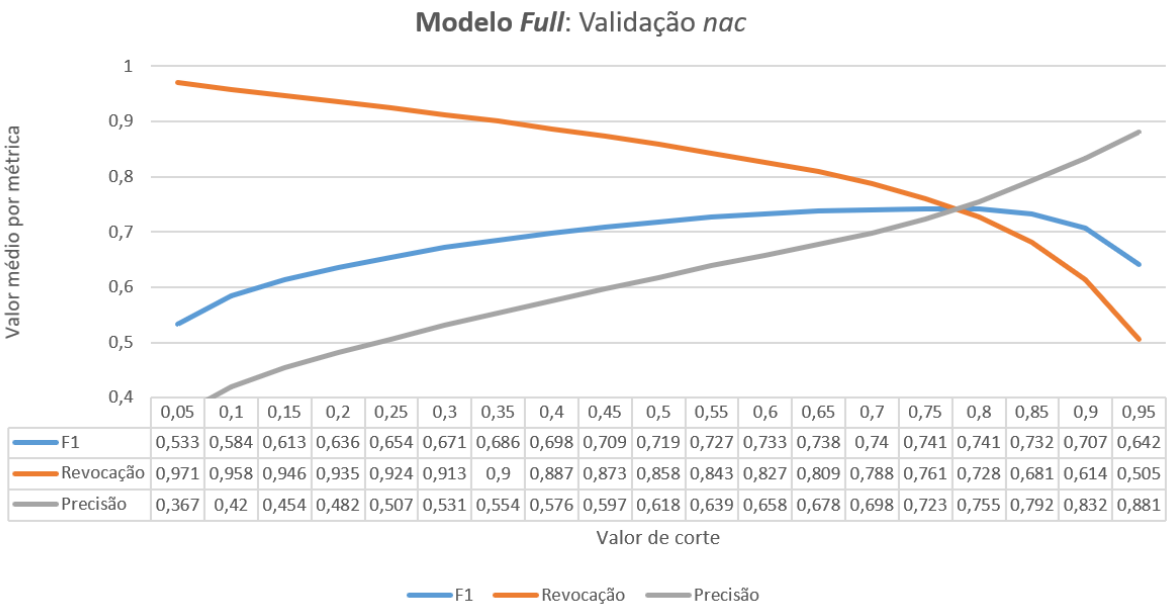
Exemplo de previsão *Inria* em *shp*	**Exemplo de previsão *Inria+* em *shp***	**Máscara verdadeira**
		
Precisão: 0,663 **Revocação:** 0,866 **Pontuação F1:** 0,751	**Precisão:** 0,896 **Revocação:** 0,911 **Pontuação F1:** 0,903	
		
Precisão: 0,283 **Revocação:** 0,931 **Pontuação F1:** 0,434	**Precisão:** 0,980 **Revocação:** 0,927 **Pontuação F1:** 0,953	

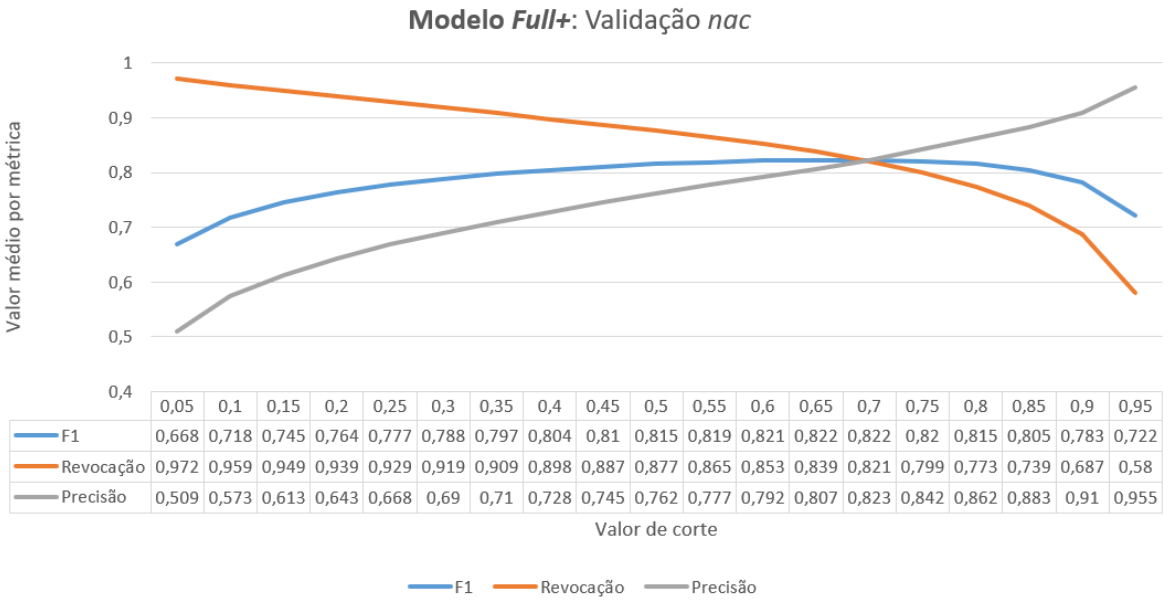
Figura 4.7 - Exemplos de previsão com experiência *Inria* e *Inria+* no conjunto de validação *nac*

Por sua vez, nas Figuras 4.8 e 4.9 são apresentados os valores médios por métrica referentes à validação *nac* para o modelo *Full* e *Full+* respetivamente. Curiosamente e ao contrário do modelo *Inria*, o modelo *Full* apresenta valores de F1 mais elevados nesta validação do que os registados na validação *int*, nomeadamente no cruzamento das métricas de precisão e revocação, em que se regista um valor de 0,741, uma diferença de 0,024 quando comparada com os resultados *Full* da Figura 4.2. Neste caso, assume-se que a melhoria se deve à maior capacidade de generalização deste modelo devido ao enorme conjunto de treino utilizado para o treinar e também devido ao facto do conjunto de validação *nac* não ser tão variável como o *int* em termos de caraterísticas, resolução e escala das imagens. O modelo *Full+*, treinado com os pesos *Full* inicializados, apresentou os melhores resultados na validação deste conjunto, registando uma pontuação F1 máxima de 0,822.



	0,05	0,1	0,15	0,2	0,25	0,3	0,35	0,4	0,45	0,5	0,55	0,6	0,65	0,7	0,75	0,8	0,85	0,9	0,95
F1	0,533	0,584	0,613	0,636	0,654	0,671	0,686	0,698	0,709	0,719	0,727	0,733	0,738	0,74	0,741	0,741	0,732	0,707	0,642
Revocação	0,971	0,958	0,946	0,935	0,924	0,913	0,9	0,887	0,873	0,858	0,843	0,827	0,809	0,788	0,761	0,728	0,681	0,614	0,505
Precisão	0,367	0,42	0,454	0,482	0,507	0,531	0,554	0,576	0,597	0,618	0,639	0,658	0,678	0,698	0,723	0,755	0,792	0,832	0,881

Figura 4.8 - Resultados *Full* na validação *nac*



	0,05	0,1	0,15	0,2	0,25	0,3	0,35	0,4	0,45	0,5	0,55	0,6	0,65	0,7	0,75	0,8	0,85	0,9	0,95
F1	0,668	0,718	0,745	0,764	0,777	0,788	0,797	0,804	0,81	0,815	0,819	0,821	0,822	0,822	0,82	0,815	0,805	0,783	0,722
Revocação	0,972	0,959	0,949	0,939	0,929	0,919	0,909	0,898	0,887	0,877	0,865	0,853	0,839	0,821	0,799	0,773	0,739	0,687	0,58
Precisão	0,509	0,573	0,613	0,643	0,668	0,69	0,71	0,728	0,745	0,762	0,777	0,792	0,807	0,823	0,842	0,862	0,883	0,91	0,955

Figura 4.9 - Resultados *Full+* na validação *nac*

Dois exemplos claros da melhoria de qualidade da previsão do modelo *Full*, após retreino com o conjunto de dados nacional, podem ser visualizados na Figura 4.10. É possível constatar, no modelo *Full+*, grandes melhorias no que toca à revocação, ou seja, uma maior capacidade de prever todos os edifícios, tal como identificado pela máscara *ground truth*. Apesar da atenção dada na escolha das imagens de validação, os resultados da revocação podem ser afetados, neste conjunto de dados, pelo facto de este ter sido extraído de um conjunto criado e utilizado para um treino em que o objetivo seria identificar apenas classes de edifícios com telha vermelha. Por isso, e como exemplo, na segunda linha da Figura 4.10, pode-se assumir que este resultado de revocação é afetado porque ambos os modelos segmentam um edifício cuja telha não é vermelha. Este exemplo demonstra também a capacidade do modelo para segmentar edifícios de variados tipos, não só pela sua cor, mas também pela sua textura e forma. Outro fator que influencia a correta validação pode ser o facto do recorte da imagem em mosaicos

(256 x 256 x 3) não garantir, por vezes, caraterísticas suficientes para a segmentação de entidades que aparecem recortadas.

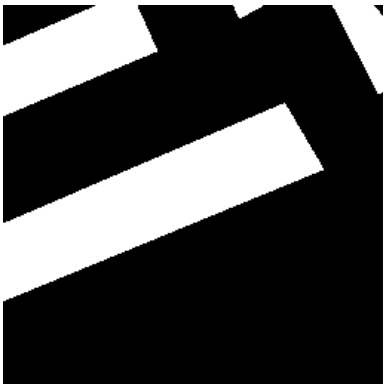
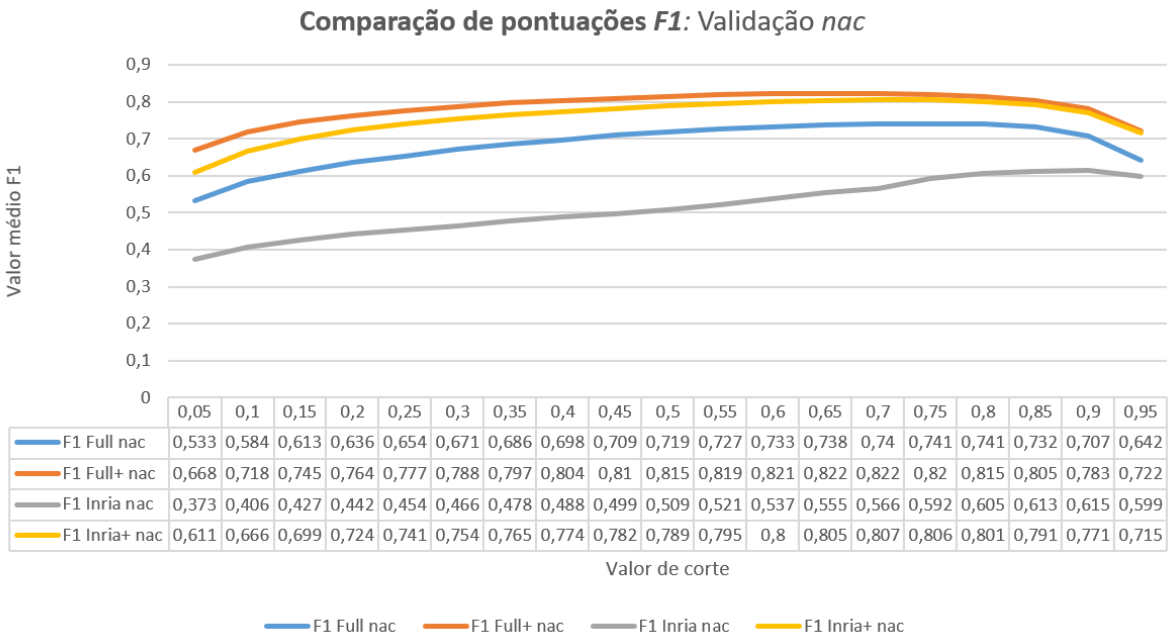
Exemplo de previsão *Full* em *shp*	**Exemplo de previsão *Full+* em *shp***	**Máscara verdadeira**
		
Precisão: 0,982 **Revocação:** 0,674 **Pontuação F1:** 0,799	**Precisão:** 0,979 **Revocação:** 0,783 **Pontuação F1:** 0,870	
		
Precisão: 0,934 **Revocação:** 0,861 **Pontuação F1:** 0,896	**Precisão:** 0,895 **Revocação:** 0,976 **Pontuação F1:** 0,934	

Figura 4.10 - Exemplos de previsão com experiência *Full* e *Full+* no conjunto de validação *nac*

É possível confirmar na Figura 4.11 que as experiências *Inria+* e *Full+* são as que obtêm os melhores resultados de pontuação F1 com a *Full+* a demonstrar o desempenho máximo. A estabilidade das curvas F1, ou a baixa variação dos valores médios, demonstra também a qualidade dos modelos sendo interessante apontar o caso da experiência *Full* que conseguiu manter uma precisão e revocação aceitáveis, o que demonstra a sua capacidade de generalização e justifica o desempenho da *Full+*. Como já foi referido, assume-se que esta capacidade de generalização se deve ao facto de este ter sido treinado com um conjunto de dados duas vezes maior e com uma cobertura mais ampla e diversificada.



	0,05	0,1	0,15	0,2	0,25	0,3	0,35	0,4	0,45	0,5	0,55	0,6	0,65	0,7	0,75	0,8	0,85	0,9	0,95
F1 Full nac	0,533	0,584	0,613	0,636	0,654	0,671	0,686	0,698	0,709	0,719	0,727	0,733	0,738	0,74	0,741	0,741	0,732	0,707	0,642
F1 Full+ nac	0,668	0,718	0,745	0,764	0,777	0,788	0,797	0,804	0,81	0,815	0,819	0,821	0,822	0,822	0,82	0,815	0,805	0,783	0,722
F1 Inria nac	0,373	0,406	0,427	0,442	0,454	0,466	0,478	0,488	0,499	0,509	0,521	0,537	0,555	0,566	0,592	0,605	0,613	0,615	0,599
F1 Inria+ nac	0,611	0,666	0,699	0,724	0,741	0,754	0,765	0,774	0,782	0,789	0,795	0,8	0,805	0,807	0,806	0,801	0,791	0,771	0,715

Figura 4.11 - Comparação de pontuações F1 na validação *nac*

4.1.3 Síntese dos Resultados de Validação

Para terminar a análise estatística, a Tabela 4.1 apresenta um sumário dos melhores resultados obtidos pelos modelos, ou seja, os valores de corte que maximizaram o desempenho em validação dos modelos, em termos de pontuação F1 e consequentemente revocação e precisão. O modelo *Full* é o que consegue manter os melhores resultados entre a validação *int* e *nac* devido à sua grande capacidade de generalização. Por sua vez, o modelo *Inria* apresenta dificuldades em manter estes valores. Os modelos retreinados *Inria+* e *Full+* apresentam os melhores resultados na validação *nac* mas não conseguem ficar à frente das bases *Full* e *Inria* na validação *int*.

Tabela 4.1 - Síntese dos melhores resultados de validação registados em cada experiência

Modelo	**Pontuação F1**	**Precisão**	**Revocação**	**Valor de corte**	**Conjunto de validação**
Inria	0,662	0,66	0,664	0,5	*int*
Inria+	0,612	0,622	0,602	0,15	*int*
Full	0,717	0,721	0,712	0,5	*int*
Full+	0,661	0,672	0,651	0,3	*int*
Inria	0,615	0,575	0,66	0,9	*nac*
Inria+	0,807	0,802	0,812	0,7	*nac*
Full	0,741	0,723	0,761	0,75	*nac*
Full+	**0,822**	0,823	0,821	0,7	*nac*

4.2 Fase de Teste

Nesta seção são explorados alguns resultados de segmentação obtidos pelo modelo *Full+* em imagens de diversas dimensões e sem qualquer tipo de máscara de referência ou *ground truth* para se realizar uma análise estatística. Os resultados aqui expostos confirmam os bons valores registados pelo modelo *Full+* na validação *nac*.

Os resultados de teste previstos pelo modelo são realizados automaticamente com recurso ao programa desenvolvido *reconst.py*, que torna possível a realização desta operação em imagens de variadas dimensões e formatos porque, como referido (Seção 3.7), a arquitetura *U-net*, realiza as melhores previsões em imagens de dimensões iguais às que foram utilizadas para treino, sendo por isso necessário realizar uma partição da imagem original em vários mosaicos desta dimensão e com uma certa sobreposição, e posterior reconstrução da dimensão original, através da concatenação das imagens binárias de previsão correspondentes aos vários mosaicos, de forma a ser obtido o melhor resultado possível.

As imagens utilizadas para teste foram recolhidas com recurso à aplicação de visualização e descarregamento de imagens satélite geocodificadas *Google Earth* para o ano 2022 e serviços *WMTS* de ortofotos realizadas por coberturas aéreas sistemáticas relativas aos anos de 2018 e 2021, abertas e publicadas no Sistema Nacional de Informação Geográfica (SNIG) pela DGT.

Na Figura 4.12 pode-se visualizar um resultado de teste aplicado numa imagem de grande escala. Nesta Tabela pode-se visualizar uma imagem *RGB* obtida por satélite e a respetiva previsão dos edifícios em formato binário. A imagem é referente à cidade de Lisboa em Portugal que apresenta uma grande irregularidade de edifícios e vários tipos e texturas de telhado. Apesar disso, os resultados visuais da previsão automática nesta imagem confirmam o bom desempenho e generalização do modelo, apresentando apenas ligeiras dificuldades na segmentação de topos com cores de tons castanhos, como se pode observar na zona do Chiado (lado direito da imagem de previsão da Tabela 4.5), ou cinzentos.

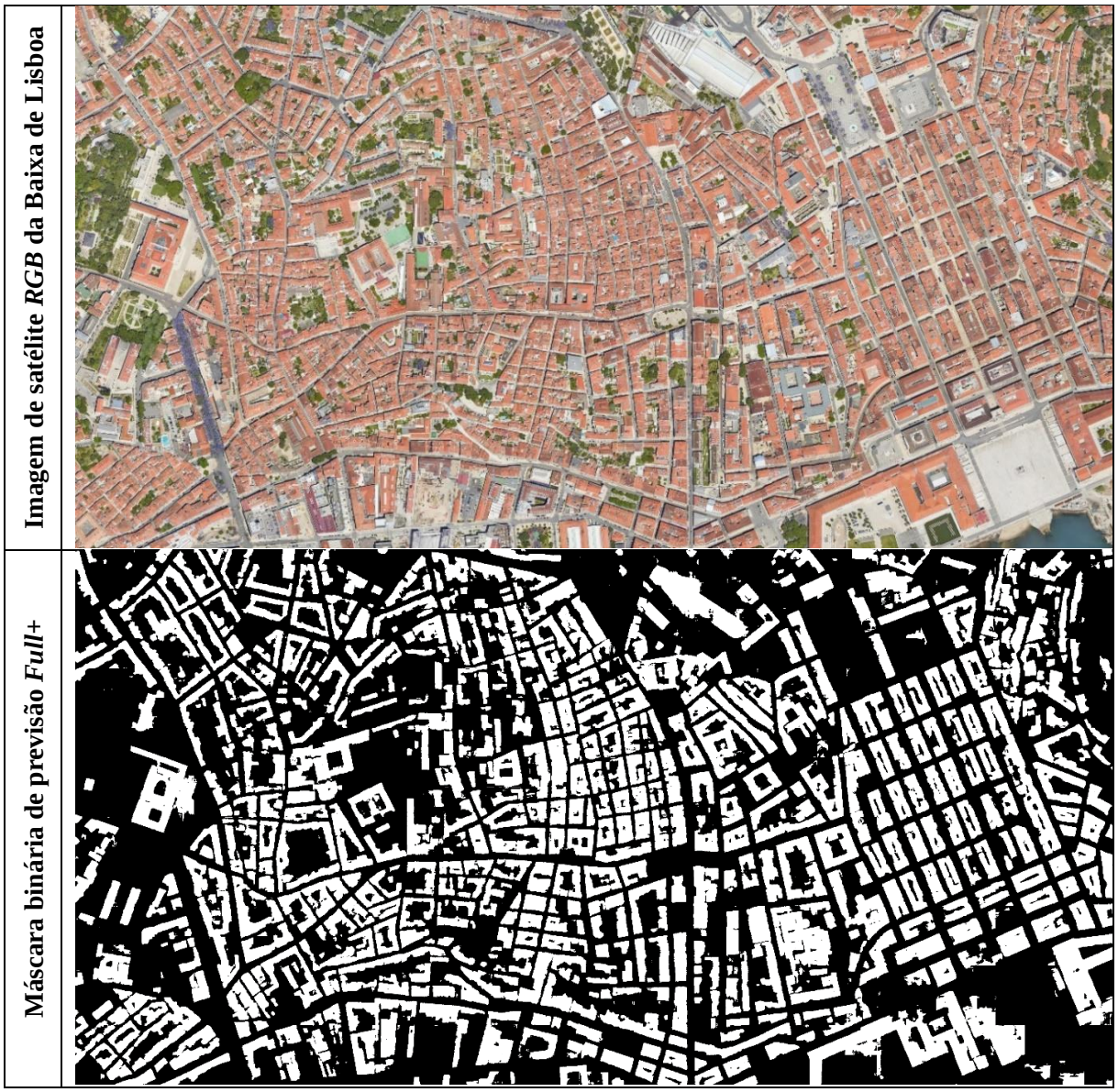


Figura 4.12 - Segmentação binária automática de edifícios de imagem de satélite *Google Earth* da Baixa de Lisboa

Por sua vez, foram também realizados testes em produtos de maior escala em quatro zonas de Lisboa, com recurso ao programa desenvolvido e a imagens de satélite *Google Earth*, que podem ser visualizados na Figura 4.13. Os resultados são considerados também bastante promissores e incentivam a experimentação aplicacional do modelo. É interessante a capacidade que este tem para segmentar edifícios com formas irregulares e pouco comuns, por exemplo, os do Parque da Saúde em Lisboa.

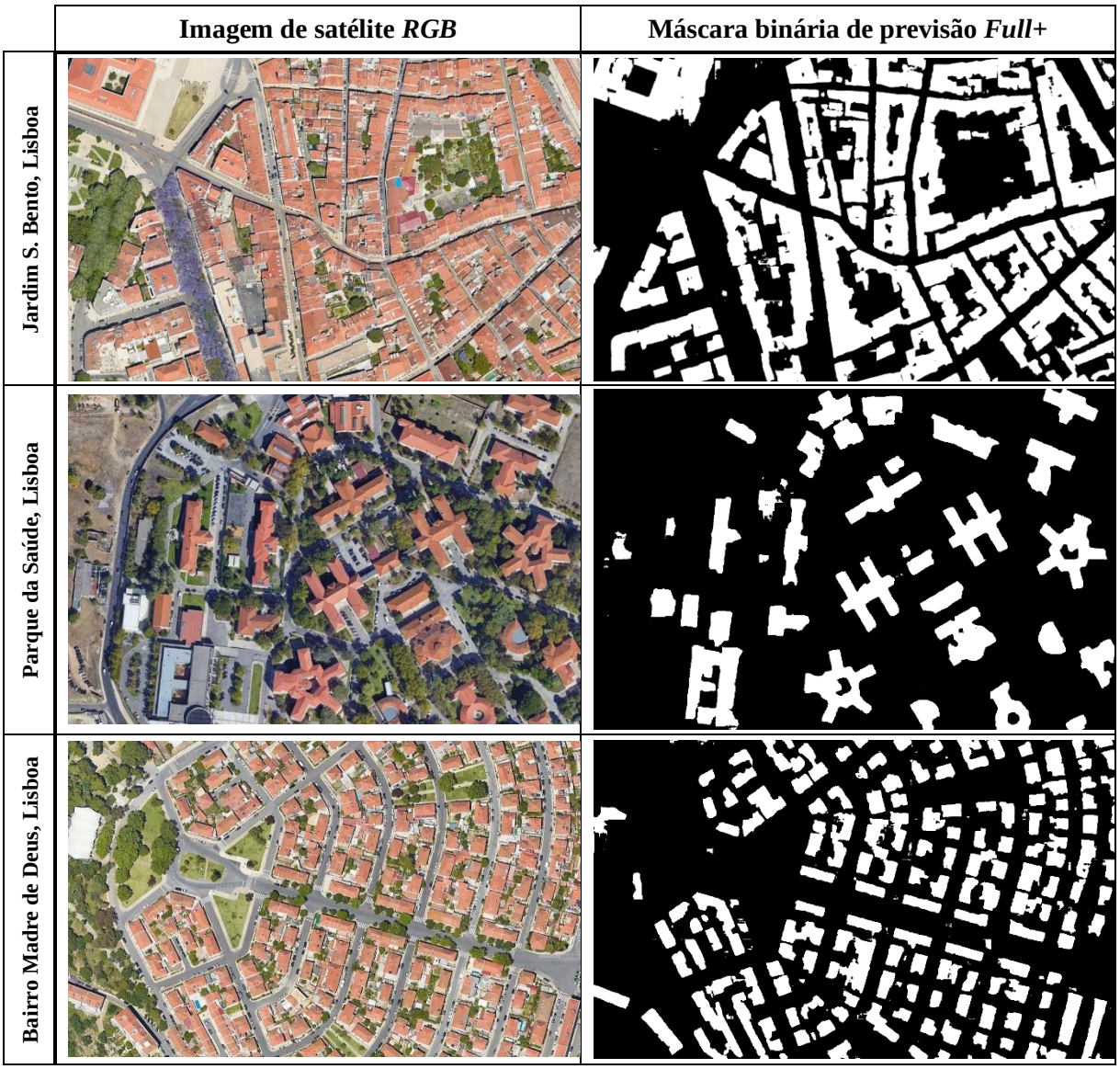


Figura 4.13 - Segmentação binária automática de imagens de satélite *Google Earth*

4.2.1 Exemplos Aplicacionais

O modelo ao ser utilizado pela ferramenta *reconst.py*, produz imagens binárias que correspondem à entidade geoespacial "edifício" em imagens *RGB* e podem ser utilizadas em SIG como o *ArcMap*. Nesta parte é explorada a capacidade aplicacional dos resultados obtidos nomeadamente no apoio à eficiência da produção cartográfica, análise espacial de dados, deteção de mudanças e na completagem de Sistemas Voluntários de Informação Geográfica (*Voluntary Geograaphic Systems, VGI*) como o *OpenStreetMap*.

A restituição de edifícios pode ser vista como uma das tarefas manuais mais morosa e necessária na produção de elementos cartográficos. Por essa razão, torna-se favorável a extração automática desta entidade geoespacial de forma a minimizar o tempo e trabalho de aquisição das mesmas. Através da análise dos resultados expostos na fase de validação, complementados com os da fase de teste, é possível afirmar que o modelo *Full+* pode ser útil na concretização desta tarefa, de forma rápida, automática e com um desempenho elevado.

Os resultados deste devem ser implementados num ambiente SIG como *ArcMap* podendo ser os *rasters* binários diretamente carregados e transformados em polígonos utilizando este *software*. Após a execução desta ferramenta os polígonos de previsão são criados e guardados em formato *shapefile*, podendo ser posteriormente sobrepostos à imagem à qual correspondem e que contém as mesmas dimensões, ou até mesmo a qualquer outro tipo de imagem georreferenciada, caso também sejam derivados de uma imagem corretamente georreferenciada, como será explorado em detalhe. Assim, torna-se possível a manipulação e edição dos resultados *shapefile*, sendo possível ajustá-los e até mesmo alterar a sua aparência e aspeto. É importante também referir que para a exemplificação de alguns dos testes aplicacionais que se seguem foram utilizadas ortofotos disponibilizadas pela DGT na plataforma SNIG. Devido às caraterísticas planimétricas dos produtos ortoretificados, derivadas normalmente de projeções diretas ou indiretas entre uma imagem aérea ou espacial e um modelo digital de superfície, a sua utilização garante a criação de produtos cartográficos topográficos nos quais se podem realizar medições e sobreposições com mapas base de forma precisa.

A Figura 4.14 é um dos muitos exemplos de falta de dados (assinalados a azul) nalguns visualizadores de informação geográfica como é o caso do *OpenStreetMap*. Este tipo de problema pode afetar as capacidades dos seus *LBS* e, por isso, é interessante explorar-se um pouco a aplicação que os modelos de segmentação automática podem ter na sua resolução.

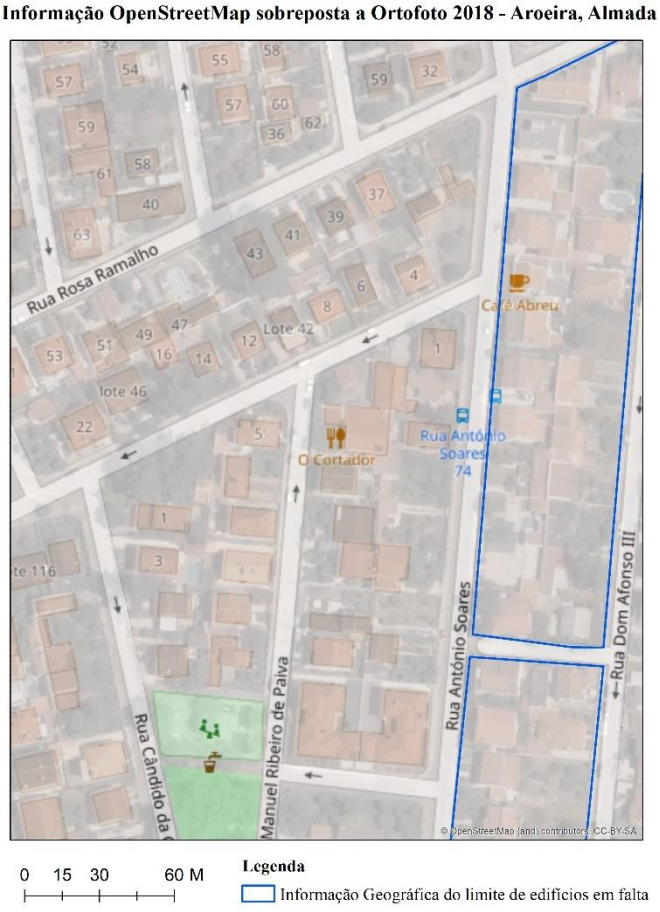


Figura 4.14 - Informação *OSM* sobreposta a ortofoto nacional de 2018

Com esse objetivo, a área da ortofoto correspondente à área a ser preenchida pode ser exportada do visualizador de dados do *ArcMap* e segmentada com o modelo *Full+*. Como a imagem binária estimada tem a mesma dimensão da ortoimagem é atribuída a mesma georreferenciação e imediata possibilidade

de integração num SIG, para posterior conversão *raster*-vetor. O resultado é apresentado na Figura 4.15. Os polígonos georreferenciados de previsão resultantes da operação descrita, podiam ser posteriormente ajustados com o editor de vértices do *ArcMap* e publicados na plataforma *OpenStreetMap*. Este é um exemplo que poderia ser realizado para uma grande cobertura, rapidamente e obtendo resultados aceitáveis que poderiam ser depois manualmente ajustados consoante o rigor necessário e utilizados para melhorar serviços abertos como este, apesar da limitação da qualidade das imagens, que se assume ser um dos principais obstáculos.

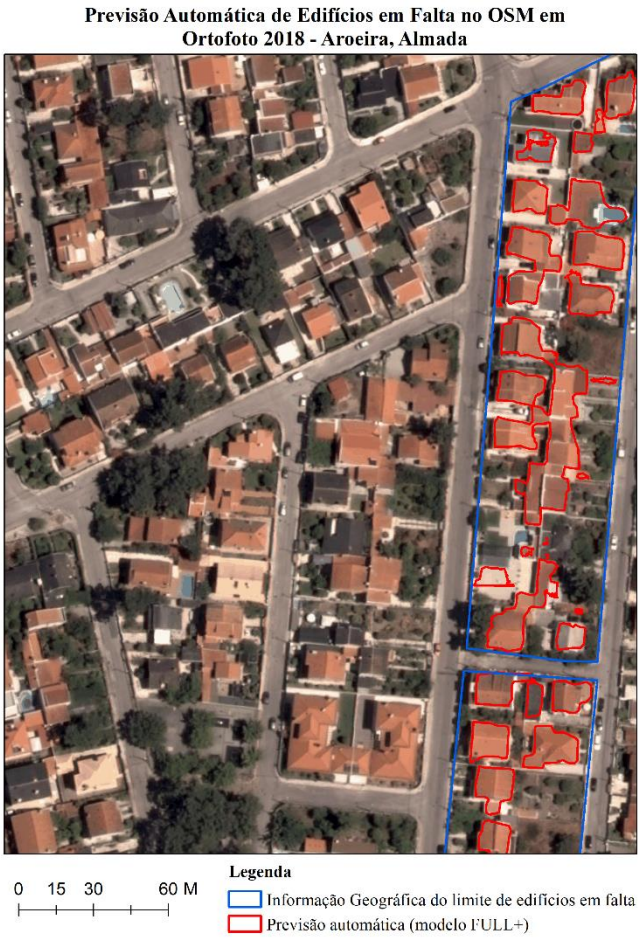


Figura 4.15 - Previsão automática do modelo *Full+* numa zona com informação em falta no *OSM*

O procedimento descrito e os programas desenvolvidos para aplicação do modelo criado, podem também ser bastante úteis no auxílio à geração de produtos cartográficos no geral, especialmente quando não são exigidas grandes escalas. A cartografia topográfica, por exemplo, pode ser definida como uma base de dados dos fenómenos naturais e artificiais que acontecem num território e que podem ser considerados importantes para o seu desenvolvimento, conhecimento e caraterização. Em Portugal, a entidade que define o modelo de dados da cartografia topográfica vetorial e de imagem é a DGT, sendo este exposto no documento "Normas e Especificações Técnicas para a Cartografia Topográfica Vetorial e de Imagem" [95]. Um dos temas que se mostra mais moroso em termos de restituição vetorial são as construções, especialmente a classe de objetos "Edificio". Tendo em conta que a restituição de pormenor pode ser realizada via ortorrectificação, podia ser também utilizado o serviço de ortofotos *WMS*, disponibilizado no *SNIG*, e os modelos criados de forma a se contribuir para uma restituição mais automatizada desta entidade, tal como exposto em parte na Figura 4.14 e 4.16.

A automatização da extração desta entidade pode ter também grande impacto na eficiência de criação de análises espaciais, cuja informação de edifícios possa ser vital, por exemplo, em casos de emergência como incêndios florestais em que a rápida análise de proximidade de edifícios de um certo foco de incêndio, com recurso a imagens de satélite ou VANT, se torna bastante importante. Pode demonstrar também uma grande utilidade quando existem muitos dados ou até dados com dimensões muito elevadas, no auxílio à criação de bases de dados espaciais, na deteção automática de mudanças, entre outros. O facto de ter sido desenvolvido um método que permite manter a georreferenciação das previsões garante também a possibilidade do cruzamento com outras entidades geoespaciais.

Finalmente, pode-se afirmar que foi criado um ambiente propício ao desenvolvimento de novos modelos ou até mesmo ao melhoramento dos criados. Todo o percurso foi descrito em detalhe e todas as ferramentas criadas e conjuntos de dados utilizados foram disponibilizados em formato aberto.

(Página deixada em branco)

5 Conclusão

O presente trabalho teve como objetivo principal introduzir e explorar as *CNN* e criar um ambiente eficiente para treino, validação e teste de modelos *ML* criados com capacidade de realizar a segmentação automática de edifícios em imagens de elevada resolução radiométrica e espacial. Recorreu-se à utilização da recente arquitetura *U-net* de segmentação de imagem, que se tem vindo a destacar pela sua contribuição em tarefas relacionadas com a segmentação de imagens de alta resolução obtidas utilizando técnicas espaciais e aéreas de aquisição de informação radiométrica e geoespacial, para se criarem quatro modelos de segmentação automática.

Foi realizada uma revisão detalhada do estado da arte com recurso a inúmeras referências bibliográficas, com especial interesse no funcionamento das *CNN*, e alguns dos mais recentes estudos que as associam à automatização de processos, que se mostraram benéficos para várias aplicações relacionadas com a extração e deteção automática de entidades geoespaciais. O funcionamento deste tipo de algoritmos de *ML* é bastante complexo, sendo por isso exposta uma análise detalhada das várias etapas que, em conjunto, tornam possível a sua correta implementação. Apresenta-se uma análise sintetizada do significado, contributo e processamento realizado por cada uma das várias camadas que podem compor uma *CNN*, bem como as técnicas e conceitos mais populares associados a cada uma.

Uma das técnicas mais importantes e utilizada constantemente no desenvolvimento deste projeto foi a programação em linguagem *Python*. Com recurso a esta, foram desenvolvidos nove programas que constituíram a implementação da metodologia própria de tratamento, processamento e avaliação de resultados em massa. Todas estas ferramentas produzidas encontram-se publicadas num repositório *GitHub* facilitando e incentivando a sua implementação em projetos futuros.

Constatou-se uma crescente disponibilização e disseminação de imagens aéreas e espaciais de alta resolução, através de múltiplos projetos e ferramentas desenvolvidos e publicados por outros autores em formato aberto, permitindo um maior acesso a dados com caraterísticas *big data*, que se mostraram vitais para a criação de modelos com altos desempenhos e capacidade de generalização. Foram descarregados da *internet* dezenas de *gygabites* de dados, relativos a conjuntos de dados de treino úteis para segmentação de edifícios em imagens aéreas e espaciais de alta resolução. Estes diferentes conjuntos de dados brutos foram alvo de tratamento automático, utilizando essencialmente as ferramentas *Python recorte.py* e *cleaner.py* desenvolvidas. Deste modo, garantiram-se amostras de treino homogéneas e passíveis de serem misturadas, ou seja, com dimensões normalizadas de (256 x 256) e pares de nomes de máscara e imagem associáveis e com nomes inequívocos.

Revelou-se também necessário garantir o processamento e armazenamento eficiente de todos estes dados, tendo sido utilizadas as ferramentas em nuvem *Google Colab* e *Google Drive* respetivamente e para esse efeito. Estas ferramentas foram cruciais, pelo que sem elas não teria sido possível criar modelos resultantes de treinos intensos, com os milhares de dados de treino recolhidos. A plataforma *Colab* destacou-se pela sua simplicidade e desempenho, sendo que o seu ambiente contém os módulos de *ML* mais populares pré-instalados e garante desempenhos computacionais gratuitos comparáveis com os registados em máquinas de milhares de euros. Os dados de treino tiveram de ser divididos em partes para não exceder os limites de memória *RAM* da versão gratuita do *Colab* e, por essa razão, realizou-se o treino com uma primeira parte e, de seguida, utilizou-se o modelo gerado para se retomar o treino com outra parte de treino que gerava outro modelo. Os modelos gerados eram retomados até não haverem mais partes de treino.

A aplicação dos programas referidos, foi essencial para normalizar os dados descarregados em dois conjuntos de treino, utilizados para criar, na fase de treino, dois dos quatro modelos experimentais de segmentação automática de edifícios: o *Full* e o *Inria*. Geraram-se ainda mais dois modelos no *Colab* com os pesos dos modelos *Full* e *Inria* inicializados: o *Full+* e o *Inria+*. Para este último treino desenvolveu-se um pequeno conjunto de treino composto por imagens e máscaras de território nacional, com edifícios restituídos manualmente de imagens do *Google Earth* no *ArcMap* e semiautomaticamente com recurso ao *OpenStreeMap, QGIS* e *ArcMap*. Após retreinados com este conjunto nacional, os modelos base mostraram grandes melhorias de desempenho na validação, tornando as segmentações de edifícios em território nacional notavelmente melhores, devido a um ajuste dos pesos às caraterísticas e texturas de telhado mais comuns.

Uma vez concebidos os vários modelos de segmentação de edifícios de imagens aéreas e espaciais, estes foram analisados e comparados com recurso a dois conjuntos de dados de validação criados para o efeito e com o objetivo de permitir a avaliação da qualidade e influência das várias técnicas de treino seguidas, especialmente em termos de generalização: o *int* e o *nac*. O conjunto de validação *int* serviu, essencialmente, para comparar os modelos base *Full* e *Inria*, misturando imagens de diversas localizações e com caraterística de edifício variadas, enquanto que o *nac* permitiu realizar uma avaliação mais detalhada do desempenho das técnicas de treino aplicadas. Pôs-se em prática um programa criado em *Pyhton* que implementa três métricas (precisão, revocação e pontuação F1), comparando e pontuando os resultados de previsão de cada modelo nas imagens de cada um destes conjuntos com as suas máscaras rotuladas.

É notável a grande capacidade de generalização da experiência *Full* quando comparada com a *Inria*. É possível concluir que esta beneficiou significativamente da mistura de vários conjuntos de dados de treino com caraterísticas diferentes e publicados por outros autores e na *web*, como se pode confirmar com os valores máximos de F1 no conjunto de validação *int* de 0,72 e 0,66 respetivamente. O valor máximo da pontuação F1 de 0,74, registado pelo modelo *Full* no conjunto *nac*, é também indicador desta capacidade de generalização. Apesar disso, tornou-se percetível que, para se aplicar um modelo que demonstre desempenhos superiores numa localização específica, se poderia recorrer a um pequeno conjunto de dados, de forma a melhorar as saídas. Assim, as experiências *Full+* e *Inria+* corresponderam a retreinos das correspondentes *Full* e *Inria*, como já foi referido. Estes modelos retreinados demonstraram resultados muito bons no conjunto de validação *nac* e grandes melhorias quando comparados com os que foram utilizados como base dos seus treinos. O modelo *Full+* regista valores máximos de pontuação F1 de 0,82 e precisões acima dos 0,9 em alguns casos, o que o coloca muito à frente do *Full*, que regista um valor médio máximo de F1 de 0,74 no mesmo conjunto. Por sua vez, o modelo *Inria+* foi o que registou a maior melhoria neste contexto de retreino, reforçando os benefícios da aplicação desta técnica de ajuste de pesos. A sua pontuação máxima média de F1 no conjunto de validação *nac* é 0,81, uma melhoria de 0,19 quando comparadas com a pontuação de 0,62 do modelo base *Inria*. No entanto, estes modelos só mostram melhorias em território nacional, como foi comprovado pela comparação de resultados médios de validação no conjunto *nac* e *int*, pelo que os valores de pontuação F1 máximos no *int* foram de 0,62 para o *Inria+* e 0,66 para o *Full+*. Assume-se que esta técnica de treino dos modelos diminui a capacidade de generalização, mas especializa o modelo para um certo território, podendo-se tirar proveito da capacidade da arquitetura *U-net* aprender rapidamente com poucas amostras.

No final, procedeu-se à análise de exemplos aplicacionais que podem beneficiar alguns processos ligados à Engenharia Geoespacial e que constituíram a fase de teste, cujo o objetivo passou também por confirmar os resultados registados na validação. Apresentaram-se técnicas que podem permitir a implementação dos modelos criados em SIG. O programa desenvolvido em *Python* de teste (*reconst.py*)

mostrou-se bastante útil nesta fase, permitindo a segmentação de imagens de qualquer tipo de dimensões. Com este, criou-se a possibilidade de aplicar os modelos a imagens aéreas e espaciais de dimensões variadas. Foi também exposto um método que permite a conversão das imagens binárias de previsão em produtos georreferenciados com possibilidade de serem transformados em polígonos *shapefile* georreferenciados, com recurso ao *ArcMap*.

Um dos maiores obstáculos para a geração de modelos que produzam resultados melhores, acaba por ser o facto da partilha de informação de treino ser geralmente escassa ou difícil de adquirir e de pouca qualidade, condicionando a capacidade deste tipo de modelos de *ML*. Esta falta de distribuição de informação com qualidade superior acaba por condicionar as inúmeras possibilidades que podem advir destas técnicas profundas de IA. A criação de rótulos é uma tarefa manual bastante morosa e uma distribuição aberta e mais generalizada destes recursos poderia vir a melhorar ainda mais os desempenhos dos algoritmos de *DL*. Além disso, os crescentes desenvolvimentos tecnológicos, especialmente na área da computação gráfica, apenas reforçam cada vez mais a necessidade desta matéria ser aplicada e estudada para solucionar problemas reais, de forma artificial e com desempenhos superiores aos convencionais. No entanto, o custo das componentes computacionais necessárias à criação de modelos de *DL*, que apresentem bons resultados, é bastante elevado e acaba também por condicionar o desenvolvimento aprimorado destes. O *Colab* foi a melhor solução encontrada para este problema, disponibilizando acesso em nuvem a poderes de processamento bastante elevados, mas ficando ainda um pouco atrás na quantidade de *RAM* disponibilizada.

Em síntese, a tarefa de segmentação de edifícios realizou-se com bastante sucesso e todos os passos necessários à sua concretização se encontram expostos e partilhados. Assim, é esperado que no futuro esta tese possa vir a contribuir para a implementação e estudo de várias outras arquiteturas e objetivos de segmentação em imagens aéreas e espaciais de alta resolução. Muitas outras entidades geoespaciais podem ser segmentadas e ter aplicações variadas, nomeadamente estradas e uso do solo. Além disso, existe a necessidade de prosseguir um estudo mais profundo e atualizado acerca das várias arquiteturas de segmentação automática existentes, de forma a serem comparados modelos validados em conjuntos iguais e treinados com os mesmos dados, para que se possam retirar conclusões acerca das que se ajustam melhor a certos cenários de extração de entidades. Os resultados aqui apresentados podem ser melhorados através do acesso a um ambiente em que não haja restrições tão elevadas de memória e a dados georreferenciados abertos relativos a imagens e respetivas entidades geoespaciais, através da sua publicação generalizada em projetos abertos como o *OpenStreeMap* e SNIG. As versões pagas *Colab Pro* e *Colab Pro+* podem ser também uma mais valia, porque permitem acesso a mais memória *RAM*, execução em segundo plano e *GPU* ainda mais poderosas na nuvem, a um preço bastante acessível e competitivo, mas infelizmente, de momento, não é possível comprar os serviços em Portugal.

(página deixada em branco)

6 Referências

[1] Szegedy, C. Toshev, A. Erhan, D. (2013). Deep Neural Networks for object detection, Advances in Neural Information Processing Systems. Advances in Neural Information Processing Systems, 2553- 2561. ISBN: 9781632660244. https://papers.nips.cc/paper/2013/hash/f7cade80b7cc92b991cf4d2806d6bd78-Abstract.html

[2] He, K. Zhang, X. Ren, S. Sun, J. (2016). Deep residual learning for image recognition. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 770–778. https://doi.org/10.1109/CVPR.2016.90

[3] Girshick, R. Donahue, J. Darrell, T. Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. IEEE Conference on Computer Vision and Pattern Recognition, 580-587. https://doi.org/10.48550/arXiv.1311.2524

[4] Long, J. Shelhamer, E. Darrell, T. (2014). Fully convolutional networks for semantic segmentation. 2015 IEEE Conference on Computer Vision and Pattern Recognition, 3431-3440. https://doi.org/10.48550/arXiv.1411.4038

[5] Maxwell, A. Warner, T. Fang, F. (2017). Implementation of machine-learning classification in remote sensing: an applied review. International Journal of Remote Sensing, 39(9), 2784-2817. https://doi.org/10.1080/01431161.2018.1433343

[6] Khelfi, L. Mignotte, M. (2020). Deep Learning for Change Detection in Remote Sensing Images: Comprehensive Review and Meta-Analysis. IEEE Access, 8, 126385-126400. https://doi.org/10.1109/ACCESS.2020.3008036

[7] Cao, D. Xing, H. Wong, M. Kwan, M. Xing, H. Meng, Y. (2021). A Stacking Ensemble Deep Learning Model for Building Extraction from Remote Sensing Images. Remote Sensing, 13(19). https://doi.org/10.3390/rs13193898

[8] European Comission. (2019). Promoting the international competitiveness of European Remote Sensing companies through Cross-cluster collaboration. CORDIS EU website. https://cordis.europa.eu/project/id/824478/reporting

[9] European Comission. (2022). Helping land managers make sense of remote sensing data. European Commission website. https://ec.europa.eu/research-and-innovation/en/projects/success-stories/all/helping-land-managers-make-sense-remote-sensing-data

[10] Landsupport. (n.d). Project overview and objectives. Landsupport website. https://www.landsupport.eu/project/

[11] European Comission. (2022). From Copernicus big data to Extreme Earth Analytics. CORDIS EU website. https://cordis.europa.eu/project/id/825258

[12] European Comission. (n.d). Horizon. 2020. European Commission website. https://research-and-innovation.ec.europa.eu/funding/funding-opportunities/funding-programmes-and-open-calls/horizon-2020_en

[13] Google. (n.d). Google Colab. Google Colaboratory website. https://colab.research.google.com/

[14] Turing, A. M. (1950). Computing Machinery and Intelligence. Mind, LIX(236), 443-460. https://doi.org/10.1093/mind/LIX.236.433

[15] Géron, A. (2019). Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media, Inc. ISBN: 9781492032649.

[16] Wang, P. Fan, E. Wang, P. (2021). Comparative analysis of image classification algorithms based on traditional machine learning and deep learning. Pattern Recognition Letters, 141, 61-67. https://doi.org/10.1016/j.patrec.2020.07.042

[17] IBM. (2020). Geospatial data definition. IBM website. https://www.ibm.com/topics/geospatial-data

[18] Bioucas-Dias, J. Plaza, A. Camps-Valls, G. Scheunders, P. Nasrabadi, N. Chanussot, J. (2013). Hyperspectral Remote Sensing Data Analysis and Future Challenges. IEEE Geoscience and Remote Sensing Magazine, 1(2), 6-36. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/MGRS.2013.2244672

[19] Homer, C. Huang, C. Yang, L. Wylie, B. Coan, M. (2004). Development of a 2001 National Land-Cover Database for the United States. Photogrammetric Engineering & Remote Sensing 70(7), 829-840. American Society for Photogrammetry and Remote Sensing. https://doi.org/10.14358/PERS.70.7.829

[20] Delua, J. (2021). Supervised vs. Unsupervised Learning: What's the Difference? IBM. https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning

[21] Shamsoshoaraa, A., Afghah, F., Razi, A., Zheng, L., Fulé, P., Blasch, E. (2020). Aerial Imagery Pile burn detection using Deep Learning: the FLAME dataset. Computer Networks, 193. https://doi.org/10.1016/j.comnet.2021.108001

[22] Engelen, J., Hoos, H. (2020). A survey on semi-supervised learning. Machine Learning. Mach Learn 109, 373–440. https://doi.org/10.1007/s10994-019-05855-6

[23] Li, Y. (2018). Deep Reinforcement Learning: An Overview. Proceedings of SAI Intelligent Systems Conference, 426-440. https://doi.org/10.1007/978-3-319-56991-8_32

[24] Pokhrel, S. (2019). How Does Computer Understand Images? Towards Data Science. https://towardsdatascience.com/how-does-computer-understand-images-c1566d4537bf

[25] Belgiua, M. Drăguţ, L. (2016). Random forest in remote sensing: A review of applications and future directions. ISPRS Journal of Photogrammetry and Remote Sensing, 114, 24-31. https://doi.org/10.1016/j.isprsjprs.2016.01.011

[26] Mountrakis, G. Im, J. Ogole, C. (2011). Support vector machines in remote sensing: A review. ISPRS Journal of Photogrammetry and Remote Sensing, 66(3), 247-259. https://doi.org/10.1016/j.isprsjprs.2010.11.001

[27] Sheykhmousa, M. Mahdianpari, M. Ghanbari, H. Mohammadimanesh, F. Ghamisi, P. Homayouni, S. (2020). Support Vector Machine Versus Random Forest for Remote Sensing Image Classification: A Meta-Analysis and Systematic Review. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 13, 6308-6325. https://doi.org/10.1109/JSTARS.2020.3026724

[28] Breiman, L. (2001). Random Forests. Machine Learning 45, 5-32. https://doi.org/10.1023/A:1010933404324

[29] Sagi O. Rokach, L. (2018). Ensemble learning: A survey. WIREs Data Mining Knowl Discov. https://doi.org/10.1002/widm.1249

[30] skilltohire. (2020). A Beginner's Guide for Gradient Boosting. Medium website. https://medium.com/@skilltohire/the-beginners-guide-for-gradient-boosting-e5c67584240e

[31] Freund Y. Schapire, R. (1997). A desicion-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1), 119-139. https://doi.org/10.1006/jcss.1997.1504

[32] Breiman, L. (1996). Bagging predictors. Mach Learn 24, 123–140. https://doi.org/10.1007/BF00058655

[33] Ho, T. (1995). Random decision forests. Proceedings of the Third International Conference on Document Analysis and Recognition, 1, 278-282. https://doi.org/10.1109/ICDAR.1995.598994

[34] Ho, T. (1998). The random subspace method for constructing decision forests. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(8), 832-844. https://doi.org/10.1109/34.709601

[35] Amit, Y. Geman, D. (1997). Shape Quantization and Recognition with Randomized Trees. Neural Computation, 9, 1545-1588. Massachusetts Institute of Technology. https://doi.org/10.1162/neco.1997.9.7.1545

[36] Mantero, P. Moser, G. Serpico S. (2017). Partially Supervised classification of remote sensing images through SVM-based probability density estimation. IEEE Transactions on Geoscience and Remote Sensing, 43(3), 559-570. https://doi.org/10.1109/TGRS.2004.842022

[37] Fawagreh, K. Gaber, M. Elyan, E. (2013). Random forests: from early developments to recent advancements. Systems Science & Control Engineering, 2(1), 602-609. https://doi.org/10.1080/21642583.2014.956265

[38] Ham, J. Chen, Y. Crawford, M. Ghosh, J. (2005). Investigation of the random forest framework for classification of hyperspectral data. IEEE Transactions on Geoscience and Remote Sensing, 43(3), 492-501. https://doi.org/10.1109/TGRS.2004.842481

[39] Gislason, P. Benediktsson, J. Sveinsson, J. (2006). Random Forests for land cover classification. Pattern Recognition Letters, 27(4), 294-300. https://doi.org/10.1016/j.patrec.2005.08.011

[40] Rawat, W. Wang, Z. (2017). Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. Neural Computation, 29(9), 2352-2449. https://doi.org/10.1162/neco_a_00990

[41] Ajit, A. Acharya, K. Samanta, A. (2020). A Review of Convolutional Neural Networks. International Conference on Emerging Trends in Information Technology and Engineering, 1-5. https://doi.org/1109/ic-ETITE47903.2020.049

[42] Song, J. Gao, S. Zhu, Y. Ma, C. (2019). A survey of remote sensing image classification based on CNN. Big Earth Data, 3(3), 232-254. https://doi.org/10.1080/20964471.2019.1657720

[43] Kattenborn, T. Leitloff, J. Schiefer, F. Hinz, S. (2021). Review on Convolutional Neural Networks (CNN) in vegetation remote sensing. ISPRS Journal of Photogrammetry and Remote Sensing, 173, 24-29. https://doi.org/10.1016/j.isprsjprs.2020.12.010

[44] Zhu, X. Tuia, D. Mou, L. Xia, G. Zhang, L. Xu, F. Fraundorfer, F. (2017) Deep learning in remote sensing: a review. IEEE Geoscience and Remote Sensing Magazine, 5(4), 8-36. https://doi.org/10.48550/arXiv.1710.03959

[45] Hoeser, T. Kuenzer, C. (2020). Object detection and image segmentation with deep learning on earth observation data: a review-Part I: Evolution and recent trends. Remote Sens, 12(10), 1667. https://doi.org/10.3390/rs12101667

[46] Reichstein, M. Camps-Vails, G. Stevens, B. Jung, M. Denzler, J. Carvalhais, N. Prabhat. (2019). Deep learning and process under-standing for data-driven Earth system science. Nature, 566(7743), 195-204. https://doi.org/10.1038/s41586-019-0912-1

[47] Pi, Y., Nath D., Behzadan A. (2020). Convolutional neural networks for object detection in aerial imagery for disaster response and recovery. Advanced Engineering Informatics, 43. https://doi.org/10.1016/j.aei.2019.101009

[48] Benjdira, B. Khursheed, T. Koubaa, A. Ammar, A. Ouni, K. (2019). Car Detection using Unmanned Aerial Vehicles: Comparison between Faster R-CNN and YOLOv3. 1st International Conference on Unmanned Vehicle Systems-Oman (UVS), 1-6. https://doi.org/10.1109/UVS.2019.8658300

[49] Sacadura, J. (2021). Segmentação de imagens multiespectrais de alta resolução utilizando o modelo U-Net para cartografia de uso do solo. Tese para obtenção do grau de Mestre em Engenharia Geoespacial pela Faculdade de Ciências da Universidade de Lisboa. http://hdl.handle.net/10451/52008

[50] Shamsoshoara, A., Afghah, F., Razi, A., Zheng, L., Fulé, P., Blasch, E. (2020). Aerial Imagery Pile burn detection using Deep Learning: the FLAME dataset. Computer Networks, 193(4). https://doi.org/10.1016/j.comnet.2021.108001

[51] Abdollahi, A. Pradhan, B. Alamri, A. (2020). An ensemble architecture of deep convolutional Segnet and U-net networks for building semantic segmentation from high-resolution aerial images. Geocarto International, 37(12), 3355-3370. https://doi.org/10.1080/10106049.2020.1856199

[52] Hou, Y. Liu, Z. Zhang, T. Li, Y. (2021). C-U-net: Complement U-net for Remote Sensing Road Extraction. Sensors 2021, 21(6), 2153. https://doi.org/10.3390/s21062153

[53] bfortuner. (2019). Forwardpropagation. GitHub Repository. https://github.com/bfortuner/ml-glossary/blob/master/docs/forwardpropagation.rst

[54] DeepAI. (n.d). Weight (Artificial Neural Network). DeepAI website. https://deepai.org/machine-learning-glossary-and-terms/weight-artificial-neural-network

[55] GeeksforGeeks. (2018). Effect of Bias in Neural Network. GeeksforGeeks website. https://www.geeksforgeeks.org/effect-of-bias-in-neural-network/

[56] Bouvrie, J. (2006). Notes on Convolutional Neural Networks. https://web-archive.southampton.ac.uk/cogprints.org/5869/1/cnn_tutorial.pdf

[57] Zhang, A. Lipton, Z., Li, M. Smola, A. (2021). Dive Into Deep Learning. https://doi.org/10.48550/arXiv.2106.11342

[58] Wang, Y. Li, Y. Song, Y. Rong, X. (2020). The Influence of the Activation Function in a Convolution Neural Network Model of Facial Expression Recognition. Appl. Sci., 10, 1897. https://doi.org/10.3390/app10051897

[59] Glorot, X. Bordes A. Bengio Y. (2011). Deep Sparse Rectifier Neural Networks. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, 15, 315-323. https://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

[60] OpenGenus IQ. (n.d). Fully Connected Layer: The brute force layer of a Machine Learning model. OpenGenus IQ website. https://iq.opengenus.org/fully-connected-layer/

[61] Seb. (2021). An Introduction to Neural Network Loss Functions. Programmathically website. https://programmathically.com/an-introduction-to-neural-network-loss-functions/

[62] baeldung. (2022). Training and Validation Loss in Deep Learning. Baeldung website. https://www.baeldung.com/cs/training-validation-loss-deep-learning

[63] bfortuner (2022). Loss functions. GitHub Repository. https://github.com/bfortuner/ml-glossary/blob/master/docs/loss_functions.rst

[64] Brownlee, J. (2019). A Gentle Introduction to Cross-Entropy for Machine Learning. Machine Learning Mastery website. https://machinelearningmastery.com/cross-entropy-for-machine-learning/

[65] Koech, K. (2020). Cross-Entropy Loss Function. Towards Data Science website. https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

[66] McGonagle, J. Shalkouski, G. Williams, C. Hsu, A. Khlm, J. Miller, A. (n.d). Backpropagation. Brilliant website. https://brilliant.org/wiki/backpropagation/

[67] Srivastava, N. Hinton, G. Krizhevsky, A. Sutskever, I. Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15 (2014) 1929-1958. https://doi.org/10.5555/2627435.2670313

[68] Siddhartha, M. (2019). Regularization Techniques in Deep Learning. Kaggle website. https://www.kaggle.com/code/sid321axn/regularization-techniques-in-deep-learning/notebook

[69] Karagiannakos, S. (2021). Regularization techniques for training deep neural networks. AI SUMMER website. https://theaisummer.com/regularization/

[70] Long, J. Shelhamer, E. Darrell, T. (2014). Fully convolutional networks for semantic segmentation. https://doi.org/10.48550/arXiv.1411.4038

[71] Zhang, N. Donahue, J. Girshick, R. Darrell, T. (2014). Part-based R-CNN for fine-grained category detection. European Conference on Computer Vision (ECCV), 8689. https://doi.org/10.48550/arXiv.1407.3867

[72] Long, J. Zhang, N. Darrell, T. (2014). Do convnets learn correspondence? Advances in Neural Information Processing Systems, 2. https://doi.org/10.48550/arXiv.1411.1091

[73] Liu, X. Song, L. Lu, S. Xhang, Y. (2021). A Review of Deep-Learning-Based Medical Image Segmentation Methods. Sustainability, 13(3), 1224. https://doi.org/10.3390/su13031224

[74] Ronneberger, O. Fischer, P. Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. Medical Image Computing and Computer-Assisted Intervention, 234-241. https://doi.org/10.1007/978-3-319-24574-4_28

[75] Mishra, D. (2020). Transposed Convolution Demystified. Towards Data Science website. https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba

[76] LeCun, Y. Bottou, L. Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. https://doi.org/ 10.1109/5.726791

[77] LeCun, Y. (n.d). LeNet-5, convolutional neural networks. Yann LeCun website. https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba

[78] Saxena, S. (2021). The Architecture of Lenet-5. Analytics Vidhya website. https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/

[79] Chollet, F. (2017). Xception: Deep Learning With Depthwise Separable Convolutions. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1251-1258. https://doi.org/10.48550/arXiv.1610.02357

[80] Badrinarayanan, V. Kendall, A. Cipolla, R. (2016). SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(12), 2481-2495. https://doi.org/10.1109/TPAMI.2016.2644615

[81] Simonyan, K. Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. https://doi.org/10.48550/arXiv.1409.1556

[82] E. Maggiori, Y. Tarabalka, G. Charpiat. P. Alliez. (2017). Can Semantic Labeling Methods Generalize to Any City? 2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), 3226-3229. https://doi.org/10.1109/IGARSS.2017.8127684

[83] Inria Aerial Image Labeling Dataset. (2016) The dataset. Project Inria website. https://project.inria.fr/aerialimagelabeling/

[84] He, H. Jiang, Z. Tan, W. Cai, Y. Fatholahi, S. Gao, K. Xu, H. Hu, B. Qing, L. Li, J. (2021). Waterloo Building Dataset: A large-scale very-high-spatial-resolution image dataset for building rooftop extraction. NRC Research Press. https://www.nrcresearchpress.com/doi/abs/10.1139/geomat-2021-0006

[85] He, H. Jiang, Z. Gao, K. Narges F. Cai, Y. Tan, W. Hu, B. Qing, L. Xu, H. Li, J. (2021). Waterloo Building Dataset. Harvard Dataverse. https://doi.org/10.7910/DVN/EXRA2V

[86] Mnih, V. (2013). Machine Learning for Aerial Image Labeling. Tese para obtenção do grau de Doctor of Philosophy pela University of Toronto. https://www.cs.toronto.edu/~vmnih/docs/Mnih_Volodymyr_PhD_Thesis.pdf

[87] Ashwath, B. (2020). Massachusetts Buildings Dataset. Kaggle website. https://www.kaggle.com/datasets/balraj98/massachusetts-buildings-dataset

[88] Torch. (n.d). What is Torch? Torch website. http://torch.ch/

[89] Tensorflow. (n.d) Tensorflow Core. Tensorflow website. https://www.tensorflow.org/guide

[90] Jia, Y. Caffe. (n.d). Caffe website. https://caffe.berkeleyvision.org/

[91] Deeplearning4j. (2022). Deeplearning4j Suite Overview. Deeplearning4j website. https://deeplearning4j.konduit.ai/

[92] Nvidia Developer. (n.d). Deep Learning. Nvidia Develop website. https://developer.nvidia.com/deep-learning

[93] Goldsborough, P. (2016) A Tour of TensorFlow. https://doi.org/10.48550/arXiv.1610.01178

[94] Keras. About Keras. Keras website. https://keras.io/about/

[95] Direção Geral do Território. (2020). Normas e Especificações Técnicas para a Cartografia Topográfica Vetorial e de Imagem. CartTop-V1.1. https://www.dgterritorio.gov.pt/sites/default/files/ficheiros_cartografia/NormasEspecificacoesTecnicasCartTop.pdf