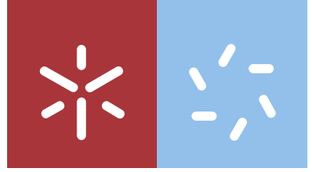
Extração de Informação de Vídeo e Técnicas de Redes Neuronais para Deteção de Objectos no Contexto da Monitorização de Espaços Industriais

Diogo Silva

UMinho | 2022



Universidade do Minho
Escola de Ciências

Diogo Vicente Magalhães Vilela Silva

Extração de Informação de Vídeo e Técnicas de Redes Neuronais para Deteção de Objectos no Contexto da Monitorização de Espaços Industriais

outubro de 2022

Universidade do Minho
Escola de Ciências

Diogo Vicente Magalhães Vilela Silva

Extração de Informação de Vídeo e Técnicas de Redes Neuronais para Deteção de Objectos no Contexto da Monitorização de Espaços Industriais

Dissertação de Mestrado
em Matemática e Computação
Área de Especialização em Computação

Trabalho efectuado sob a orientação de:
Professor Doutor Luís Lima Ferrás
Doutor Manuel João Ferreira

outubro de 2022

Agradecimentos

Em primeiro lugar gostaria de agradecer ao meu orientador Professor Doutor Luís Ferrás por todo o apoio, compreensão e disponibilidade demonstrados ao longo de vários meses. Agradeço também à Professora Doutora Maria Fernanda Costa por se juntar ao meu orientador e ter também contribuído com um sentido crítico e construtivo. Quero ainda agradecer ao Doutor Manuel João, o meu orientador da empresa Neadvance, a oportunidade de trabalhar e aprender com alguém tão experiente na área e ao Tiago, chefe de equipa da empresa Neadvance, por tudo o que me ensinou e ajudou.

Aos meus pais por me darem a oportunidade de prosseguir estudos e chegar até aqui, me apoiarem em todas as minhas decisões e por me ajudarem todos os dias a ser uma pessoa melhor. À minha namorada por me ter ajudado a superar as minhas dificuldades e me motivar a me esforçar o máximo na elaboração desta dissertação. À minha irmã que me ajudou a rever os dados das tabelas.

Por fim, agradeço a todas as pessoas que direta ou indiretamente contribuíram para a finalização deste trabalho mas não foram aqui mencionadas.

Declaração de Integridade

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração. Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Resumo

O alicerce de qualquer indústria é o chão de fábrica. O chão de fábrica é um pátio ou um armazém onde estão concentradas as linhas de produção compostas pelos funcionários e as máquinas necessárias para produção. É onde a força laboral é combinada com a tecnologia das máquinas para produzir os itens que consumimos e utilizamos no dia a dia.

Com a interdependência e variabilidade dos processos que ocorrem no chão de fábrica, surgem vários cenários e problemas, capazes de interromper todo o processamento de uma empresa. Assim, respeitar planos torna-se por vezes complicado, sendo que, quando o processamento de uma empresa é interrompido, é crucial uma resposta de gestão rápida e eficiente. Esta gestão do chão de fábrica, por norma, é feita por pessoas, tornando-a muito propensa ao erro humano, erro esse que é cada vez mais provável com o aumento do tamanho do espaço a ser vigiado (chão de fábrica).

Com a Indústria 4.0 e o surgimento de novos algoritmos focados na deteção e reconhecimento de objetos, tornou-se possível a aplicação deste tipo de modelos e algoritmos nas áreas da logística para melhorar a eficiência do espaço fabril. Assim, nesta dissertação, foram estudados e avaliados modelos para estimar a pose de indivíduos no chão de fábrica, com o objetivo de detetar possíveis erros na produção.

Começou por se definir o esqueleto que se queria prever, estipulando assim os *keypoints* desejados (locais do corpo que se pretende detetar). Foram avaliados 6 modelos diferentes de deteção e comparados através de 4 métricas. Foram ainda observadas as confianças de previsão destes modelos e o número de *keypoints* com uma certa confiança mínima. Os *datasets* utilizados foram criados de raiz e são compostos por imagens de colaboradores da empresa Neadvance a trabalhar na secretária e na montagem de peças numa porta.

Os resultados obtidos mostraram-se promissores, sendo possível determinar com alguma precisão a pose dos funcionários no chão de fábrica.

Palavras-chave: *Machine Learning*, *Deep Learning*, Visão por Computador, Inteligência Artificial, Estimação de Pose, Redes Neuronais Artificiais, Redes Neuronais Convolucionais.

Abstract

The foundation of any industry is the factory floor. The factory floor is a yard or warehouse where the production lines, consisting of the workers and the machinery necessary for production, are concentrated. This is where the labour force is combined with the machinery to produce the things we consume and use every day.

Due to the interdependence and variability of processes, there are multiple scenarios and potential problems that can bring all of a company's processes to a halt. As a result, adherence to plans occasionally becomes complicated and, in these cases, a quick and efficient management response is critical. Normally, this management is carried out by humans, which makes it highly susceptible to human error, which becomes more and more likely as the size of the space to be guarded increases.

With Industry 4.0 and the emergence of new algorithms focused on object detection and recognition, it became possible to apply this type of models and algorithms in the fields of logistics to improve the efficiency of the factory space. Thus, in this work, models were studied and evaluated to estimate the pose of people in the factory floor. First, the skeleton to be predicted was defined by determining the desired keypoints. Six different models were evaluated and compared using four metrics. The prediction confidence of these models and the number of keypoints with a certain minimum confidence were also taken into account. The datasets used were created from scratch and consist of images of Neadvance employees working at desks and assembling parts in a door.

The results obtained were promising, since it was possible to determine with some accuracy the pose of the workers in the factory floor.

Keywords: *Machine Learning*, *Deep Learning*, *Computer Vision*, *Artificial Intelligence*, *Pose Estimation*, *Artificial Neural Networks*, *Convolutional Neural Networks*.

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Introdução

O alicerce de qualquer indústria é o chão de fábrica. O chão de fábrica é um pátio ou um armazém onde estão concentradas as linhas de produção compostas pelos funcionários e as máquinas necessárias para produção. É onde a força laboral é combinada com a tecnologia das máquinas para produzir os itens que consumimos e utilizamos no dia a dia.

Nos processos do chão de fábrica, a interdependência e a variabilidade são aspetos cruciais. A interdependência entre as atividades refere-se à necessidade de existir uma certa ordem entre os processos de produção de um certo produto. Quanto à variabilidade, deve destacar-se a enorme variedade de problemas e fatores que não se conseguem prever, como por exemplo a qualidade de uma matéria-prima, a falta de um colaborador ou a avaria de uma máquina. Com estes problemas surge então a dificuldade de, por vezes, conseguir cumprir o planeamento estipulado e a obrigatoriedade de tomar decisões de gestão com eficiência e qualidade rapidamente.

No passado recente, toda a monitorização e controlo do chão de fábrica eram feitos manualmente e, por conseguinte, eram muito sensíveis ao erro humano. Quanto maior o tamanho do espaço industrial a ser vigiado, maior a probabilidade de erro dos colaboradores encarregados destas tarefas.

O surgimento da Indústria 4.0 permitiu que uma grande parte das empresas industriais sofresse uma profunda transformação digital, ajudando a corrigir este tipo de falhas nos sistemas de controlo e vigilância. A Indústria 4.0 pode ser descrita como a integração de várias tecnologias, como a Inteligência Artificial e a Internet das Coisas, e a representação de toda a automatização industrial. Os seus principais objetivos são os de promover a digitalização da indústria, através do aumento da produtividade, e digitalizar e integrar em ecossistemas digitais a mão humana. Segundo o conceito de Indústria 4.0, a interligação das máquinas, sistemas de produção e equipamentos, permitirá que as empresas concebam redes inteligentes ao longo de toda a cadeia de valor com o fim de verificar e dirigir os processos de produção de forma independente.

Resumidamente, o aparecimento da Indústria 4.0 permitiu o aumento da produtividade (isto é, com vários processos a serem automatizados, há uma maior flexibilidade e mais rigor, através da colocação dos colaboradores em pontos estratégicos que permitam uma melhoria nos resultados), ganho em eficiência (com um melhor aproveitamento dos recursos disponíveis e redução do número de erros durante os procedimentos), redução dos custos de produção (fornecendo mais autonomia às máquinas na concretização de

processos e/ou no planeamento da manutenção) e operações integradas (isto é, a valorização das fábricas inteligentes, onde é possível controlar as máquinas e equipamentos no momento e até remotamente. Desta forma, os dados são mais facilmente controlados, o que aumenta a transparência dos processos).

Aliado à Indústria 4.0, surgem novos algoritmos desenvolvidos com o intuito de detetar, reconhecer e seguir objetos em ambientes não controlados, como sistemas de vigilância em ambientes *indoor* e *outdoor*, controlo de tráfego e *smart parking*. Temos então agora a oportunidade de inserir este tipo de modelos/algoritmos nas áreas da logística com o fim de gerir de uma forma mais eficiente o espaço fabril.

Com estas soluções "inteligentes" de logística será possível reduzir o custo de operação e manutenção associado às instalações fabris. Para além disto, o investimento neste tipo de soluções possibilita a diminuição dos tempos de espera de produção e de paragem de máquinas, uma vez que reduz o número de intervenções de humanos em tarefas de logística.

1.1 Motivação e enquadramento

A motivação deste trabalho foi a de melhorar os processos de produção no chão de fábrica, tornando automáticos e independentes da intervenção humana todos os processos de verificação de erros. Desta forma será possível gerir de uma forma mais eficiente o espaço fabril.

1.2 Objetivos

O objectivo desta dissertação foi o estudo e desenvolvimento de soluções inteligentes baseadas em *Machine Learning* para estudar a pose dos funcionários em chão de fábrica.

1.3 Estado da arte

Como o tema principal desta dissertação é a estimação da pose dos funcionários no chão de fábrica, será agora apresentado o estado da arte sobre a Visão por Computador.

A Visão por Computador é uma área da Inteligência Artificial dedicada à obtenção de informação importante a partir de dados visuais como imagens ou vídeos. Exemplos deste tipo de informação são o

cálculo da geometria 3D de um objeto numa imagem, o *tracking* de pessoas ou objetos num vídeo, a deteção e identificação de faces ou de ações humanas, etc.

Embora a Visão por Computador tenha presenciado enormes avanços no passado recente, principalmente em 2012 quando a rede neuronal *AlexNet* ganhou o concurso de *software ImageNet Large Scale Visual Recognition Challenge*, esta não é uma ciência recente. Há mais de 60 anos que *computer scientists* investigam a melhor forma de extrair significado de imagens e vídeos, e, alguns dos trabalhos mais significativos serão agora apresentados.

Em 1959 foi desenvolvido um trabalho pioneiro por David Hubel e Torsten Wiesel, dois neurofisiologistas. David e Torsten publicaram um dos artigos mais importantes para a Visão por Computador, *Receptive fields of single neurons in the cat's striate cortex* [11], onde estudaram as propriedades dos campos recetivos dos neurónios do córtex visual de gatos através de experiências feitas tanto manualmente como por um sistema controlado por um computador utilizando linhas simples, linhas duplas e linhas múltiplas (grades). David e Torsten começaram por colocar elétrodos (polos) na área do córtex visual primário do cérebro de um gato anestesiado e observaram a atividade neuronal naquela região ao mostrarem ao animal várias imagens. No início não conseguiram obter qualquer resposta. No entanto, após alguns meses de pesquisa, repararam por acaso que, quando colocavam um novo *slide* no projetor, um neurónio era ativado. Após alguma confusão inicial, David e Torsten perceberam que o que excitava o neurónio era o movimento da linha criada pela sombra da borda afiada da lâmina de vidro do *slide*.

Com isto, estabeleceram que existem neurónios simples e complexos no córtex visual primário e que o processamento visual começa sempre com estruturas simples como as bordas de um objeto. Ora, isto é o conceito principal do *Deep Learning*, que começa por encontrar características "mais óbvias" para os humanos, e vai descobrindo mais particularidades conforme a complexidade do modelo.

Ainda no mesmo ano, surgiu o primeiro *scanner* de imagens digital. Russel Kirsch e alguns colegas construíram um aparelho capaz de transformar imagens em grades de números, valores binários que as máquinas conseguissem perceber. Uma das primeiras imagens a ser digitalizada foi uma foto do filho de Russel. Tinha apenas 5cm$\times 5$cm e 30,976 píxeis (um *array* 176×176).

A tese de doutoramento *Machine perception of three-dimensional solids* [12] de Lawrence Roberts, publicada em 1963, é considerada um dos precursores da Visão por Computador moderna. Na sua tese, Lawrence descreve o processo para derivar informação 3D de objetos sólidos a partir de fotografias

2D. No fundo, Lawrence reduziu o mundo visual a figuras geométricas simples.

Desenvolvido e descrito neste artigo [12], o programa escrito por Lawrence pretendia transformar imagens 2D em desenhos de linhas, depois, a partir destes desenhos construir representações 3D dessas linhas e, por fim, exibir estruturas 3D de objetos removendo as linhas ocultas.

Na década de 60, a Inteligência Artifical tornou-se uma disciplina académica e com isso surgiu um grande otimismo de alguns investigadores sobre o futuro da área, resultando na crença de que não demorariam mais de 25 anos a criar um computador tão inteligente como um humano. Foi também nesta altura que Seymour Papert, um professor do laboratório de Inteligência Artifical do MIT, decidiu criar o *Summer Vision Project* [13] para resolver, em poucos meses, o "problema de visão" das máquinas. Assim, um pequeno grupo de alunos do MIT coordenado por Seymour e Gerald Sussman, pretendia criar uma plataforma que conseguisse, automaticamente, aplicar *background/foreground segmentation* (segmentação em primeiro e segundo plano) e extrair objetos não sobrepostos de imagens do mundo real. No entanto, o projeto não foi um sucesso, e 50 anos mais tarde, ainda não se conseguiu resolver muitos dos problemas da Visão por Computador. Ainda assim, este evento ficou marcado na história como o nascimento da Visão por Computador como uma área científica independente.

Em 1982, surgiu um novo artigo muito influente para o campo da Visão por Computador, sendo este *Vision: A computational investigation into the human representation and processing of visual information* [14] de David Marr, um neurocientista britânico. Este trabalho tem como base as ideias de Hubel e Wiesel, e estabeleceu uma nova ideia muito importante: a de que a visão é hierárquica. Assim, David introduziu uma *framework* para a visão onde, algoritmos *low-level* que detetam características como bordas, curvas e cantos, são usados como indicadores para se entender, de uma forma mais geral ou *high-level*, dados visuais. Esta *framework* representacional incluía:

- Um esboço inicial de uma imagem, onde bordas, barras, limites, entre outras coisas, são representados (aqui uma clara referência ao artigo de Hubel e Weisel);
- Um esboço 2.5D para representar superfícies, informação sobre profundidade e descontinuidades de uma imagem;
- Um modelo 3D organizado hierarquicamente em termos de superfícies e primitivas volumétricas.

Nem o tipo de modelação matemática que poderia ser usado em sistemas de visão artificial, nem o tipo de processamento de aprendizagem estavam explicados no artigo e, como tal, apesar de inovador, o artigo era demasiado abstrato e *high-level*.

Por volta da mesma altura, um *computer scientist* japonês, Kunihiko Fukushima [15], também inspirado por Hubel e Wiesel, desenvolveu uma rede artificial auto-organizadora, com células simples e complexas, capaz de reconhecer padrões mesmo com alterações da posição. Esta rede, *Neocognitron*, incluía várias camadas convolucionais cujos campos recetivos tinham vetores de peso, mais conhecidos como filtros. Aos deslizar por *arrays* 2D, como os píxeis de uma imagem, estes filtros aplicam certos cálculos e, conforme o resultado, produzem eventos de ativação que seriam usados como entradas para as camadas subsequentes da rede. Assim, com toda esta estrutura, *Neocognitron* foi a primeira rede neuronal a merecer ser chamada de *deep*. É também considerada como o "avô" das redes convolucionais de hoje em dia.

Alguns anos mais tarde, mais concretamente em 1989, Yann LeCun, um cientista francês, decidiu aplicar um algoritmo de aprendizagem do tipo *backpropragation* à arquitetura da rede neuronal convolucional de Fukushima. Após trabalhar durante alguns anos neste projeto, lançou a rede *LeNet 5* [16] - a primeira rede convolucional moderna que introduziu alguns dos passos cruciais que ainda se usam nas redes neuronais convolucionais atualmente. Tal como Fukushima, LeCun decidiu aplicar a sua rede para reconhecer caracteres. Este trabalho acabaria por resultar na criação do dataset MNIST [17] de dígitos manuscritos, um dos mais famosos *datasets* de avaliação na área de *Machine Learning*.

Em 1997, um professor de Berkeley, Jitendra Malik, e o seu aluno, Jianbo Shi, lançaram um artigo sobre as suas tentativas para tratar o *perceptual grouping* [18] (agrupamento percetivo), isto é, tentaram que as máquinas conseguissem dividir as imagens em partes que fizessem sentido juntas (determinar automaticamente que píxeis numa imagem deviam estar juntos e distinguir objetos do meio onde estavam) usando um algoritmo de Teoria de Grafos. Não conseguiram obter resultados significativos e este tema continua sem uma solução eficaz até aos dias de hoje.

Já no fim dos anos 90, a área de Visão por Computador sofreu uma mudança de foco. Por volta de 1999, muitos investigadores pararam de tentar reconstruir objetos recriando-os em modelos 3D, como foi sugerido por Marr, e passaram a direcionar os seus esforços para *feature-based object recognition*. O artigo intitulado *Object Recognition from Local Scale-Invariant Features* [19] de David Lowe foi um

grande indicativo desta mudança. Neste artigo, é descrito um sistema de reconhecimento visual que utiliza características locais que não sofrem alterações com rotações, localização e, parcialmente, mudanças na iluminação. Ora, segundo Lowe, este procedimento é muito semelhante ao que ocorre nos neurónios do córtex temporal inferior dos primatas em situações de deteção de objetos.

Pouco tempo depois deste artigo ser publicado, em 2001, a primeira *framework* de deteção de faces em tempo real foi introduzida por Paul Viola e Michael Jones [20]. Embora não fosse baseada em *Deep Learning*, tinha certas influências deste conceito uma vez que, enquanto processava as imagens, aprendia que caraterísticas poderiam ajudar a localizar faces. Atualmente, o detetor de Viola e Jones ainda é utilizado em vários sistemas. É um classificador binário eficiente, construído a partir de vários classificadores mais fracos, com uma fase de treino/aprendizagem relativamente demorada e que utiliza o *Adaboost*, um algoritmo de *boost*, no treino destes classificadores mais fracos. De forma a encontrar uma face, o modelo começa por dividir uma imagem em vários retângulos que serão enviados para a cascata de classificadores mais fracos. Caso um destes retângulos consiga passar todas as fases da cascata, é classificado como positivo. Por outro lado, basta que falhe uma fase para que seja logo rejeitado. Este processo é repetido várias vezes e em várias escalas (retângulos de diferentes tamanhos). Uma câmara com a capacidade de reconhecer faces em tempo real foi lançada passados cinco anos pela Fujitsu que tinha como base o algoritmo de Viola e Jones.

Com os avanços na área da Visão por Computador, surgiu a necessidade (na comunidade) de um *dataset* de imagens que servisse de referência para avaliação dos modelos que iam sendo criados. Assim, em 2006, foi criado o projeto *Pascal VOC* [21]. Este projeto forneceu um conjunto de dados *standard* para classificação de objetos e um conjunto de ferramentas para aceder a este *dataset* e às suas anotações. Para além disto, os fundadores deste projeto decidiram também realizar um concurso anual, de 2006 a 2012, que permitiu avaliar a *performance* de diferentes métodos aplicados no reconhecimento de classes de objetos.

O *Deformable Part Model* [22] é um modelo *feature-based* desenvolvido por Pedro Felzenszwalb, David McAllester e Deva Ramanan em 2008, que, resumidamente, dividia objetos em várias partes e organizava-as em coleções, baseando-se em modelos pictóricos [23] introduzidos por Fischler e Elschlager nos anos 70, forçando um conjunto de restrições geométricas entre elas para depois modelar, assim, potenciais centros de objetos. Este modelo apresentou um ótimo desempenho na deteção de objetos onde

eram utilizadas *bounding boxes* para localizar objetos, superando vários métodos populares na altura, como por exemplo o *template matching*.

Um ano depois, começou a famosa competição *ImageNet Large Scale Visual Recongnition Competition (ILSVRC)* [24]. Seguindo os passos do *Pascal VOC*, ocorre anualmente e ainda inclui *workshops* depois da competição onde os participantes discutem novas ideias e aprendizagens que adquiriram com as entradas mais inovadoras. Enquanto o *dataset* do *Pascal VOC* só tinha vinte categorias de objetos, o *dataset ImageNet* possuía mais de um milhão de imagens, tratadas e processadas manualmente, e com mil classes de objetos. Desde o momento em que foi criada, esta competição tornou-se a referência na classificação de objetos em categorias e também na deteção de objetos. Em 2010 e 2011, a taxa de erro da competição era à volta dos 26% até que, em 2012, uma equipa da universidade de Toronto inscreveu um novo modelo, o *AlexNet* [25]. Era uma rede neuronal convolucional que viria a mudar tudo. Um modelo, semelhante na sua arquitetura ao *LeNet 5* de Yann LeCun, que conseguiu uma taxa de erro de 16.4%, sendo este um momento decisivo para as redes neuronais convolucionais. Nos anos seguintes, as taxas de erro na classificação de imagens no *ILSVRC* baixaram bastante e os vencedores foram sempre modelos com redes neuronais convolucionais.

Em 2014, uma equipa de investigadores da universidade de Montreal sugeriu que as máquinas poderiam aprender mais rapidamente ao serem compostas por duas redes neuronais em constante competição. A ideia é a de que uma das redes tente gerar dados falsos que pareçam o mais reais que consiga, e a outra rede tente identificar quais dos dados são falsos e, com o passar do tempo, ambas vão melhorar o seu desempenho. Estas redes chamam-se *Generative Adversarial Networks (GANs)* [26] e são consideradas um avanço significativo para a área.

Passado um ano, o *Facebook* revelou que o seu algoritmo *DeepFace* [27] é capaz de identificar uma pessoa 97.35% das vezes, colocando-o assim muito perto do nível de um Ser Humano.

Ainda em 2015, Leon Gatys, no seu artigo *A neural algorithm of artistic style* [28], introduziu um sistema artificial baseado em redes neuronais profundas (*deep neural networks*) que era capaz de criar imagens artísticas de grande qualidade. Este sistema utiliza representações neuronais para separar e recombinar o conteúdo e estilo de imagens arbitrárias.

Entre 2015 e 2017 surgiram os primeiros modelos capazes de aplicar segmentação semântica a *datasets* "do mundo real", tais como *FCN* em 2015, *DeepLab* em 2016 e *DeepLabv3* em 2017. Em

2017, Kaiming He, Georgia Gkioxari, Piotr Dollár e Ross Girshick publicaram um artigo onde apresentam o método *Mask R-CNN*. Enquanto as versões antigas do *R-CNN (Region based convolutional neural networks)* se focavam apenas em deteção de objetos, o *Mask R-CNN* adicionava segmentação de instância.

À medida que foram surgindo mais modelos e algoritmos na área de *Machine Learning* e Visão por Computador, surgiu também a Indústria 4.0.

Com isto, os ecossistemas industriais começaram a transformar-se em sistemas interligados e inteligentes, o que levou a um crescimento na procura por formas de automatizar o chão de fábrica em diferentes processos como Internet das coisas, *Cloud Manufacturing*, Sistemas Ciber-Físicos, entre outros [29, 30].

Em conjunto, estes paradigmas permitiram criar máquinas computorizadas que fornecem grandes quantidades de dados em tempo real e automaticamente. Estes dados são cruciais no processo de tomada de decisões e de gestão porque advêm de informações úteis para operações, manutenção e gestão para que se identifiquem e resolvam problemas que impactam a qualidade dos produtos e a produtividade, bem como a otimização dos processos.

1.3.1 A Visão por Computador e o Chão de Fábrica

A ideia principal deste projeto é a de conseguir monitorizar os movimentos dos funcionários em chão de fábrica e mais facilmente detetar que tipos de movimentos resultam numa peça/produto bem feita ou numa peça/produto com defeito. Esta dissertação prende-se, então, pelo uso de abordagens que resultam do estado da arte, para permitir monitorizar o chão de fábrica e o espaço industrial aberto. Para isto, tem-se por base os objetivos da transformação digital em ambiente industrial e da evolução da área da Indústria 4.0.

No sentido da digitalização, este processo tem vindo a permitir a automatização de tarefas outrora manuais e a otimização de processos. Contudo, esta adaptação em indústrias com sistemas *legacy* traz também vários desafios, pois estes sistemas, eletronicamente, fornecem poucos dados.

De forma a resolver estas questões, podem ser instalados sistemas embebidos, sensores ou outros equipamentos que vão permitir extrair mais informação. Esta abordagem é bastante lenta e envolve custos elevados e, mesmo assim, pode não se conseguir obter dados suficientes de boa qualidade [29]. O

que acontece é que, muitas das vezes, acaba por se deixar partes do chão de fábrica não monitorizado. Deshpande et al. [29] abordam as limitações da digitalização de uma forma não invasiva no controlo do ambiente do chão de fábrica. Para isso, apresentam uma *Computer Vision Toolkit*, juntando tecnologias de computação e de rede, de forma a integrá-las com plataformas de análise de *software*. Este *kit* incluía câmaras que controlavam remotamente objetos no chão de fábrica baseando-se em tecnologias de Visão por Computador.

No desenvolvimento do trabalho, foi feita uma integração de ferramentas que existiam previamente na empresa, tendo sido utilizado o *MTConnect*, um padrão de dados já conhecido que traduz entradas digitais em fluxos de dados. Esta proposta foi executada e demonstrada na Universidade de Cincinnati, num ambiente de simulação, demonstração esta que provou a sua capacidade para ser utilizada, bem como as potenciais vantagens do seu uso. Contudo, foram também identificadas falhas que não foi possível colmatar, tais como a falta de sensibilidade dos sistemas de Visão por Computador quando as condições de iluminação e os ângulos da câmara não eram os mais favoráveis.

Este tipo de sistemas é agora utilizado na indústria como forma de monitorizar o chão de fábrica e as áreas de logística, no interior e exterior dos edifícios. Uma vez que é necessário juntar a isto à mão humana na área da segurança para controlar os sistemas e corrigir possíveis erros, a comunidade científica tem trabalhado no sentido de encontrar formas de corrigir estas lacunas.

Kim et al. [31] como resposta fazem a sugestão de usar tecnologias de Inteligência Artificial para colmatar as necessidades de recursos humanos, substituindo-os por camadas de inteligência autónoma. Nesta proposta Kim et al. [31] testaram um sistema inteligente que recorre a algoritmos de pré-processamento e processamento de imagem, Visão por Computador e técnicas de *Deep Learning* de forma a seguir e identificar possíveis intrusos. Para além disto, propuseram o acrescento de redes neuronais convolucionais para classificar e detetar objetos. Para isto, foram testados e foi feita uma comparação de vários tipos de redes neuronais disponíveis com o proposto no projeto, tais como *AlexNet*, *GoogleNet* e *VggNet*.

Os diferentes métodos usados no chão de fábrica advêm de soluções que têm por base a análise de visão *stereo* instaladas no interior e exterior das fábricas. Por exemplo, métodos patenteados, como o de Jin et al. [32], fazem a descrição do conceito de análise de vídeos para segurança de instalações utilizando sensores sob a forma de câmaras. Estas câmaras, dispostas com orientação espacial fixa, formam um detetor *stereo* de distribuições 2D de intensidade de luz de uma zona de vigilância que, por sua vez,

formam as imagens detetadas. Neste caso, foi utilizada uma comparação de relevo 3D medindo cada elemento com o relevo já calculado com ausência de intruso e que foi memorizado.

Contudo, os autores Aravamuthan et al. [33] destacam as dificuldades relativas ao nível de sensibilidade destas soluções e também dos modelos propostos, problemas que se identificam pela existência de um grande número de falsos positivos/falsos alarmes. Os mesmos tentaram solucionar esta questão com o sistema *Physical Intrusion Detection System (PIDS)* que utiliza câmaras *stereo* e estimativas para demonstrar a utilidade do modelo proposto, modelo este capaz de obter informações sobre a posição e dimensão de objetos em situações de grande profundidade. Foi ainda capaz de identificar intrusos voadores como *drones*.

Tendo em conta os resultados observados nestes e noutros trabalhos, é possível concluir que este tipo de projetos é viável e que existe uma grande necessidade de aprofundar os avanços tecnológicos referentes à monitorização de chão de fábrica e espaços de logística abertos. A área da Inteligência Artificial tem sido um foco para empresas que têm intenções de evoluir e automatizar os seus processos, existindo até já exemplos de casos que provam a melhoria na logística das suas operações. Por exemplo, empresas como a *CommonSense Robotics*, *GreyOrange* e *XPO Logistics* começaram a disponibilizar produtos baseados em novos paradigmas tecnológicos para a automação inteligente [34].

Beneficiando do crescimento exponencial da área da Inteligência Artificial e aproveitando o seu potencial com *Machine Learning*, as empresas têm vindo a aplicar Inteligência Artificial em diversas áreas de negócio [35], obtendo resultados muito positivos, especialmente na indústria e logística com a automação de processos. Mais, conceitos como *Machine Learning* e Redes Neuronais serão um dos pontos chave nas aplicações da Indústria 4.0 e demonstram já uma grande margem de progressão em aplicações de Internet das Coisas.

Segundo a Deloitte, no seu estudo *"The warehouse of tomorrow, today"* [36], no futuro, toda a nova informação da cadeia de operações de armazéns, assim como a antiga, será utilizada por esta tecnologia para prever e sugerir tomadas de decisão, assim como para emitir alertas sobre ações a evitar. Todas estas decisões serão tomadas em tempo real e em prol da eficiência da otimização, organização, despacho e todas as outras operações em armazéns. Tudo isto será possível porque as tecnologias de *Machine Learning* têm vindo a solucionar certas dificuldades nas áreas de Visão por Computador e do Processamento de Linguagem Natural que, até agora, eram insuperáveis.

A solução proposta por Alias et al. [37] exemplifica a possível aplicação destas tecnologias na monitorização e controlo de logística combinando imagens extraídas de câmaras, *Machine Learning* e Realidade Aumentada. A deteção e *tracking* de objetos em vídeos são feitas através da extração de características como bordas, cores, movimentos e texturas, sendo o resultado útil para a gestão de uma fábrica. Neste caso, foram testados diferentes casos de uso:

- Caso de uso na Logística - Gestão de armazém: foi instalada a solução num armazém (na Alemanha) de distribuição de eletrónicos para comprovar a viabilidade de ter um sistema de gestão de armazéns baseado em câmaras. Foram monitorizados os movimentos das empilhadoras vazias ou carregadas para aferir o estado da carga dentro da área a monitorizar e registar esses estados. Além disso, foram também definidas zonas onde determinados produtos devem ser armazenados, havendo fluxos de processos específicos para diferentes zonas. Com estas informações, pretendia-se também fazer uma gestão de *stocks* mais fundamentada.

- Caso de uso na Produção - Controlar áreas de *buffer* e armazenamento: foi instalada a solução (na Alemanha) numa empresa de tubos e canos de plástico para detetar transportes vazios ou carregados entre a área de produção e a área de armazenamento, carregando a informação automaticamente para os registos da empresa, por exemplo, sobre produtos acabados recentemente ou quais a produzir a seguir. Isto permite disponibilizar dados úteis sobre tempo de progresso real no chão de fábrica.

Estes sistemas apresentaram, tanto ao nível da aplicabilidade como ao nível de algoritmos de *Machine Learning* utilizados, falta de otimização, mas, ainda assim, demonstraram que poderão vir a conduzir a grandes melhorias na gestão e monitorização do espaço fabril.

Até agora vimos a evolução do estado da arte em Visão por Computador e a sua aplicação na indústria. Porém, no caso concreto desta dissertação, pretendemos estudar a pose dos funcionários num chão de fábrica, sendo de real importância a deteção de objetos.

O reconhecimento de objetos e o mapeamento 2D/3D são regularmente utilizados com o objetivo de reconhecer vários objetos e/ou entidades a partir de dados de entrada como vídeos ou imagens e, para tal, são utilizados métodos como modelos 3D, identificação de componentes, deteção de limites e análise da

aparência a partir de diferentes ângulos. Esta área é aplicada, por exemplo em robótica, carros autónomos, no sistema da *Microsoft Kinect*, etc. [38].

Algumas das técnicas utilizadas para reconhecimento de objetos em Visão por Computador são:

- *Scale Invariant Feature Transform (SIFT)*;
- *Speeded Up Robust Feature (SURF)*;
- *Geometric Hashing*;
- *Histogram of Oriented Gradients (HOG)*.

Estas técnicas são muito eficientes, porém, com a gradual melhoria de desempenho das redes neuronais na deteção de objetos, a área de *Deep Learning* começou a ser adaptada para diferentes tarefas visuais, como deteção de objetos em imagens com mais de um objeto e com um fundo muito maior [39].

Por último, deve ainda ser realçado que, apesar de todos os estudos já feitos e das diversas técnicas já apresentadas, existem sempre problemas difíceis de contornar, como o facto de um objeto complexo (por exemplo, o corpo humano, que é o caso de estudo nesta dissertação) poder ser visto de diferentes ângulos e estar em diferentes posições, o que pode resultar em dificuldades acrescidas para o método de deteção, resultando em erros de previsão. Para além disto, as condições de luminosidade de um objeto têm uma grande influência no resultado final, principalmente em sistemas de reconhecimento baseado em cores. E ainda existem mais desafios, como a oclusão, escala, deformação/articulação, variação dentro da mesma classe e camuflagem com o *background* [40].

Outra dificuldade acrescida ao reconhecimento de objetos em tempo real é o reconhecimento de múltiplos objetos, que pode ainda ser mais problemático quando os objetos se tocam ou sobrepõem [40].

Com os estudos técnico-científicos que encontramos na literatura concluímos que estas soluções ainda têm que ser melhoradas uma vez que há problemas ainda sem solução ou a sua solução é muito dependente do tipo de dados usados. Exemplos de problemas a resolver: ruído nas imagens a analisar, objetos que se sobrepõem, saber a quantidade de dados necessários para um bom treino e *performance* do modelo, saber quais os algoritmos mais adequados conforme o problema em questão (no caso desta dissertação, são problemas que fazem uso de técnicas como processamento e análise de imagens e/ou vídeos e reconhecimento de objetos e comportamentos).

1.4 Organização da dissertação

No Capítulo 2 são apresentados conceitos básicos, mas essenciais, sobre *Machine Learning*, como redes neuronais artificiais e tipos de aprendizagem, e sobre Visão por Computador, como por exemplo um tipo específico de redes neuronais, as redes neuronais convolucionais.

O Capítulo 3 é dedicado à Estimação de Pose. São apresentadas e explicadas as diferentes metodologias existentes para estimar a pose de uma pessoa. São referidas algumas vantagens e desvantagens das diferentes metodologias e também alguns dos modelos com as abordagens mais importantes e marcantes para a área.

No Capítulo 4 é explicada a metodologia aplicada para detetar a pose de uma pessoa, sendo definidos os *keypoints* utilizados (punho, cotovelo, etc). É ainda apresentado o modelo escolhido para estimação de pose, e são feitos estudos quanto à sua robustez e eficácia. De notar que foram desenvolvidos de raiz os *datasets* de treino e teste que colocaram à prova os modelos em vários ambientes diferentes. Assim, para além do cenário de pessoas sentadas numa secretária, o modelo é testado no chão de fábrica usando como caso de estudo a montagem de peças numa porta

Por fim, no Capítulo 5 são apresentadas as conclusões e o trabalho futuro.

2 Conceitos básicos

Neste capítulo, serão apresentados alguns conceitos básicos necessários para facilitar a compreensão do trabalho desenvolvido. O principal foco será colocado nos conceitos de *Machine Learning* e Visão por Computador.

2.1 Machine Learning

Em 1959, Arthur Samuel, um *computer scientist* pioneiro no estudo da Inteligência Artificial, descreveu *Machine Learning* como "O estudo que permite que os computadores sejam capazes de aprender sem serem explicitamente programados". Com o avanço dos anos e da área, esta definição foi atualizada, e, hoje em dia, podemos descrever como uma disciplina da Inteligência Artificial desbloqueadora da aprendizagem autónoma das máquinas através de dados e experiências passadas com o intuito de identificar padrões e executar previsões com intervenção humana mínima [41–43]. Mais concretamente, aplicações de *Machine Learning* são algoritmos capazes de aprender, evoluir, desenvolver e adaptar ao serem confrontados com novos dados [44]. O aproveitamento de algoritmos identificadores de padrões em conjunto com um processo de aprendizagem iterativo, permite a estas aplicações obterem conclusões perspicazes sobre grandes *datasets* [42][43]. Estes algoritmos utilizam métodos de computação para aprender diretamente dos dados com que estão a trabalhar, algo não comum entre os algoritmos usuais, que depositam todas as suas expectativas numa equação pré-definida. Um dos fatores mais importantes para a obtenção de um desempenho ótimo destes algoritmos é a quantidade de dados disponíveis. Normalmente, a quantos mais dados o algoritmo tem acesso para analisar e aprender, maior é a probabilidade de se obter a eficácia desejada [45].

Regra geral, um problema a ser solucionado através de *Machine Learning* envolve um determinado *dataset*, cujo conteúdo varia mediante a temática do problema. Este *dataset* origina dois subconjuntos, os dados de treino e os dados de teste, onde o primeiro será utilizado para aprendizagem e treino do modelo e o segundo para avaliação da *performance* do modelo. A sua criação advém de uma separação direta do *dataset* inicial seguindo apenas uma referência de percentagens, isto é, por norma selecionam-se entre 70% e 90% dos dados para dados de treino (treinar o modelo), e a restante percentagem será usada para

dados de teste (avaliação da *performance* do modelo).

Depois de definidos os dois conjuntos, é selecionado um algoritmo de *Machine Learning*. Este algoritmo será treinado através dos dados de treino, isto é, ao analisar e inferir conclusões sobre esses dados, irá adaptar-se de maneira a obter as melhores previsões possíveis [41]. Deste processo resulta um modelo. Os dados de teste, dados que o modelo nunca viu, serão enviados para este modelo sobre os quais fará previsões. Estas previsões são avaliadas e, conforme o desempenho do modelo, decide-se se se deve voltar a treiná-lo ou passar à sua implementação num ambiente do mundo real.

Existem várias abordagens para a fase de treino de um algoritmo. A escolha de uma dessas abordagens dependerá de questões como a quantidade de dados, a sua qualidade, o tipo de tarefa, entre outros. Assim, podemos dividir os algoritmos de *Machine Learning* em quatro categorias [41][45]:

- *Supervised Learning*;
- *Unsupervised Learning*;
- *Semi-supervised Learning*;
- *Reinforcement Learning*.

2.1.1 Supervised Learning

O método *Supervised Learning* ensina as máquinas pelo exemplo. A sua estratégia utiliza dados já rotulados, dados para os quais já é sabida a sua classificação de acordo com a tarefa em questão, para treinar os algoritmos e, assim, criar inteligência artificial [41][45][43][44]. Os algoritmos são treinados até que sejam capazes de identificar objetivamente padrões subjacentes, relações entre os dados de treino e as classificações dos mesmos, e, eventualmente, estejam prontos para classificar precisamente novos dados que sejam apresentados [45].

A *Supervised Learning* é usualmente utilizada em problemas de classificação e regressão, os quais serão explicados de seguida [44].

Algoritmos de classificação: Um algoritmo de classificação tenta categorizar novos dados conforme um conjunto de opções de classes disponíveis. Um destes algoritmos pode ainda fazer parte de um grupo mais restrito, algoritmos de classificação binária, onde existem apenas duas opções. Alguns exemplos

deste tipo de algoritmos são a classificação de um *email* como *spam* ou não *spam* (caso restrito de classificações binárias), e a classificação de um número manuscrito, onde as opções vão desde o valor 0 ao 9 [45][44].

Algoritmos de Regressão: Neste tipo de algoritmos é esperado um resultado que represente uma relação numérica entre os dados de entrada e de saída. Alguns exemplos seriam a previsão do valor de uma casa baseada no seu código postal ou do valor que os clientes estariam dispostos a pagar por um produto tendo em conta a sua idade [45][44].

Resumindo, este método de aprendizagem tem como objetivo encontrar sentido num *dataset* tendo em conta o contexto de uma tarefa específica.

Como foi dito na secção 2.1, os algoritmos de *Machine Learning* têm como base o treino. Nos algoritmos de *Supervised Learning*, a fase de treino é muito semelhante à explicação genérica da secção 2.1 de como é feito o treino de um algoritmo. O sistema recebe um *dataset* classificado que o informa que tipo de dados de entrada estão relacionados com as possibilidades de resultados. O modelo treinado é depois confrontado com os dados de teste medindo a sua capacidade de efetuar previsões em dados nunca vistos antes pelo modelo.

A eficácia depende muito de dois fatores: a quantidade de dados rotulados disponíveis e o algoritmo que será utilizado. Para além disto, são enumeradas de seguida uma série de necessidades que podem também ter um impacto importante no resultado final [44]:

- Os dados de treino têm de ser tratados, limpos e equilibrados. Dados desnecessários e dados duplicados vão influenciar a capacidade do modelo de aprender e melhorar;
- A diversidade dos dados de treino será crucial para determinar o quão bem o modelo irá cumprir a tarefa quando confrontado com novos dados. Sem variedade de dados suficiente o modelo fracassará e as suas respostas não serão de confiança;
- Um valor de *loss* muito baixo não significa necessariamente um bom desempenho. Pode também significar que o modelo atingiu um estado de *overfitting*, onde o modelo se adaptou demasiado aos dados de treino e, por isso, quando tenta fazer previsões sobre dados novos, tem um desempenho muito abaixo do ambicionado. Logo, uma análise crítica dos resultados é também importante.

De entre os muitos algoritmos disponíveis na literatura sobre *Supervised Learning*, os mais usuais são: *Linear Regression*, *Logistic Regression*, *Neural Networks*, *Linear Discriminant Analysis*, *Decision Trees*, *similarity Learning*, *Bayesian Logic*, *support Vector Machines* e *Random Forests*.

2.1.2 Unsupervised Learning

Nos algoritmos de *Supervised Learning*, os dados de treino e teste utilizados são devidamente classificados. Neste tipo de aprendizagem (*Unsupervised Learning*) os algoritmos são alimentados com dados sem rótulo sendo os modelos concebidos para detetar padrões e similaridades nestes dados [41][42].

Os problemas de classificação e regressão não são apropriados para estes algoritmos. Por outro lado, comportam-se otimamente em tarefas de encontrar estruturas subentendidas nos dados, de acordo com uma medida de similaridade, e, posteriormente, na representação do *dataset* inicial numa forma comprimida com ajuda das estruturas encontradas [43][42].

De notar que este método de aprendizagem se assemelha bastante mais ao processo por que os humanos passam, isto é, aprender pela experiência e com dados que não foram classificados nem categorizados. Um exemplo da aplicação de *Unsupervised Learning* é por exemplo a diferenciação entre *emails spam* e não *spam*. Devido à grande variabilidade de características deste tipo de *emails*, até para humanos pode ser difícil atribuir alguma classificação.

Os algoritmos de *Unsupervised Learning* podem ser divididos em duas categorias: problemas de *clustering* e problemas de associação.

Problemas de Clustering: O *clustering* é um método de agrupamento de dados em subconjuntos. Para as decisões de colocação dos dados num grupo específico é utilizado um critério de medição da similaridade dos dados. Assim, dados com um grande nível de similaridade acabam no mesmo grupo e, simultaneamente, os dados de grupos diferentes possuem um nível de similaridade muito baixo ou nulo. A análise de *clusters* identifica as semelhanças entre os objetos e categoriza-os segundo a presença dessas semelhanças.

Problemas de Associação: Uma regra de associação é um método de *Unsupervised Learning*

que é utilizado para encontrar relações entre as variáveis de grandes bases de dados. Como resultado, determina o subconjunto destas relações que ocorrem em simultâneo. Um exemplo da aplicação deste método é em estratégias de *marketing* ao tentar perceber se uma pessoa que compra um objeto X é mais propensa a comprar o objeto Y.

Alguns dos algoritmos mais conhecidos de *Unsupervised Learning* são: *K-means clustering*, *KNN(k-nearest neighbors)*, *Hierarchical Clustering*, *Anomaly Detection*, *Neural Networks*, *Principal Component Analysis*, *Independent Component Analysis*, *Apriori Algorithm* e *Singular Value Decomposition*.

2.1.3 Semi-supervised Learning

Apesar dos bons resultados obtidos através das técnicas apresentadas nas subsecções 2.1.1 e 2.1.2, muitas vezes estas estratégias tornam-se impraticáveis para treinar um algoritmo. A rotulação de todo um *dataset* é um processo muito demorado e dispendioso e, no caso de se utilizar dados não rotulados, muitas vezes os modelos não atingem a eficácia desejada [46].

A *Semi-supervised learning* é uma categoria de técnicas de *Machine Learning* que tanto utilizam dados rotulados como não rotulados. Desta forma, e tal como se pode inferir pelo nome, é uma combinação entre *Supervised Learning* e *Unsupervised Learning* [41][46].

Os dados já rotulados são utilizados para treinar parcialmente um modelo e, com este treino, aplicar uma classificação aos dados restantes, processo que se denomina *pseudo-labelling* [46]. O modelo é depois treinado com a junção dos dados originalmente já rotulados e os dados que agora já possuem classificação.

A viabilidade deste método melhorou bastante com o aparecimento das *Generative Adversarial Networks (GANs)*, sistemas de *Machine Learning* que conseguem criar dados completamente novos a partir dos dados já rotulados do *dataset* original.

2.1.4 Reinforcement Learning

O *Reinforcement Learning* é um método baseado na recompensa de comportamentos desejados e/ou o castigo por comportamentos incorretos. O seu objetivo é aprender a ter o comportamento ótimo

para se obter a recompensa máxima possível. O sistema aprende a tomar as decisões corretas, a ter um bom comportamento, através de interações com o ambiente e observações de como este responde, algo semelhante ao que acontece com as crianças, quando começam a explorar todo o ambiente à sua volta para perceber que ações as ajudam a atingir o fim desejado [43][44][45].

No caso de não existir um supervisor, o sistema tem de descobrir sozinho quais os passos a tomar para obter a maior recompensa possível. Este processo é muito semelhante a uma experiência de tentativa-erro [43], onde a qualidade das decisões é medida não só por uma recompensa imediata, mas também por uma recompensa atrasada que é possível obter.

Os principais elementos de um sistema de *Reinforcement Learning* são o agente, o ambiente em que o agente está, as regras que o agente segue para tomar decisões e o sinal de recompensa que agente observa ao tomar uma ação [45].

Resumidamente, um problema de *Reinforcement Learning* envolve um agente explorador num ambiente desconhecido com um objetivo "em mente". *Reinforcement Learning* é baseado na hipótese de que todos os objetivos podem ser descritos pela maximização da recompensa acumulada esperada. Para tal, este agente tem de aprender a perceber e manipular o estado do ambiente tomando ações para atingir a recompensa máxima.

2.2 Redes neuronais artificiais

A *Deep Learning* é uma subárea de *Machine Learning* focada na produção de modelos baseados em redes neuronais de várias camadas. Aliás, *deep*, que em português significa profundo, tem precisamente a ver com o facto destes modelos possuírem várias camadas de neurónios [45].

As redes neuronais, também conhecidas como redes neuronais artificiais, são a base dos algoritmos de *Deep Learning* [41]. O seu nome e estrutura são inspirados no cérebro humano e tentam copiar a forma como os neurónios humanos comunicam entre si [41][42][47].

2.2.1 Neurónio humano

Na biologia, os neurónios transmitem informação entre si através de potenciais de ação, isto é, impulsos nervosos em que há alteração do potencial elétrico das membranas das células. Isto acontece através da

conexão entre o axónio de um neurónio e a dendrite do neurónio seguinte. A este processo dá-se o nome de sinapse. Assim, a informação começa na dendrite, passa para o corpo da célula, de seguida para os axónios e, por fim, para a sinapse para enviar a informação para o próximo neurónio [48]. É possível observar a estrutura descrita na figura 1.

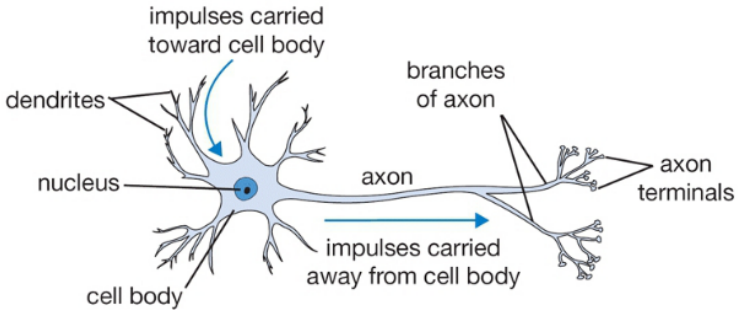


Figura 1: Representação de um neurónio biológico (adaptada de [1]).

2.2.2 Neurónio artificial

Tendo em conta a complexidade de um neurónio humano, à qual pode ser acrescentada uma série de elementos bioquímicos dificilmente simuláveis que ainda aumentam mais essa complexidade, informáticos e matemáticos chegaram a um modelo minimalista do que poderia ser um simulador destes neurónios [42]. Neste modelo, houve o cuidado de conseguir uma equivalência de quase todos os conceitos apresentados na subsecção 2.2.1, algo que se pode observar na figura 2.

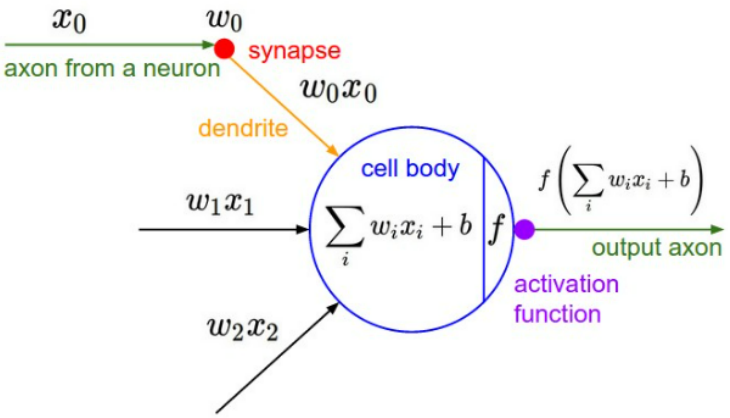


Figura 2: Representação de um neurónio artificial (adaptada de [1]).

Na figura 2 está representada, não só a estrutura do neurónio artificial, como também o processo do tratamento dos dados enviados para este neurónio. O neurónio começa por receber informações através de várias sinapses, informações estas representada por

$$x_i, \quad i = 0, 1, 2, ... \tag{1}$$

Os dados/informações serão combinados tendo ainda em conta que as conexões entre neurónios não são sempre iguais (podem ter diferentes pesos) e, por isso, esta variação da força entre neurónios vai influenciar a sua comunicação. A força de conexão está representada por

$$w_i, \quad i = 0, 1, 2, ... \tag{2}$$

sendo estes valores conhecidos como os pesos da rede.

Os dados recebidos pelo neurónio serão os produtos

$$w_i x_i, \quad i = 0, 1, 2, ... \tag{3}$$

As contribuições de todas as sinapses são depois somadas, ainda com o acrescento de um certo valor de *bias*, b, formando o somatório

$$\sum_i w_i x_i + b, \quad i = 0, 1, 2, ... \tag{4}$$

O valor obtido pela equação 4 é depois passado a uma função de ativação não linear $f(.)$ que produz o valor de

$$f(\sum_i w_i x_i + b). \tag{5}$$

Este resultado representa a ativação do neurónio e será enviado para os neurónios que estejam conectados aos seus axónios (ver figura 2).

2.2.3 Funções de ativação

No procedimento explicado na subsecção anterior (2.2.2), o resultado de $\sum_i w_i x_i + b$ é depois enviado para uma função de ativação $f(.)$ não-linear. A necessidade deste passo advém dos resultados dos neurónios biológicos não serem de todo uma combinação linear dos dados recebidos, algo que iria comprometer a tentativa de simulação de um neurónio humano caso não fosse aplicado. É de notar que, caso não fosse aplicada nenhum tipo de não-linearidade, então o neurónio apenas aprendia uma relação entre o valor de entrada e o valor de saída, visto que a composição de funções lineares é também uma função linear. Assim, funções de ativação não-lineares são cruciais na estrutura dos neurónios artificiais já que, só assim, serão capazes de desempenhar tarefas não-lineares e mais complexas.

Existem várias funções de ativação na literatura. De todas, as mais comuns são ReLU, Sigmoid e Tanh que serão definidas de seguida.

ReLU - Rectified Linear Unit

Esta função retorna o valor 0 quando recebe valores negativos, e o próprio valor quando este é maior ou igual a 0 [49]:

$$f(x) = x^+ = \max(0, x), \tag{6}$$

onde x é o argumento recebido pela função.

A maioria dos programadores de *Machine Learning* utiliza esta função nas camadas intermédias das redes neuronais uma vez que, devido à sua simples teoria matemática, as redes com este tipo de unidades são mais facilmente otimizadas [49] do que com outras funções como a Sigmoid, que será apresentada posteriormente. A figura 3 ilustra o comportamento desta função.

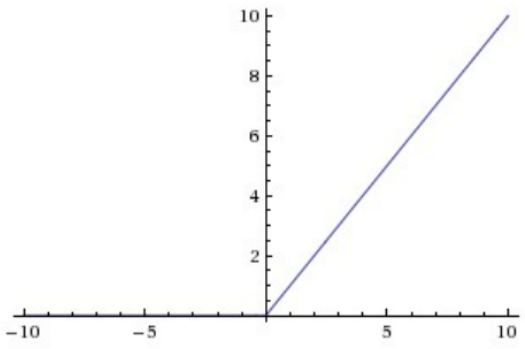


Figura 3: Representação da função de ativação ReLU (adaptada de [1]).

Sigmoid

Como se pode ver na figura 4, a função Sigmoid recebe valores reais e transforma-os em valores entre 0 e 1. Existem diferentes funções Sigmoids, mas, uma das mais usadas é a função logística:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{7}$$

onde x é o argumento recebido pela função.

É possível verificar que o crescimento da função é muito acentuado quando os valores de x (eixo horizontal) variam entre -2 e 2. Assim, existe uma mudança significativa em S para qualquer mudança mínima no X, dentro destes valores. Este fator faz com que a função tenha tendência para atribuir valores extremos, e, assim, leva o classificador a fazer distinções de previsões claras [49].

Uma das suas vantagens é a primeira observação aqui feita, a da limitação dos valores de S entre 0 e 1, que torna fácil o controlo da excitação dos neurónios, evitando assim explosões nos valores dos resultados [49].

No entanto, um dos problemas é a fraca resposta a valores de X que se situem na região perto onde se atinge no limite o valor $S = 1$ ou $S = 0$. Isto implica um valor de gradiente baixo (derivada da função) que leva a problemas de desaparecimento do gradiente que, por sua vez, faz com que a rede aprenda extremamente devagar ou que até nem consiga continuar a evoluir/aprender [49].

O seu gráfico pode ser observado na figura 4.

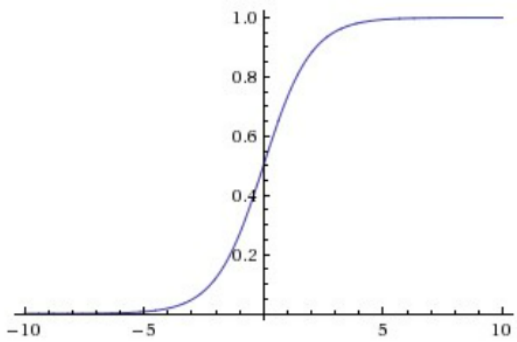


Figura 4: Representação da função de ativação Sigmoid (adaptada de [1]).

Tanh

Uma variação da Sigmoid é a função Tanh (figura 5) definida por:

$$T(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{8}$$

onde x é o argumento recebido pela função.

Similarmente à Sigmoid, os resultados de Tanh estão também limitados, mas, neste caso, por -1 e 1 em vez de 0 e 1.

O gradiente de Tanh é quatro vezes maior que o gradiente da função Sigmoid. Isso significa que o uso da função de ativação Tanh resulta em maiores valores de gradiente durante o treino e maiores atualizações nos pesos da rede. Então, se queremos gradientes fortes e grandes saltos, devemos usar a função de ativação Tanh.

Outra diferença é que o *output* da Tanh é simétrico em torno de zero, levando a uma convergência mais rápida.

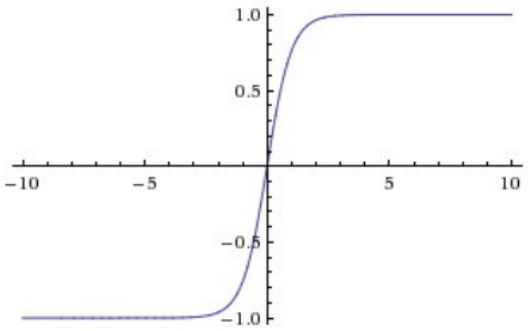


Figura 5: Representação da função de ativação Tanh (adaptada de [1]).

2.2.4 Arquitetura

Na subsecção 2.2.2, foi descrita a organização de um neurónio artificial. Porém, apenas um neurónio não tem a capacidade de desempenhar grandes tarefas nem obter soluções muito complexas. Grandes conjuntos destes neurónios são capazes de feitos incríveis. Assim, estes neurónios serão as unidades trabalhadoras de toda a estrutura das redes neuronais artificiais, a chamada arquitetura, que descreve os neurónios e a sua conectividade [44][41]. No cérebro humano, os muitos neurónios estão conectados a outros milhares, algo ainda não possível de fazer computacionalmente, e, por isso, estas arquiteturas artificiais ainda não são tão bem conectadas (em termos quantitativos) como o cérebro. Na figura 6, está representado um diagrama genérico de como está organizada uma rede neuronal artificial.

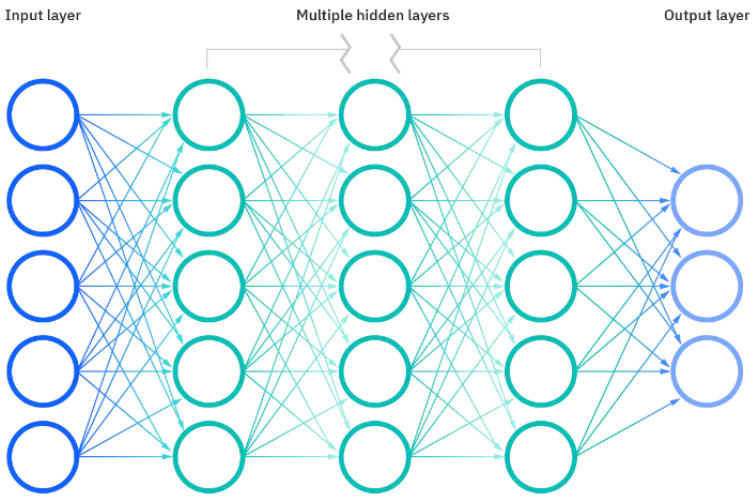


Figura 6: Arquitetura de uma rede neuronal artificial (adaptada de [2])

Cada um dos nodos representa um neurónio artificial e, de acordo com a figura 2, a ligação à sua esquerda, caso exista, ilustra o processo sinapse com um peso atribuído, e a ligação à direita representa o axónio dos dados de saída.

Uma sequência de neurónios na vertical forma uma camada e, aproveitando a distinção de cores, existem 3 tipos de camadas: *input layer* (azul escuro), *hidden layer* (verde) e a *output layer* (azul claro) [50].

A *input layer* é composta por tantos neurónios quanto o número de características que o modelo irá utilizar para fazer as suas previsões. Caso sejam dados tabulados, por exemplo a idade, peso e altura duma pessoa, o número de neurónios utilizado seria um para cada uma destas características, ou seja, 3. Se os dados em questão forem dados visuais, como imagens, então seria optado por um neurónio responsável por cada píxel da imagem.

Na outra extremidade do diagrama está a *output layer*. Esta camada é responsável pelo tratamento final dos dados para serem conhecidos os resultados da rede. Assim, o número de neurónios será representante da quantidade de previsões que sejam desejadas. Mais concretamente, em problemas de regressão, como por exemplo descobrir o valor de uma casa, um neurónio seria o suficiente, mas, caso a tarefa fosse descobrir as posições dos cantos de um quadrado para limitar a posição de um objeto, então já seriam necessários pelo menos 2 neurónios. Noutro tipo de problemas, como problemas de classificação, usualmente é atribuído um neurónio a cada classe disponível.

Por fim, as camadas representadas a verde são *hidden layers*, que se situam entre a *input layer* e a *output layer*. O tipo de problema a ser tratado será um fator fulcral para determinar a quantidade destas camadas necessárias na rede. Uma rede com uma quantidade de *hidden layers* reduzida pode não ser capaz de obter informação suficiente para a resolução da tarefa, assim como demasiadas destas camadas podem acabar por se tornar prejudiciais para o seu desempenho ao se tornar muito moldada aos dados (*overfitting*). Por norma, 1 a 5 camadas são suficientes para a maioria dos problemas. Ainda assim, quando se trata de trabalhar com imagens ou dados de voz, são necessárias centenas de camadas e, por isso, recorre-se a modelos pré-treinados, como *YOLO*, *ResNet* e *VGG*, que permitem utilizar parte das suas arquiteturas já pré-treinadas e ainda adicionar novas camadas se forem necessárias.

Para alguns *datasets*, ter uma primeira camada com muitos neurónios seguida de camadas mais pequenas leva a um melhor desempenho do modelo, uma vez que a primeira camada é capaz de apren-

der características *low-level* que serão úteis para as próximas camadas aprenderem atributos de uma ordem superior. Segue um exemplo do tipo de características que podem ser retiradas/aprendidas de uma imagem ao longo de uma rede (figura 7).

Devemos notar que este processo é complexo e que, na realidade, a rede não funciona exatamente como o nosso cérebro. O programador terá sempre que fazer testes com diferentes números de camadas e diferentes arquiteturas, mediante o problema a resolver.

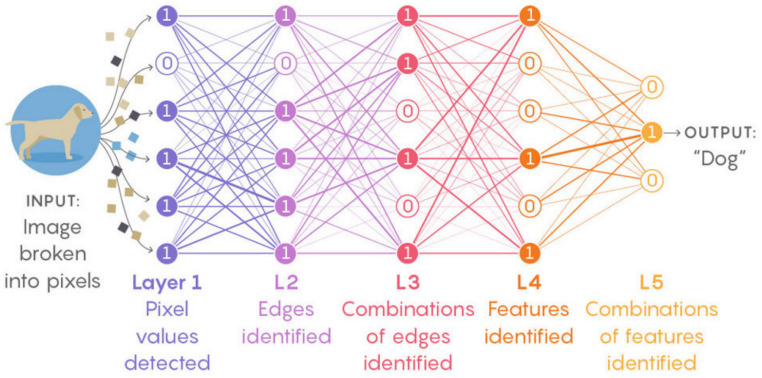


Figura 7: Exemplo do funcionamento de uma rede neuronal artificial (adaptada de [3]).

2.2.5 Parâmetros

Os parâmetros de uma rede são os valores que podem ser afinados para treinar a rede. Estes parâmetros incluem os pesos, anteriormente referidos como a força de conexão entre os neurónios, e o *bias*. Assim, o número de parâmetros da *input layer* para a primeira *hidden layer* seria $n^{\circ} de\ neurónios\ de\ entrada \times n^{\circ} de\ neurónios\ na\ hidden\ layer + n^{\circ} de\ biases$. Para saber o total, seria efetuado o cálculo entre todas as camadas.

Por norma, quanto maior a rede neuronal maior a sua capacidade e, por conseguinte, problemas cada vez mais complexos conseguem ser resolvidos. No entanto, esta maior capacidade de resolução necessita também de mais recursos para o treino, assim como mais memória para armazenar os pesos, mais tempo de espera durante o treino e mais poder de processamento para atualizar os pesos durante o treino.

2.2.6 Treino

O treino de uma rede neuronal não é nada mais nada menos do que o ajuste dos seus parâmetros, isto é, os seus pesos e *bias*. Para tal, um *dataset* (um conjunto de dados, por exemplo, imagens, sons, etc.) será observado e analisado, repetidamente, até que o modelo já tenha aprendido e se tenha adaptado aos dados em questão. Esta adaptação do modelo é a constante atualização dos parâmetros [47]. A passagem/análise completa de todos os dados deste conjunto denomina-se época [47], e os modelos são usualmente treinados durante milhares de épocas para conseguirem uma adaptação ótima. Para além do número de vezes que os dados serão revistos, é também definida a quantidade de dados que serão observados pelo modelo de cada vez, o *batch size*. Assim, estes grandes *datasets*, por vezes com milhões de exemplos, serão divididos em blocos que serão enviados à vez para a rede neuronal.

Para medir o quão incerta é uma previsão, é utilizada uma *loss function* $L(real, previsão)$. A comparação dos valores previstos pelo modelo com as classificações reais dos dados irá determinar o estado do treino do modelo. Um valor de *loss* muito alto significa que as previsões da rede estão a diferenciar-se muito dos valores reais e, como seria de esperar, caso este valor seja baixo, os valores previstos assemelham-se aos valores reais.

A *loss function* transforma o problema de encontrar os melhores pesos num problema de otimização [44]. Este problema será encontrar os pesos que minimizam esta *loss function* (ou o valor médio da *loss function* ao longo de um *dataset* de treino). A cada iteração, cada vez fica mais próxima a solução deste problema de otimização: $min_{pesos} L(real, previsão, pesos)$

As *loss functions* mais conhecidas são: *MAE*, *MSE* e *RMSE*.

A função *MAE (Mean Absolute Error)* representa a média da diferença absoluta entre o valor real e o previsto. Mede a média dos resíduos no *dataset* e define-se por [51]:

$$\frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{9}$$

onde y_i é o valor real e $\hat{y}_i$ é o valor previsto .

A função *MSE (Mean Squared Error)* representa a média do quadrado das diferenças entre o valor real e o valor previsto. Define-se por [51]:

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{10}$$

onde y_i é o valor real e $\hat{y}_i$ é o valor previsto.

A *RMSE (Root Mean Squared Error)* é a raiz da *MSE* definindo-se por [51]:

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{11}$$

onde y_i é o valor real e $\hat{y}_i$ é o valor previsto.

Os últimos conceitos necessários para a fase de treino são o otimizador e a taxa de aprendizagem. Um otimizador determina, baseando-se na *loss function*, como e quanto cada parâmetro deve ser alterado [52]. Resolve o problema de atribuição de culpa, isto é, a que parâmetros atribuir culpa quando a rede não está a ter um bom desempenho. A solução para este problema é a aplicação do método do gradiente, que utiliza o gradiente da *loss function* para executar alterações em cada parâmetro. A taxa de aprendizagem define o quão rapidamente a rede neuronal deve atualizar os seus pesos conforme o que já aprendeu. Valores inferiores desta taxa necessitam de mais épocas de treino tendo em conta as pequenas mudanças que são feitas aos pesos de cada vez, enquanto que taxas muito altas resultam em grandes mudanças e, portanto, não são precisas tantas épocas. Uma taxa de aprendizagem muito alta pode fazer com que o modelo convirja demasiado rápido para uma solução não ótima e uma taxa muito baixa pode interromper o processo [52].

Agora que já vimos os conceitos mais básicos de *Machine Learning*, vamos então estudar a Visão por Computador.

2.3 Visão por Computador

Enquanto humanos, passamos grande parte da nossa vida a observar o ambiente no qual estamos inseridos. Percebendo o seu contexto conseguimos diferenciar objetos, estimar a que distância estão de nós e entre si, calcular a que velocidade se movem, etc., sendo estes apenas alguns exemplos do que os humanos são capazes de fazer com as suas habilidades visuais. Da mesma forma, a Visão por Computador permite que sistemas baseados em Inteligência Artificial treinem e aprendam a executar estas mesmas ações, combinando câmaras, algoritmos e dados [53][54].

Uma grande vantagem destes sistemas é que nunca se cansam, enquanto que, para os humanos, o mesmo não pode ser dito. Tendo isso em conta, é possível treinar máquinas alimentadas pela Visão por

Computador para analisarem milhares de imagens e produtos em minutos. Ainda assim, a necessidade de bases de dados enormes para obter resultados significativos, algo nem sempre fácil de adquirir, é um dos desafios desta área. Tal como nos algoritmos de *Machine Learning*, no seu processo de treino e aprendizagem (dos algoritmos alimentados por Visão por Computador), estes dados são analisados repetidamente até que o sistema tenha retirado todas as informações possíveis e necessárias para desempenhar a tarefa em questão.

As máquinas interpretam as imagens num formato bastante simples: uma série de píxeis, cada um com o seu próprio conjunto de valores de cores [54]. Consideremos a figura 8 numa escala de cinzentos e a progressiva simplificação para uma matriz de inteiros:

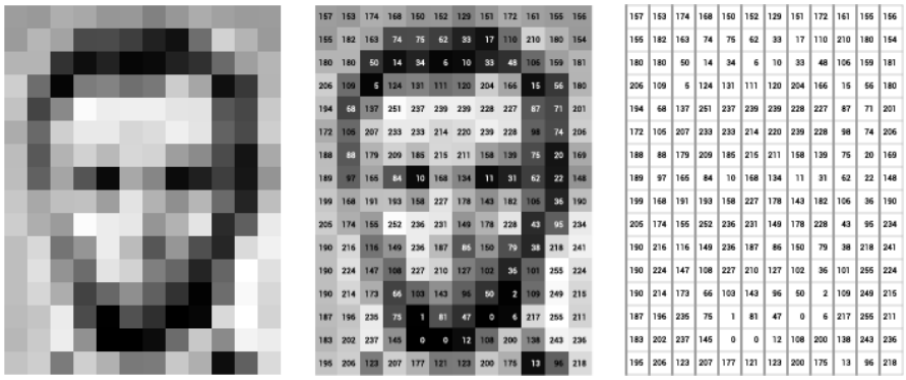


Figura 8: Conversão de uma imagem em escala de cinzentos na sua representação matricial (adaptada de [4]).

Como se pode concluir pela imagem, neste caso, cada píxel é representado por um valor entre 0 e 255. A matriz com apenas valores é o que a máquina "vê" quando recebe uma imagem como *input*. Uma vez que esta imagem tem 12 colunas e 16 linhas, um sistema que recebesse esta imagem teria $12 \times 16 = 192$ valores de *input*.

Ao trabalhar com imagens de cor a representação de uma imagem fica mais complexa. Por norma, é utilizado o formato *RGB* com uma gama de valores entre 0 e 255. Neste caso, cada píxel passará a ter 3 valores associados, um representante de cada cor (R - *Red*, G - *Green*, B - *Blue*). Ou seja, temos um valor para vermelho, um valor para verde e um valor para azul.

Um dos problemas de trabalhar com imagens é o seu grande custo computacional. O valor de cada

canal é guardado em 8 *bits*, ou seja, ao trabalhar com uma imagem a cores cada píxel necessita de 24 *bits* (8 *bits* para o R, 8 *bits* para o G e 8 *bits* para o B). Supondo que é utilizada uma imagem de tamanho 1024×768 píxeis, então seriam necessários 19 milhões de *bits*, aproximadamente 2.36 *megabytes*, apenas para uma imagem. Por norma, e como foi dito anteriormente, são necessários milhares de dados (imagens neste caso) para obter bons resultados ao treinar os modelos. Este foi um dos motivos que levou à demora na evolução deste campo científico.

As redes neuronais convolucionais, um tipo de redes neuronais que será explicado mais detalhadamente na secção 2.4, foram cruciais para o desenvolvimento e avanço da área Visão por Computador. Este tipo de redes utiliza o mesmo conceito das redes neuronais, mas, acrescenta alguns passos antes do processo usual. Estes passos são focados na extração de características e na descoberta da melhor representação possível da imagem a ser tratada, o que levará ao melhor nível de entendimento dos dados pelo modelo. Este tratamento, feito antes do processo usual de uma rede neuronal, alivia o poder computacional necessário para treinar o resto da rede, ajudando assim a ultrapassar o obstáculo explicado no parágrafo anterior.

Entre todas as tarefas que podem ser solucionadas com a Visão por Computador as mais comuns são:

- Classificação de objetos: O sistema recebe uma imagem ou *frame* de um vídeo e classifica o conteúdo desses dados de entrada de acordo com uma série de opções;
- Identificação de objetos: Ao receber uma imagem ou *frame* de um vídeo o sistema identifica onde está o objeto na imagem, isto é, por exemplo, numa imagem com vários cães, identificar um cão em específico;
- *Tracking* de objetos: Os dados de entrada são um vídeo e o sistema encontra o objeto desejado e segue-o ao longo de todo o seu movimento.

2.4 Redes neuronais convolucionais

Como já foi referido na secção anterior 2.3, as redes neuronais convolucionais (RNC) revolucionaram o campo da Visão por Computador pela sua capacidade de extrair aspetos essenciais de imagens e vídeos (o que reduziu a necessidade computacional) e, por conseguinte, foram o principal meio utilizado na análise

de imagens e no seu processamento. Assim como as redes neuronais artificiais, definidas e explicadas na secção 2.2, a arquitetura das RNC é também semelhante ao padrão de conectividade dos neurónios do cérebro humano e foi ainda inspirada na organização do córtex visual humano, onde cada um destes neurónios reage a estímulos de regiões restritas do campo visual, intituladas de campos recetivos, um conjunto de sobreposições dos quais forma toda a área visual [55].

A vantagem computacional das RNC [45] no tratamento de dados visuais deve-se à sua abordagem: a redução de dimensão de uma imagem através do uso de filtros. Esta redução é aplicada metodicamente, de maneira que não sejam desperdiçadas as informações essenciais da imagem. Assim, o objetivo de uma RNC é reduzir as imagens a uma forma que seja mais facilmente processada e sem perda de nenhuma característica que seja crucial para a previsão final.

Uma RNC é então um algortimo de *Deep Learning* capaz de receber dados visuais, como imagens e vídeos ou partes de vídeos, e atribuir importância e sentido a diferentes blocos da imagem assim como diferenciá-los [45]. Para além disto consegue também capturar dependências espaciais e temporais na imagem devido à utilização de filtros [55]. O pré-processamento que é necessário aplicar nestas redes é muito inferior ao de outros tipos de algoritmos de classificação. Algo importante a acrescentar é que os filtros já referidos que são aplicados às imagens, e que receberão mais atenção posteriormente, começaram por ser criados manualmente, mas, com a evolução da área, tornou-se possível que, à custa de treino, as RNC aprendessem a desenvolver estes filtros.

2.4.1 Dados de entrada

Anteriormente, na secção 2.3, foi explicado e exibido o formato de uma imagem em escala de cinzentos. No entanto, na grande maioria dos casos de estudo reais, as imagens são a cores e por isso é necessária uma explicação mais pormenorizada da estrutura deste tipo de imagens. Para tal, observa-se na figura 9 um esquema do que seria uma imagem a cores, neste caso utilizando o formato *RGB*:

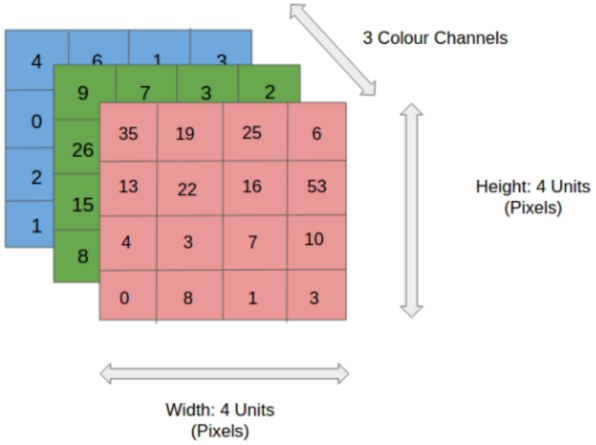


Figura 9: Canais de uma imagem *RGB* (adaptada de [5]).

O formato *RGB* representa os 3 canais de cores: vermelho, verde e azul. Na figura 9 está representada uma imagem em formato *RGB* dividida por 3 matrizes, cada uma representante de um canal de cor [54]. Cada píxel terá então 3 valores atribuídos. É importante acrescentar que, para além deste, existem ainda muitos mais formatos de canais como *Grayscale*, *BGR (Blue, Green, Red)* e *HSV (hue, saturation, value)*.

2.4.2 Arquitetura de uma rede neuronal convolucional

As RNC são tipicamente compostas por três tipos de camadas: *convolutional layer*, *pooling layer* e *fully-connected layer* [55]. As duas primeiras desempenham a fase de extração de informação dos dados de entrada e, a terceira, utilizando os resultados destas duas, tratará da fase de retorno de resultados, como por exemplo a classificação de uma imagem numa tarefa cujo propósito seja esse (por exemplo, dizer se uma imagem tem ou não um gato). Por norma, a sua arquitetura consiste na repetição de vários blocos de *convolutional* e *pooling layers* em pares, seguidos de uma ou mais *fully-connected layers*, como se pode observar na figura 10.

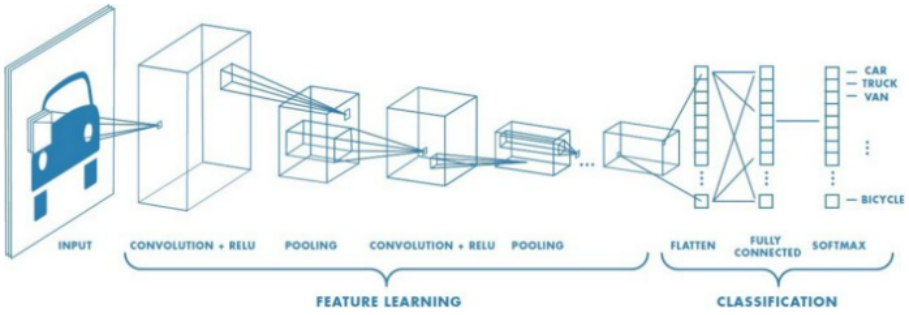


Figura 10: Típica estrutura de uma rede neuronal convolucional (adaptada de [6]).

Convolutional layer

As redes neuronais convolucionais obtiveram o seu nome devido a uma das mais importantes operações executadas na sua arquitetura: convolução. Esta operação é executada na *convolutional layer* e daí esta camada ser reconhecida como um dos blocos principais da arquitetura deste tipo de redes [55]. O seu procedimento será descrito a seguir com maior detalhe.

A abordagem das RNC para analisar imagens passa por fazer inspeções mais pequenas, isto é, dividindo estes dados em imagens de dimensões inferiores. Para este processo de análise foram criados filtros, também conhecidos por *kernels* [55][45], definidos como uma janela de pequena extensão que será movida ao longo de todos os píxeis possíveis da imagem recebida com o intuito de detetar uma especificidade.

No decorrer da aplicação dos filtros ao *input*, é executada uma multiplicação ponto a ponto entre a secção da imagem coberta pelo *kernel* e o mesmo, seguido de um somatório dos resultados das multiplicações. Todo o desenvolvimento da aplicação dos filtros pode ser observado na figura 11.

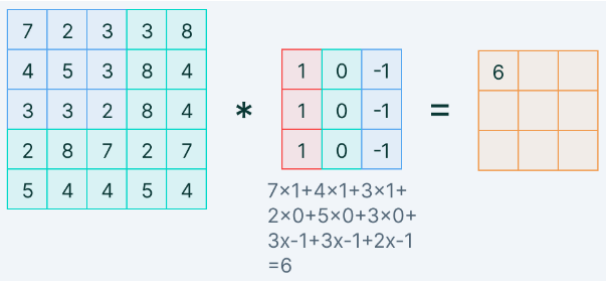


Figura 11: Exemplo da aplicação de um filtro/*kernel* (adaptada de [7]).

É importante observar que, com a translação de um filtro por todos os pontos da matriz representante da imagem de *input*, surge a possibilidade de identificação de uma característica repetida em vários locais.

Deste procedimento resulta uma matriz denominada *feature map* [55]. É de acrescentar que, com a utilização de vários filtros, as RNC podem ainda aprender a identificar e entender múltiplas características paralelamente, resultando, nesses casos, num *feature map* para cada um desses filtros. Por fim, quando terminada a fase de criação dos *feature maps*, estes são empilhados e retornados como dados de saída da camada. Na figura 12 está exemplificado o processo de criação do *feature map* com apenas 1 filtro.

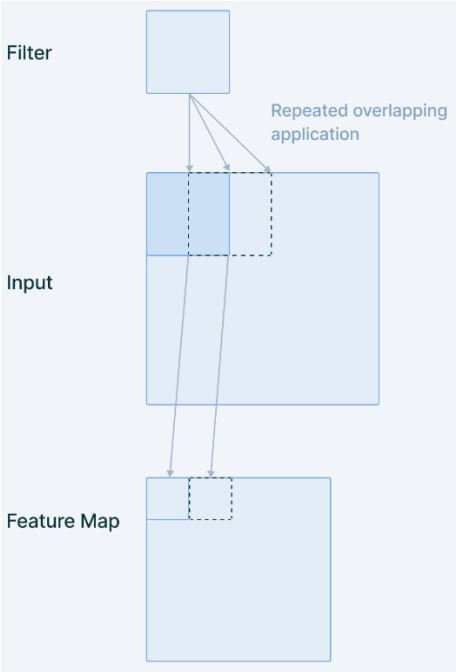


Figura 12: Criação do *feature map* (adaptada de [7]).

Duas noções muito importantes surgem neste tipo de redes neuronais artificiais: conectividade local e arranjo espacial.

A conectividade local é uma abordagem que define que cada neurónio ficará encarregue de um espaço restrito de toda a área visual, conceito que foi introduzido na secção 2.4. Para tal, cada neurónio só estará conectado a um grupo restrito de neurónios da camada anterior. Assim, e cumprindo o objetivo desta abordagem, cada neurónio só receberá informação acerca de uma parte específica da imagem que for recebida [45]. Este limite espacial imposto pela quantidade de conectividades é o campo recetivo do

neurónio, outro conceito também já referido na secção 2.4, e que vai determinar o tamanho do filtro que será aplicado. Na prática, tome-se como exemplo uma imagem com tamanho $128 \times 128 \times 3$ e um filtro $5 \times 5 \times 3$. Então, cada neurónio terá um total de $5 \times 5 \times 3 = 75$ conexões (informação de 75 píxeis).

A escolha do tamanho do filtro é extremamente importante porque terá um grande impacto na tarefa de classificação da imagem. Janelas mais pequenas são capazes de obter muito mais informação sobre características muito específicas em determinados locais da imagem, e, pelo facto de reduzir pouco o tamanho da imagem, também possibilita a criação de redes mais profundas, isto é, com mais camadas. Contrariamente, janelas maiores extraem menos informação e levam a uma maior rapidez na redução do tamanho das imagens e a um menor número de camadas, o que costuma conduzir a desempenhos mais fracos. Ainda assim, este tipo de janelas tem a sua utilidade sendo mais apropriadas para extrair características maiores. Resumindo, o tamanho desta janela dependerá sempre da tarefa e do *dataset* em questão, ainda que janelas mais pequenas, por norma, levem a melhores desempenhos por conseguirem aprender características muito mais complexas devido à profundidade da rede [55][45][52] .

O arranjo espacial controla o volume dos dados de saída dos neurónios e como estes estarão organizados. Para tal são definidos 3 hiperparâmetros:

- *Padding*: Para uma imagem de tamanho $n \times n$ e um filtro $f \times f$, a dimensão resultante da aplicação do filtro será $(n - f + 1) \times (n - f + 1)$. Caso fosse uma imagem 8×8 e um filtro 3×3, o resultado seria 6×6, ou seja, a imagem encolhe sempre que é aplicada uma operação convolucional. Isto limita o número de vezes que esta aplicação pode ser repetida. Para além disto, os píxeis dos cantos são muito menos utilizados do que os do centro o que faz com que a informação dos cantos não seja tão bem preservada como a do meio. O *padding* serve para adicionar uma camada de valores à volta da imagem de *input* para que se evitem os problemas enumerados anteriormente. Supondo o caso anterior, caso se acrescentasse uma camada de píxeis à volta da imagem, o resultado já seria de tamanho 8×8. Assim, utilizando *padding*, aumenta-se a contribuição dos píxeis nos cantos da imagem original movendo-os ligeiramente mais para o centro da imagem [55][45].

- *Stride*: Indica quantos píxeis (para o lado e para baixo) a janela se deve mover a cada iteração. O impacto que o *stride* tem na RNC é semelhante ao do tamanho do filtro. Quanto menor o

stride, mais especificidades da imagem são aprendidas já que há mais informação da imagem a ser recolhida e analisada. O aumento do *stride* limita a extração de características e o número de camadas [55] [45].

- *Depth*: A profundidade dos dados de saída será igual ao número de filtros que forem aplicados, com o objetivo de descobrir diferentes características das imagens.

Na figura 13 é exemplificado o processo de aplicação de um *kernel* de dimensões $3 \times 3 \times 3$ quando atribuído o valor 1 tanto ao hiperparâmetro *padding* como ao *stride*, sendo o primeiro bloco da imagem ilustrador da primeira iteração e, depois, o segundo bloco, a segunda iteração.

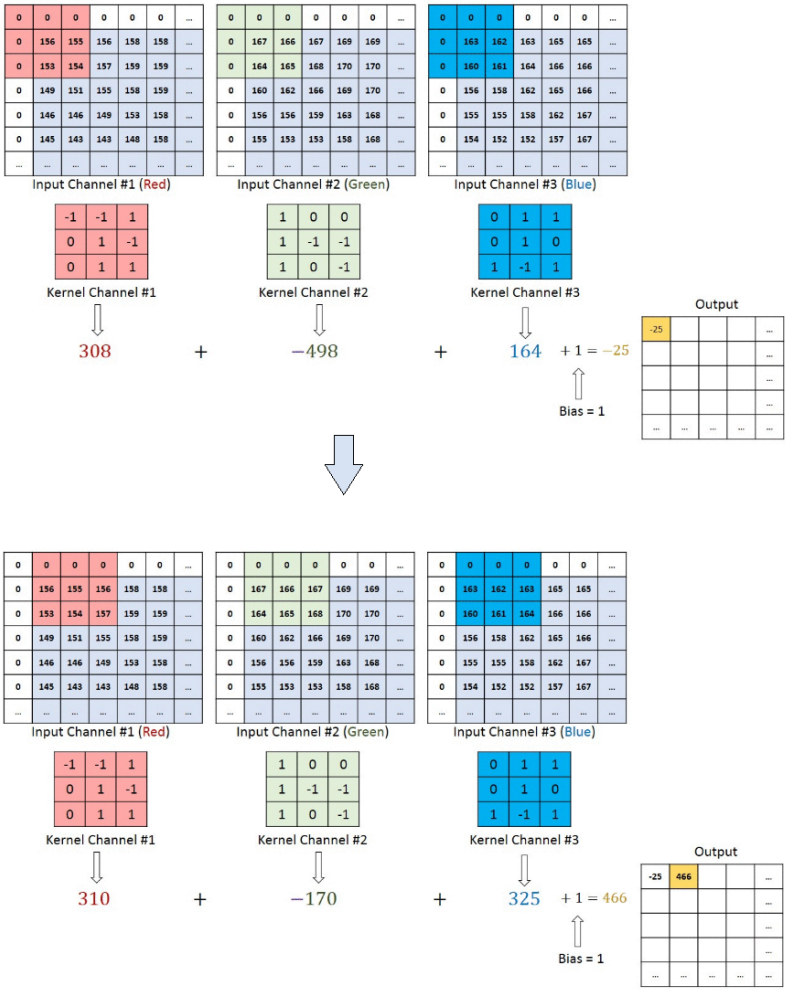


Figura 13: Exemplo de uma operação convolucional (adaptada de [6]).

O último conceito que será apresentado para esta camada é a partilha de parâmetros. Como se viu na figura 12, o mesmo filtro é aplicado em diferentes regiões da imagem. Assim, em vez de se aprender uma matriz de pesos para cada neurónio, utiliza-se a mesma para todos, o que leva a uma melhoria drástica no tempo de treino [55]. De notar que, a imagem de um cão será sempre a imagem de um cão esteja em que posição estiver, e as RNC têm em conta esta propriedade partilhando os parâmetros ao longo dos vários blocos de imagem selecionados. Assim, é possível identificar um cão com a mesma matriz de características quer o cão esteja na coluna i ou na coluna $i + 100$ da imagem.

Por fim, tal como acontece nas redes neuronais artificais, um dos grandes fatores para as redes neuronais convolucionais conseguirem resultados excecionais é a sua não-linearidade. Nesta camada é utilizada a função de ativação ReLU (figura 3) uma vez que foi observado por Saha et al. [56] que as RNC treinavam mais rápido ao utilizar esta função. Assim, todos os valores negativos nos resultados da *convolutional layer* são substituídos por 0 evitando que a soma destes seja nula.

Pooling layer

Tendo como propósito a redução do tamanho da representação da imagem para diminuir o poder computacional necessário ao processar os dados, é acrescentada uma *pooling layer* no fim de cada *convolutional layer* da arquitetura [55][52]. Mais, esta camada é ainda muito útil para identificar as características mais dominantes já que estas são invariáveis à rotação e posição devido à sua importância. Nos casos em que as imagens são a cores, esta camada opera independentemente em cada uma das representações dos canais e, posteriormente, são sobrepostas.

Dos vários métodos de aplicação de *pooling* os mais comuns são *max pooling* e *average pooling*. O *max pooling* retorna o valor máximo dos valores que a janela cobre do *feature map*. Assim, depois desta aplicação, o resultado será um *feature map* com as características mais dominantes do mapa. No caso do *average pooling* é feita uma média dos valores selecionados [45].

Os hiperparâmetros definidos nesta camada são o tamanho da janela, que indica as dimensões da janela que será passada ao longo do mapa, e o *stride*, que indica quantos píxeis são movidos para o lado e para baixo ao movimentar esta janela [55].

Na figura 14 é demonstrada a aplicação de *pooling* com uma janela de tamanho 2×2 e *stride* 2.

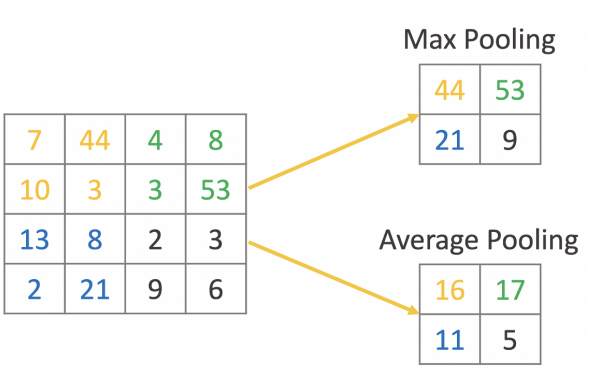


Figura 14: Resultado de aplicação dos dois tipos de *pooling* (adaptada de [8]).

Fully-connected layer

Como foi dito na subsecção 2.4.2, a *convolutional layer* e a *pooling layer* formam um bloco nas redes neuronais convolucionais. A quantidade destes blocos que uma rede terá pode ser aumentada com o objetivo de capturar características mais específicas e mínimas, dependendo da complexidade da tarefa em questão e com o custo de maior poder computacional.

Depois de serem obtidas as informações desejadas, a representação final das características da imagem, estas são enviadas para uma *flatten layer* que irá convertê-las num vetor 1D, isto é, por exemplo, um tensor $5 \times 5 \times 2$ é convertido num vetor de tamanho 50. Este vetor é, por fim, passado a uma *fully-connected layer* com uma função de ativação adequada à tarefa [55]. No caso de ser uma tarefa de classificação de imagens, a *softmax* é, por norma, a escolha mais usual.

A função *softmax*, utilizada como ativação, transforma um vetor de valores num vetor de probabilidades e é definida por [55]:

$$softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{12}$$

onde z é o vetor dos valores, $e \approx 2.718$ é o número de Nepper e a i-ésima entrada do vetor de saída de $softmax(z)$ é a probabilidade prevista para a classe i.

3 Estimação de pose

3.1 Introdução

A estimação de pose é uma categoria de problemas centrados na descoberta da postura de uma pessoa, ou objeto, em dados visuais, como imagens ou vídeos [10][57]. A resolução destes problemas passa pela identificação e localização de certos *keypoints* e, posteriormente, o seu *tracking*, caso sejam tarefas que envolvam movimento (vídeos/sequência de *frames*). Estes *keypoints* podem ser um ombro ou joelho, para o caso de uma pessoa, ou cantos e particularidades, para o caso dos objetos [10][57]. Em ambas as situações, o objetivo do modelo é sempre encontrar e seguir os *keypoints* escolhidos. De notar que, tarefas que envolvem humanos serão usualmente mais difíceis já que a tendência para a seleção dos *keypoints* nos humanos são articulações e, por isso, as posições relativas entre estas podem variar imenso, uma situação muito menos frequente nos objetos que, maioritariamente, possuem características rígidas e fixas entre elas.

Até ao surgimento desta área, a tradicional deteção de objetos tratava a identificação de pessoas num dado visual como uma simples caixa, denominada *bounding box*, que envolveria totalmente o corpo da pessoa encontrada. Tendo como motivação habilitar os computadores a compreender a linguagem corporal humana, começaram a ser desenvolvidos vários métodos e técnicas com o intuito de conseguir identificar, não só onde estaria uma pessoa, mas também todos os seus *keypoints*.

Uma possível subcategoria da estimação de pose é a estimação de pose humana. Esta foca-se totalmente na previsão de poses apenas em seres humanos e na localização dos *keypoints* definidos. Por norma, e como já foi dito anteriormente, estes são as articulações. É importante referir que, uma vez que os movimentos das poses são conduzidos por uma certa ação, conhecer a pose de um humano é fundamental para o reconhecimento de ações.

Várias alterações no aspeto do corpo humano como a forma da roupa, a oclusão arbitrária de *keypoints*, oclusão devido a ângulo de visão e o contexto de fundo, tornam a estimação de pose humana extremamente desafiadora. Uma tarefa como localizar os *keypoints* já é por si difícil para os modelos de processamento de imagens, e fica ainda mais complicada quando estas articulações são muito pouco visíveis. Assim, este tipo de previsão precisa de ser robusta para variações do mundo real como a iluminação

e o ambiente.

Para além da dificuldade em criar modelos prontos para as várias situações imprevisíveis do mundo real que podem afetar as imagens e vídeos que serão trabalhados, também existe a necessidade de grandes recursos computacionais, o que limita a eficácia das previsões. Ainda assim, hoje em dia, com os mais recentes avanços na área e *datasets* disponíveis [10], é possível criarem-se aplicações em tempo real que demonstram o potencial deste tipo de deteção, tais como carros autónomos e robôs de entregas.

Atualmente, os modelos de processamento de imagens mais poderosos são baseados em redes neuronais convolucionais, e, como tal, também para a área de estimação de pose são utilizados maioritariamente modelos com arquitetura de redes neuronais convolucionais, muitas vezes especialmente adaptados para a inferência da pose humana.

3.2 Metodologia

Aquando da inferência da pose humana, existem 3 pontos fundamentais que devem ser tidos em conta:

- espaço;
- quantidade de poses;
- procedimento.

Estes 3 pontos resultam em métodos 2D ou 3D (espaço), *single* ou *multi-poses* (quantidade de poses) e *bottom-up/top-down* ou *direct-regression/heatmap-based* (procedimento).

Tendo em conta a tarefa e quantidade de informação necessária relativa ao espaço, os algoritmos podem ser de estimação de pose 2D ou 3D.

Os métodos de estimação de pose 2D calculam a posição 2D ou a localização espacial de um *keypoint* do corpo humano em imagens ou vídeos. Numa primeira fase da área de estimação de pose, a extração de *features* era responsabilidade de técnicas criadas manualmente. Mais, o corpo humano era apenas ilustrado como uma figura com traços a representar os membros. Nos dias de hoje, as abordagens à base de *Deep Learning* melhoraram imenso o desempenho da estimação de pose 2D e é possível representar o corpo humano de forma muito mais realista, como será visto mais à frente [9].

Para além da localização espacial, a estimação de pose 3D fornece ainda resultados sobre a profundidade de uma pose, isto é, a localização de um *keypoint* em vez de ser apenas (x, y) como no 2D, passa a ser do formato (x, y, z), onde z é uma medida de profundidade, isto é, a distância do ponto ao sensor da câmara. Apesar dos avanços significativos na estimação de pose humana 2D, os casos 3D continuam a ser um grande desafio. A maioria dos trabalhos existentes trata deste tipo de tarefas a partir de imagens monoculares ou vídeos, o que é um problema mal elaborado e inverso devido à projeção de 3D no referencial 2D, onde uma dimensão é perdida. Quando estão disponíveis várias perspetivas ou informação de sensores como IMU e LiDAR, a estimação de pose 3D pode ser um problema bem implementado ao utilizar técnicas de fusão de dados. Outro problema que surge é a dificuldade de obter *datasets* apropriados uma vez que tanto a obtenção de dados visuais como a sua anotação manual não são nada práticos.

Conforme a quantidade de elementos a identificar num dado visual, uma tarefa de estimação de pose pode tornar-se cada vez mais complexa, demorada e dispendiosa.

Assim, todos os métodos de estimação de pose humana começam por ser incluídos em duas categorias tendo em conta o número de indivíduos na imagem a ser avaliada: abordagens *Single-person* e abordagens *Multi-person* [10][57].

Quando se trata de uma imagem de *input* onde apenas está presente uma pessoa, recorre-se a *Single-person*. Assim, o sistema deteta a pose de um indivíduo específico existente na imagem. Tendo em conta esta informação posicional, isto é, sabendo que nos dados apenas surge uma pessoa, a abordagem *Single-person* tenta resolver o que pode ser visto como um problema de regressão, uma vez que tenta localizar os *keypoints* sabendo também a quantidade destes. Em casos em que o número de indivíduos que aparecem na imagem é superior a 1, emprega-se a abordagem *Multi-person*. Ao contrário do primeiro caso, neste tenta-se resolver um problema sem restrições visto que tanto o número de pessoas como a sua posição são uma incógnita [57].

Dentro de cada uma das abordagens, *Single-person* e *Multi-person*, existem métodos diferentes para se resolver o problema de identificação dos *keypoints*, métodos estes que podem ser classificados conforme o seu procedimento.

Os métodos que fazem parte da abordagem *Single-person* podem ser distribuídos pelos baseados em *Direct-regression* e os baseados em *Heatmaps*. Os primeiros aproveitam-se de *features maps* para fazer a regressão dos *keypoints* diretamente. Já os segundos, começam pela geração de um *heatmap*,

uma matriz representadora de uma imagem onde o valor de cada pixel define a possibilidade de existir um *keypoint* nessa posição, e, depois, passam à fase de previsões baseando-se sempre nas informações destes mapas [57].

É difícil decidir que métodos são considerados os melhores. Por um lado, os baseados em *Direct-regression* são rápidos e simples, treinados da forma usual e sem a utilização de novos parâmetros como os *heatmaps*. Para além disto são facilmente adaptáveis a cenários 3D. Porém, têm limitações, como o facto de serem inutilizáveis em cenários de *Multi-person*. Por outro lado, os baseados em *Heatmaps* são de fácil compreensão visual, o que melhora o entendimento dos humanos sobre o processo da rede, e pode ser aplicado em casos mais complexos. Em contrapartida, o seu consumo de memória é muito elevado para que se consiga obter *heatmaps* de alta resolução e são difíceis de adaptar a cenários 3D. Concluindo, cada um tem as suas vantagens e desvantagens e por isso a escolha será sempre dependente do objetivo do projeto e da tarefa em questão [57].

Passando agora para análise dos métodos que fazem parte da abordagem *Multi-person*, estes podem ser divididos em métodos *Top-down* e *Bottom-up*. As imagens tratadas por métodos *Top-down* passam por duas fases: a deteção de humanos e a estimação de pose humana *Single-person*, isto é, começa-se por obter uma *bounding box* para cada pessoa e, depois, aplica-se a estimação de pose *Single-person* a cada uma delas. Seguindo um processo quase inverso ao anterior, os métodos *Bottom-up* começam pela identificação de todos os *keypoints* presentes e, de seguida, passa para uma fase de agrupamento destes tendo em conta que devem fazer parte do mesmo indivíduo [57].

Ambas as subcategorias do método *Multi-person* apresentam as suas limitações. As *Top-down*, devido ao seu procedimento, são muito dependentes da identificação de humanos, visto que, caso a deteção falhe, não há a possibilidade de correção. Para além disto, o custo computacional de um destes modelos varia com o número de pessoas apresentadas na imagem, isto é, o tempo de processamento é diretamente proporcional ao número de indivíduos já que, por cada pessoa, é executada uma estimação de pose *Single-person*. Apesar de conseguir ultrapassar a dependência de identificação de humanos, tornando o tempo de processamento independente do número de pessoas, nos métodos *Bottom-up* surgem outros problemas. Uma das maiores dificuldades é a separação dos *keypoints*, um problema ainda maior quando existe uma grande sobreposição entre corpos diferentes [10].

Na comparação das abordagens *Bottom-up* e *Top-down* são tidos em conta dois fatores extrema-

mente importantes: a *accuracy* e a velocidade.

Comecemos por analisar a *accuracy* . Em primeiro lugar, tanto o vencedor do *COCO 2017 keypoint* como o de *AI Challenger Human Skeletal System Keypoints Challenge* apresentaram modelos que se aproveitam da abordagem *Top-down*. Uma observação importante relativamente às *Bottom-up* é que podem não conseguir aprender *features* consistentes a partir das imagens devido à grande variação de representação destes *keypoints* em diferentes pessoas. A resolução de um *feature map*, assim como a capacidade de uma rede neuronal, estão limitados pela capacidade da GPU. Ora, a resolução média dos métodos *Single-person* na abordagem *Bottom-up* é inferior à da abordagem *Top-down* durante a fase de treino com a mesma rede e memória de GPU. Assim, o que realmente limita a *accuracy* dos *Bottom-up* podem mesmo ser as limitações do *hardware*.

Relativamente à velocidade, nos *Top-down*, a pose das pessoas é calculada uma a uma, o que, como foi visto anteriormente, consome tempo que aumenta linearmente com o aumento de humanos na imagem. Por outro lado, a imagem passa apenas uma vez por uma rede que siga a *Bottom-up*. Como é possível atingir um tempo de agrupamento de *keypoints* realmente baixo, a *Bottom-up* é capaz de atingir uma maior velocidade de previsão [57]. Tendo ambas as suas vantagens e desvantagens, tal como as abordagens anteriores, a escolha de uma destas dependerá sempre da tarefa em questão e do resultado pretendido.

3.3 Modelação do corpo humano

Um aspeto muito importante na área de estimação de pose humana é a representação dos resultados obtidos. Um corpo humano é uma entidade muito complexa, composta por articulações, membros, texturas, entre outros, e, portanto, é importante definir uma boa reprodução de todas estas informações. Para tal, existem 3 tipos de modelos que servem para este efeito, tendo sempre em conta que todos terão as suas vantagens e desvantagens: *Kinematic*, *Planar* e *Volumetric*. Na figura 15 estão representadas as 3 opções [9].

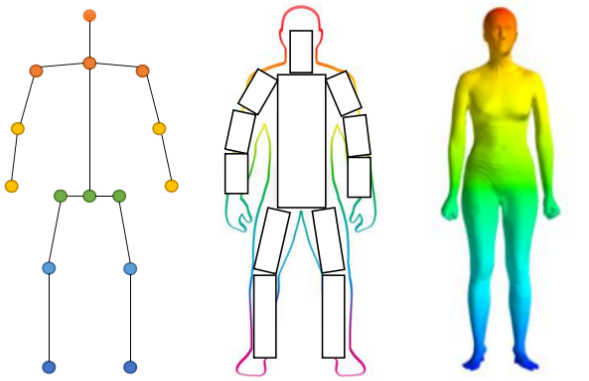


Figura 15: Opções para modelação do corpo humano. (adaptada de [9]).

Através de um conjunto de posições de articulações e das orientações de membros, é possível construir a representação de um corpo humano segundo o modelo *Kinematic*. A visualização deste modelo é bastante intuitiva e a sua estrutura é muito flexível. Para além disto, pode ser aplicado tanto em tarefas de estimação de pose 2D como em 3D. Contudo, demonstra grandes limitações na exibição de texturas e de informações sobre a forma do indivíduo [9].

O modelo *Planar*, para além de capturar as relações entre diferentes partes do corpo como o *Kinematic*, é ainda capaz de representar a forma e a aparência do corpo humano. Por norma, as partes do corpo são simbolizadas por retângulos, aproximando assim a imagem final dos verdadeiros contornos de um indivíduo. No entanto é apenas aplicável em tarefas de estimação de pose 2D [9].

Por fim, o *Volumetric* é capaz de obter as relações entre partes do corpo, assim como a textura e o relevo, e de desenhar formas muito semelhantes às verdadeiras do corpo humano.

3.4 Estimação de pose com Deep Learning

Com o desenvolvimento e evolução das soluções da área de *Deep Learning* nos últimos anos, estes modelos mostraram-se capazes de superar os métodos de Visão por Computador clássicos em várias tarefas como segmentação de imagem e deteção de objetos. Com isto, as técnicas de *Deep Learning* apresentaram avanços significativos e ganhos de desempenho em problemas de estimação de pose.

Seguindo agora uma seleção feita por Munea et al. [10], assim como a sua explicação, serão apresen-

tados alguns dos algoritmos mais eficazes para a área de estimação de pose.

3.4.1 DeepPose

Em 2014, Toshev et al. [58] apresentaram uma nova arquitetura de modelo dedicada à estimação de pose *Single-person*. Para a sua elaboração, encararam a descoberta das localizações de *keypoints* como uma problema de regressão baseado em redes neuronais convolucionais. Para além disto, utilizaram como arquitetura *backbone* no seu modelo a *AlexNet* [25] com o objetivo de analisar os efeitos do treino de uma arquitetura *multi-stage* quando combinada com supervisão intermédia repetida.

O *DeepPose* melhora as suas primeiras previsões, isto é, as localizações de cada *keypoint*, recorrendo a uma série de regressores, a chamada *Cascade of Regressors*, onde cada um dos regressores retorna as posições dos *keypoints* no formato (x, y). No fim desta primeira previsão, a imagem será transformada em múltiplas imagens mais pequenas, uma por cada *keypoint*, sendo cada uma delas centrada na localização de cada um e com uma certa medida de largura. No fim deste processo, todas as novas imagens serão enviadas para a próxima fase. Ao redimensionar a imagem original em imagens mais específicas para cada *keypoint*, o modelo consegue aprender *features* mais locais visto que, com resoluções maiores, conseguem obter uma melhor previsão.

Assim, o *DeepPose* é composto por 3 fases de *Cascade of Regressors*, ou seja, as imagens fazem 3 passagens por esta série de regressores para estimar a pose de um indivíduo numa imagem.

Na figura 16 é possível observar a estrutura desta rede assim como o seu procedimento de melhoria de previsões. De notar ainda que a verde estão representadas as *convolutional layers* e a azul as *fully-connected layers*.

Figura 16: Estrutura da rede *DeepPose*. (adaptada de [10]).

O seu desempenho foi avaliado utilizando os *datasets* FLIC[59] e LSP[60] e as métricas de avaliação

foram PCK[58] (percentagem de keypoints corretos - *Percentage of Correct Key-points*) e PCP[58] (percentagem de partes corretas - *Percentage of Correct Parts*).

3.4.2 ConvNet Pose

Tompson et al. [61] propuseram uma arquitetura para uma rede neuronal convolucional multi-resolução. Esta arquitetura tem a particularidade de ser responsável pelo desenvolvimento de *heatmaps*, em vez de uma regressão contínua que prevê a probabilidade da localização de um *keypoint* em imagens *RGB* monoculares. Esta rede aproveita-se de várias arquiteturas RNC de diversas resoluções em paralelo para descobrir *features* de diferentes escalas simultaneamente.

Através de uma *sliding window*, começa por ser produzido um *heatmap* pouco cuidadoso, que será posteriormente melhorado por uma *ConvNet* de refinamento de pose. Este refinamento permite obter uma localização mais correta dos *keypoints*, o que ajuda na recuperação da eficácia espacial perdida na aplicação de *pooling* no primeiro modelo.

Portanto, o modelo é composto por 3 módulos, isto é, 3 redes neuronais convolucionais. A primeira trata da localização menos precisa dos *keypoints*. A segunda trata do corte da imagem original tendo em conta as posições previstas anteriormente. E, por fim, a terceira aplica o *fine-tunning*, fazendo os últimos ajustes necessários nos parâmetros da rede. Esta estrutura é explicada na figura 17.

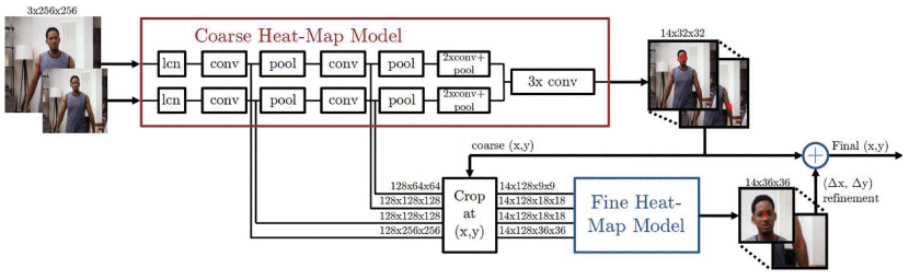


Figura 17: Estrutura da rede *ConvNet*. (adaptada de [10]).

Este modelo foi avaliado pelas métricas PCK[61] e PCKh@0.5[61] com os *datasets* FLIC [59] e MPII [62]. O PCKh é uma variante do PCK em que se define o limite de correspondência exata como 50% do comprimento do segmento da cabeça (*head* - h).

3.4.3 CPM: Convolutional Pose Machines

Apresentada no trabalho de Wei et al. [63], a *CPM* tem como um dos seus principais focos a aprendizagem de relações espaciais de longo alcance à custa de grandes campos recetivos. Para tal, a sua estrutura é composta por uma sequência de redes neuronais convolucionais que criam *belief maps* 2D para ajudar a identificar as posições dos *keypoints*. Esta estrutura implica um tipo de previsão sequencial, o que leva a rede a, para além de conseguir entender informações espaciais implícitas, é também capaz de perceber a representação de *features* nas imagens ao mesmo tempo. Para além disto, a sua estrutura *multi-stage* torna possível que os *belief maps* sejam enviados para a fase seguinte como *input*. A *CPM* aplica ainda supervisão intermédia no fim de cada fase para evitar o problema de desaparecimento do gradiente.

Na figura 18 está representada toda a estrutura e os campos recetivos da *CPM*. A rede *CPM* é dividida em várias fases, a quantidade das quais é considerada um hiperparâmetro por norma com o valor 3, e, a cada fase, o *belief map* de cada *keypoint* é calculado. As secções a) e b) exemplificam a estrutura da *pose machine*, as c) e d) mostram as redes neuronais convolucionais correspondentes e, por fim, a e) mostra os campos recetivos ao longo de várias fases.

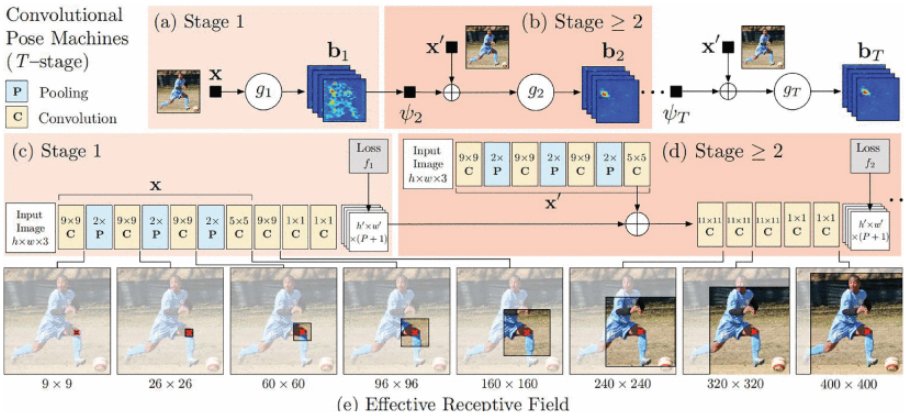


Figura 18: Estrutura da rede *CPM* (adaptada de [10]).

Numa primeira fase, uma rede neuronal convolucional básica, como uma *VGG*, representada por X, começa por gerar os *belief maps* de cada *keypoint* a partir da imagem original. Mais concretamente, se a imagem contiver k *keypoints*, então o *belief map* terá k camadas, cada uma representativa do

heatmap de cada *keypoint*. De notar ainda que, a *loss* de todas as camadas é somada e utilizada na supervisão intermédia, o que ajuda nos problemas de desaparecimento de gradiente.

Nas fases posteriores a estrutura mantém-se e ainda será acrescentado o *belief map* ao *input*. Assim, a fase seguinte receberá o *belief map* e o resultado da passagem da imagem original por X'.

A *CPM* foi avaliada utilizando 3 *datasets* diferentes: MPII [62], LSP[60] e FLIC[59], e utilizaram-se as métricas PCK@0.1[63], PCK@0.2[63] e PCKh@0.5[63].

3.4.4 Stacked Hourglass

Newell et al. [64] apresentaram uma rede composta por sucessivas aplicações de *pooling* e *up-sampling* (aumento de resolução) com o objetivo de capturar informação a todos os níveis. No caso de estimação de pose humana, este tipo de informação seria, por exemplo, a orientação de um indivíduo, a relação entre articulações adjacentes e a posição dos membros.

De reparar que, tendo em conta a sua organização, esta rede usufrui tanto da abordagem *Bottom-up*, ao converter imagens de alta resolução para menor resolução com o *pooling*, como da abordagem *Top-down*, ao converter imagens de baixa resolução para alta resolução com recurso a *up-sampling*. Para além disto, e tal como a *CPM* e a *DeepPose*, aplica ainda supervisão intermédia. Devido à sua capacidade de identificar *features* de várias escalas diferentes, consegue identificar tanto informação local como global. Na figura 19 está representado um dos vários módulos desta rede e, na figura 20, está representada a estrutura total da rede, onde figuram vários módulos. O bloco da figura 19 está representado na figura 20 por duas pirâmides sob a linha a tracejado.

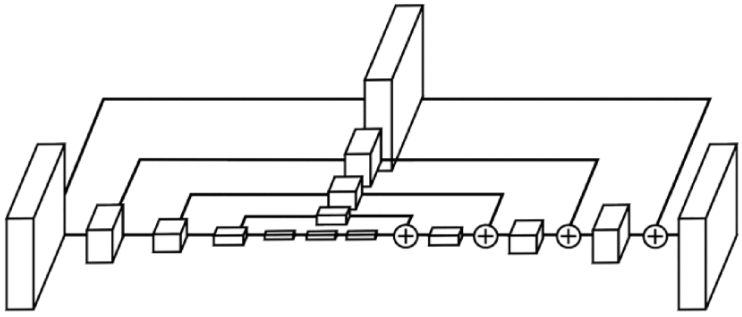


Figura 19: Um bloco/módulo da estrutura da rede *Stacked Hourglass* (adaptada de [10]).

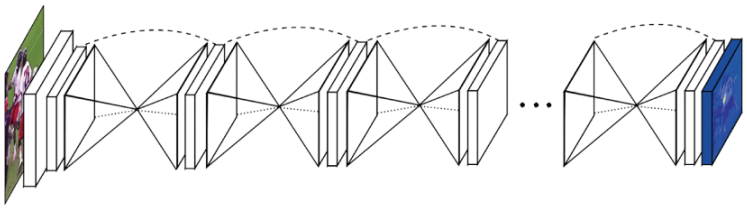


Figura 20: Estrutura da rede *Stacked Hourglass* (adaptada de [10]).

Por cada um dos módulos, é aplicado uma sub-rede *Hourglass* de 4° ordem que extrai *features* da escala original para $\frac{1}{16}$ desta. Esta extração não altera o tamanho dos dados, apenas a sua profundidade. O módulo é utilizado para capturar as características locais presentes na imagem a diferentes escalas.

Passemos agora ao procedimento que ocorre dentro de cada um destes módulos. No início, uma *Convolutional layer* e uma *Max Pooling layer* são aplicadas para reduzir as características a uma escala inferior. A cada *Max Pooling*, a rede divide-se e aplica uma convolução para obter as *features* utilizando a resolução original, isto é, antes do *pooling*. Depois de conseguir as *features* de escala mais baixa, a rede aplica então *up-sampling* e, gradualmente, combina informações de escalas variadas.

Na aplicação de *down-sampling* a rede utiliza *max-pooling* e, no *up-sampling*, aplica *nearest-neighbor interpolation*.

Esta rede foi testada com os *datasets* MPII[62] e FLIC[59] com as métricas PCK@0.2[64] e PCKh@0.5[64].

3.4.5 IEF: Iterative Error Feedback

Carreira et al. [65] definiram como base do seu modelo a correção de resultados. Mais concretamente, o processo de previsão, seguido de identificação dos erros desta previsão, e, por fim, a aplicação de uma correção iterativa executada à custa de um mecanismo de *feedback top-down*, é a essência desta implementação. Carreira et al. [65] apresentam ainda uma estrutura de rede neuronal convolucional usual com o acrescento da possibilidade de utilização dos espaços do *input* e do *output*. No que toca a previsões, este modelo não identifica diretamente os *keypoints* numa imagem. Em vez disso, previsões do erro são continuamente e progressivamente enviadas para a solução inicial por um modelo de auto-correção. Na figura 21 é possível ver a implementação deste modelo para a estimação de pose humana.

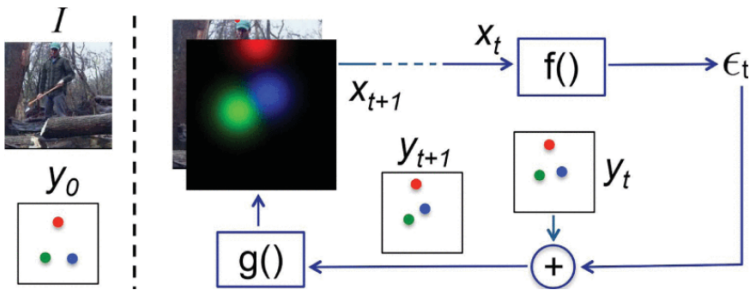


Figura 21: Estrutura da rede *IEF* (adaptada de [10]).

No lado esquerdo da figura 21 estão representados por I uma imagem e por y_0 os *keypoints* inicialmente calculados. Os 3 pontos podem simbolizar a posição da cabeça, e dos dois pulsos, por exemplo. Depois é definido X_t como $X_t = I \oplus g(y_{t-1})$, onde I representa a imagem e y_{t-1} é o *output* anterior (concatenação entre a imagem *RGB* I e a representação visual, g, do *output* estimado). A função $f(X_t)$, modelada como uma rede neuronal convolucional, produz a correção ϵ_t como resultado e, este resultado, é adicionado ao *output* atual y_t para produzir y_{t+1}, que significa que a correção foi considerada. A função $g(y_{t+1})$ converte cada posição dos *keypoints* num canal de *heatmap* gaussiano para fazer parte do *input* com a imagem na próxima iteração. Este procedimento é repetido T vezes até que se obtenha um y_{t+1} melhorado que é muito próximo do valor real.

Este modelo foi avaliado com dois *datasets*, o LSP[60] e MPII[62], e apenas se utilizou uma métrica de avaliação, a PCKh@0.5[65].

3.4.6 Part Affinity Fields

No artigo de Cao et al. [66], o principal foco foi a identificação das maiores dificuldades enfrentadas em tarefas de estimação de pose *Multi-person*. Alguns dos problemas encontrados foram o cálculo do número de indivíduos na imagem, as oclusões que a interação entre indivíduos provocava e as diferentes escalas para cada pessoa. Tendo em conta estas dificuldades, os autores desenvolveram uma nova abordagem para facilitar a conexão de partes de um corpo humano. Para tal, utilizaram *Part Affinity Fields (PAFs)*, um método sem parâmetros apropriado para modelos de estimação de pose *Multi-person* que optam pela abordagem *Bottom-up*.

Na figura 22 está exibida toda a estrutura e processamento desempenhado por este modelo. Segue-se

agora uma breve explicação do procedimento. Dada uma imagem como *input* 22.a), os *part confidence maps* vão determinar a localização de cada um dos *keypoints* 22.b). A orientação e posição de cada parte do corpo será determinada pelos *PAFs* 22.c), que será um vetor 2D representador do grau de associação entre partes do corpo. De seguida, estes candidatos a partes do corpo passam para uma fase de análise para que seja construído um conjunto de correspondência bipartida 22.d) e, por fim, é construída uma pose de corpo completa tendo em conta os resultados desta análise 22.e).

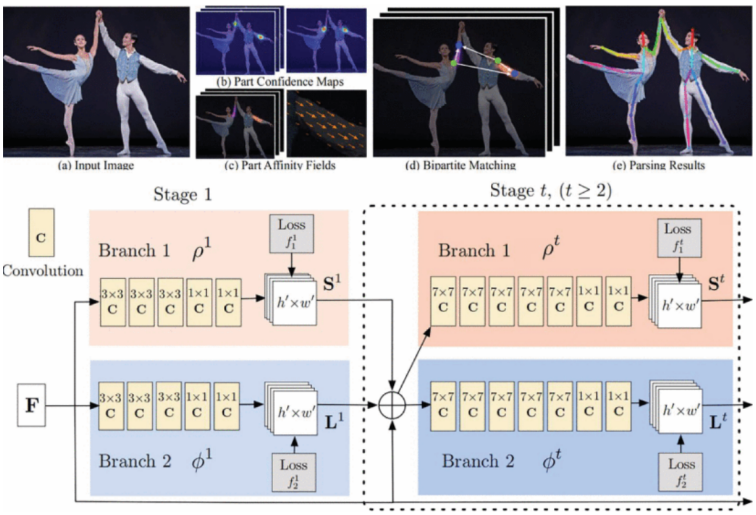


Figura 22: Estrutura e procedimento da rede que utiliza *PAFs* (adaptada de [10]).

Depois de uma breve explicação mais prática do que ocorre nesta rede, passemos a uma análise mais técnica. A rede neuronal convolucional *multi-stage* de dois caminhos começa por receber um *feature map* F de uma imagem tratada pelas 10 primeiras camadas de uma arquitetura *VGG*. De seguida, o modelo cria *confidence maps*, representados por S, para prever a localização das articulações, sendo desenvolvidos J *confidence maps* para cada articulação, ou seja, $S = S_1, S_2, ..., S_J$. Simultaneamente, são criados *Affinity fields*, representados por L (conjunto de vetores 2D), para codificar a associação entre partes/membros, que contém C vetores que correspondem a cada parte, ou seja, $L = L_1, L_2, ..., L_C$.

Após o término da primeira fase, será retornado um conjunto de *confidence maps* de deteção e *part affinity fields*. Para as fases posteriores, os *inputs* serão uma combinação dos resultados das últimas duas fases e o *feature map* F. Será ainda aplicado *Bipartite matching* aos *confidence maps* e *part affinity fields*, uma inferência *greedy* que obtém os *keypoints* 2D de cada indivíduo na imagem. No fim

de cada fase é ainda implementada supervisão intermédia para evitar o problema de desaparecimento do gradiente ao restaurar o gradiente periodicamente.

Este modelo foi avaliado com os *datasets* COCO[67] e MPII[62] e as métricas utilizadas foram AP[66], mAP[66] e PCKh@0.5[66].

3.5 DeepPoseKit

Os métodos de estimação de pose apresentados anteriormente têm algumas limitações em termos de velocidade e robustez. Então, apresentamos agora um novo *kit* de ferramentas de *software* fácil de usar, o *DeepPoseKit* (desenvolvido por Graving et al. [68]), que aborda estes problemas de deteção de pose usando um modelo eficiente de *Deep Learning* em várias escalas, chamado *Stacked DenseNet*, e um algoritmo rápido de deteção baseado em GPU para estimar locais de *keypoints* com precisão de subpíxeis. Estes avanços melhoram a velocidade de processamento em ordens superiores a 2x sem perda de precisão em comparação com os métodos atualmente disponíveis [69, 70]. De notar que este *software* foi aplicado na deteção de poses de animais, mas que facilmente pode ser adaptado para estimação de poses de humanos.

Para melhorar a eficiência dos seus modelos e, assim, obter as melhores previsões o mais rapidamente possível, Graving et al. [68] basearam-se em métodos de última geração de estimação de pose [64], modelos de regressão convolucional [71] e algoritmos de Visão por Computador convencionais [72]. A partir disto, desenvolveram dois modelos, incluindo uma nova arquitetura denominada *Stacked DenseNet* e um novo método de processamento de *confidence maps*, que intitularam *subpixel maxima*, que produz deteções rápidas e exatas para a estimação da localização dos *keypoints* com precisão sub-píxel, mesmo com resoluções espaciais baixas.

Para além disto, Graving et al. [68] debatem ainda sobre a integração de um grafo de postura hierárquico para que o modelo aprenda sobre a geometria multi-escalar entre os *keypoints* do corpo do animal para melhorar a eficácia dos seus modelos.

Os trabalhos desenvolvidos por Mathis et al. [69] e Pereira et al. [70] serviram de referência para a avaliação de resultados de Graving et al. [68] com o *DeepPoseKit*. Apesar do grande avanço relativamente à qualidade e detalhe dos dados quando comparados com os métodos clássicos vistos anteriormente, estes

modelos de referência revelaram limitações na velocidade e robustez, o que pode interferir no desempenho das suas aplicações práticas.

Então, Graving et al. [68], de forma a otimizarem a sua abordagem, testaram os seus modelos em variadas experiências e, posteriormente, compararam os resultados com estes modelos de referência [69, 70]. Os fatores utilizados para estas comparações foram a velocidade, o tempo de treino e a capacidade de generalização. Quanto aos dados, foram criados 3 *datasets* de imagens que incluíam recortes de imagens com várias interações entre indivíduos.

Para facilitar a utilização dos modelos, Graving et al. [68] desenvolveram um pacote de *software*, escrito em *python* e construído em *Tensorflow*. O seu *software* oferece uma *GUI* personalizada para anotar dados de treino com *active learning*, similar a Pereira et al. [70], e um *pipeline* flexível para aplicação de *data augmentation*, treino e avaliação do modelo e execução de inferências em novos dados. Tudo isto foi organizado num módulo de *python* e foi nomeado *DeepPoseKit*.

De seguida os modelos desenvolvidos por Mathis et al. [69] e Pereira et al. [70] serão explicados com mais detalhe, sendo assim possível fazer uma comparação entre eles e uma comparação com o *DeepPoseKit*.

3.5.1 DeepLabCut e LEAP

Mathis et al. [69] e Pereira et al. [70] foram pioneiros na utilização de redes neuronais convolucionais para estimação de pose animal. Pesquisas na área de estimação de pose humana, como os trabalhos realizados por Andriluka et al. [62], Insafutdinov et al. [73] e Newell at al. [64], serviram de inspiração com a utilização de *fully-convolutional neural networks (F-CNNs)* [74], também conhecidas como modelos *encoder-decoder*. De forma a ilustrar a postura corporal de um indivíduo, estas redes são treinadas até que sejam capazes de, eficazmente, transformar imagens em estimativas probabilísticas da localização dos *keypoints* e, assim, gerar os *confidence maps*. Estes mapas representam a postura de um ou mais indivíduos e são a chave para, ao serem processados, se obterem as melhores previsões das coordenadas dos *keypoints* num espaço 2D.

Como já foi dito anteriormente, a necessidade de grandes quantidades de dados era um problema recorrente na resolução de problemas de *Machine Learning* e os algoritmos de *Deep Learning* não

escapavam à regra. No entanto, Mathis et al. [69] e Pereira et al. [70] demonstraram que também com quantidades inferiores às esperadas de dados é possível obter uma eficácia semelhante ao desempenho humano. Nos seus trabalhos, de forma a assegurar a generalização para grandes *datasets*, ambos apresentaram ideias baseadas num refinamento iterativo do *dataset* de treino que permitisse obter o melhor modelo possível. Mais concretamente, Pereira et al. [70] descreve uma técnica, denominada *active learning*, que reduz imenso o tempo de anotação ao aproveitar um modelo treinado para inicializar novos dados de treino. Mathis et al. [69] também descreve várias técnicas para o aperfeiçoamento dos dados de treino, assim como para a minimização dos erros ao se executarem previsões no conjunto de dados completo. Esta lista de técnicas incluía a filtragem dos dados e a seleção de novos dados de treino tendo como referência a confiança do modelo e a entropia dos *confidence maps* que resultam do modelo.

DeepLabCut

O modelo *DeepLabCut*, desenvolvido por Mathis et al. [69], surgiu das alterações aplicadas a um modelo já existente denominado *DeeperCut* [73]. A sua arquitetura é baseada na *ResNet* [75], um modelo de *Deep Learning* dedicado à classificação de imagens. A escolha desta arquitetura permitiu a incorporação de um *encoder* pré-treinado para melhorar o desempenho e reduzir o tamanho do *dataset* de treino necessário, um protocolo conhecido como *transfer learning* [76]. No entanto, foi concluído mais tarde que o *transfer learning* não levou a melhorias significativas quando comparado com um modelo com pesos inicializados aleatoriamente. Mais, é preciso ainda ter em conta a extrema parameterização de uma arquitetura pré-treinada que pode não ser vantajosa para a resolução de problemas. Ainda assim, quando bem estruturada, este grande nível de parameterização possibilita a criação de um modelo com previsões bastante acertadas, apesar da sua demora. Para aliviar este efeito de demora, Mathis e Warren [77] mostraram que a rapidez de previsão do *DeepLabCut* [69] pode ser aumentada com a redução das dimensões da imagens de treino, algo que também causará redução da eficácia.

Uma vez que o *DeepLabCut* de Mathis et al. [69] não foi implementado em *Keras*, um requerimento da *framework DeepPoseKit*, surgiu a necessidade, por parte de Graving et al. [68], de adaptar a sua implementação. A versão criada não é exatamente igual à descrita no seu artigo mas é idêntica à exceção dos dados de saída. Em vez de se usar os *location refinement maps* descritos por Insafutdinov et al. [73] e pós-processamento dos *confidence maps* no CPU, a versão de Graving et al. [68] tem uma

camada convolucional transposta para aumentar as dimensões dos resultados em 25% e utiliza o algoritmo de *subpixel maxima*.

O *DeepLabCut* permite ainda a inicialização do modelo com pesos pré-treinados no *dataset ImageNet* [78] e a incorporação de várias *backbones* como *resnet50*, *mobilenetv2*, *densenet121* e *densenet169*, entre outras.

LEAP

Dando preferência à velocidade em vez da eficácia, Pereira et al. [70] implementou uma versão modificada do *SegNet* [79] limitando a complexidade do modelo e a sua sobreparameterização para atingir o seu desejo de maior velocidade na previsão. Esta velocidade de classificação origina limitações como a falta de robustez na variação dos dados, por exemplo a rotação ou variação da luminosidade, e incapacidade de generalização em novos ambientes e configurações experimentais.

A acrescentar a estas dificuldades, a rotina de treino aplicada por Pereira et al. [70] recorre a um pré-processamento muito pesado computacionalmente e, por isso, torna-se impraticável e ineficiente para uma grande quantidade de *datasets* e tarefas.

Para os seus hiperparâmetros foi utilizado $1\times$resolução do *output* com uma *integer-based global maxima* uma vez que Graving et. al [68] pretendia comparar os seus modelos mais complexos com este modelo na sua configuração original.

Comparação entre o DeepLabCut e o LEAP

Um dos grandes debates na área de *Machine Learning* é o *speed-accuracy trade-off* [80], e os dois modelos apresentados anteriormente, são representativos dos dois extremos desta questão. Por um lado Mathis et al. [69] prioriza a precisão sobre a velocidade utilizando modelos sobreparameterizados enquanto que, Pereira et al. [70], prefere exatamente o oposto, escolhendo modelos mais velozes e pequenos e menos robustos.

Este dilema limita as capacidades das redes neuronais convolucionais e, tendo isso em conta, estão a ser desenvolvidos vários trabalhos para tornar estes modelos mais eficientes sem pôr em causa a sua eficácia. Já nas *F-CNNs (Fully Convolutional Neural Networks)*, a utilização de inferências multi-escalares, através do aumento da conectividade entre camadas em várias escalas espaciais, proporciona melhorias na eficiência e robustez. O recurso a inferências multi-escalares capacita o modelo a integrar

informação global de grande escala, como a luminosidade, o plano de fundo, ou a orientação do corpo de um indivíduo, informação de escala intermédia, como a geometria anatómica, e ainda informação local de baixa escala, como as diferenças de cor, textura ou padrões de pele para partes do corpo específicas. Através desta arquitetura multi-escalar, o modelo identifica relações hierárquicas entre os vários espaços escalares e, mais tarde, reúne toda esta informação para criar representações dos *keypoints* ao resolver tarefas de estimação de pose.

3.5.2 Stacked DenseNet e Stacked Hourglass

Com o foco em obter o equilíbrio perfeito entre velocidade e eficácia para equiparar os seus modelos ao de Pereira et al. [70] em termos de velocidade e ao de Mathis et al. [69] em termos de eficácia, Graving et al. [68] implementaram dois modelos de estimação de pose rápida, como já foi dito anteriormente. Para o conseguir, estipularam que as suas implementações teriam de ter menos parâmetros do que o *DeepLabCut* e o *LEAP* e, simultaneamente, teriam de ser capazes de ultrapassar algumas das limitações apresentadas por estas arquiteturas.

Para impedir a sobreparameterização e minimizar a *loss* do desempenho ao mesmo tempo, os modelos foram criados para permitir inferências multi-escalares concorrentemente com a otimização dos hiperparâmetros para a obtenção de maior eficiência.

Stacked DenseNet

Um dos modelos implementados, o *Stacked DenseNet*, é uma nova implementação da *FC-DenseNet* de Jégou et al. [71], organizada numa configuração de *stacks* semelhante à de Newell et al. [64].

Este modelo é composto por uma camada convolucional 7×7 com uma *stride* de 2 para eficientemente reduzir a resolução do *input*, seguindo Newell et al. [64]. De seguida, surge um bloco de redes *densely-connected hourglass* com supervisão intermédia aplicada ao resultado de cada rede. Também foram incluídos hiperparâmetros para as camadas de *bottleneck* e compressão, descritas por Huang et al. [81], para tornar o modelo o mais eficiente possível. Isto consiste na aplicação de convoluções 1×1 para, de uma forma computacionalmente leve, comprimir o número de *feature maps* antes de cada convolução 3×3 assim como quando se aplica redução e aumento das dimensões dos dados [81].

No que toca aos seus hiperparâmetros, neste modelo foi utilizada uma *growth rate* de 48 (48 filtros

por convolução 3×3), um fator de *bottleneck* de 1 e um fator de compressão de 0.5, isto é, os *feature maps* são comprimidos com convoluções 1×1 para $0.5m$ quando se aumenta e diminui as dimensões dos dados, onde m é o número de *feature maps*.

Foram ainda substituídas as usuais configurações de *batch normalization* e função de ativação ReLU por algo mais recente, a função de ativação de auto-normalização *SELU* [82], uma vez que provocava uma redução do tempo de inferência.

O *Stacked DenseNet* oferece ainda a possibilidade de utilizar um *encoder* pré-treinado no *dataset ImageNet* [78].

Stacked Hourglass

O outro modelo, o *Stacked Hourglass*, é uma variação da versão apresentada por Newell et al. [64] do *Stacked Hourglass* e que, através dos hiperparâmetros, permite a variação do número de filtros em cada bloco de convolução para limitar o número de parâmetros, substituindo a forma usual de utilizar 256 filtros para todas as camadas como foi feito por Newell et al. [64].

Relativamente aos seus hiperparâmetros, neste modelo foi utilizado um bloco de 64 filtros por convolução 3×3 com um fator de *bottleneck* de 2.

3.5.3 Grafo de keypoints

Para dimiuir o erro na previsão das localizações dos *keypoints*, os modelos criados por Graving et al. [68] foram construídos de forma que, para além das posições dos *keypoints*, retornassem um grafo hierárquico que descreve a geometria multi-escalar entre os *keypoints*. O *DeepLabCut* e *LEAP* tiveram o seu *output* modificado para que obtivessem os mesmos resultados. Para tal, foi necessário o acrescento de *confidence maps* ao *output* onde as arestas do grafo de postura são representadas por linhas *Gaussian-blurred*.

Assim, o grafo de postura é composto por 4 níveis de representação: um conjunto de *confidence maps* para os menores segmentos de membros no gráfico, como do pé para o tornozelo e do joelho para a cintura, um conjunto de *confidence maps* para cada membro individual, um mapa com o grafo de postura completo e um mapa que combina o grafo de postura com os pontos de confiança para cada um dos *keypoints*. Cada nível deste grafo hierárquico é construído a partir de níveis mais baixos, o que obriga

o modelo a aprender informações correlacionadas através de várias escalas ao fazer previsões. É possível observar os resultados destes mapas nas figuras seguintes.

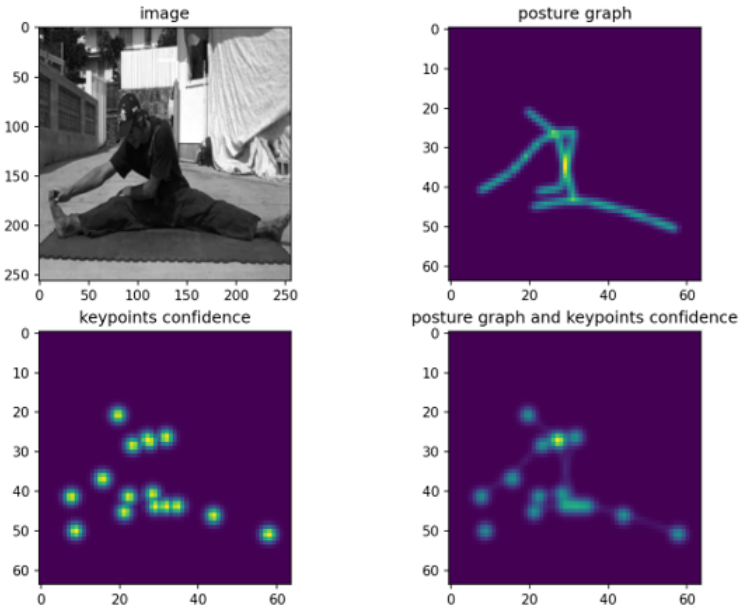


Figura 23: Exemplo dos vários mapas visualizáveis.

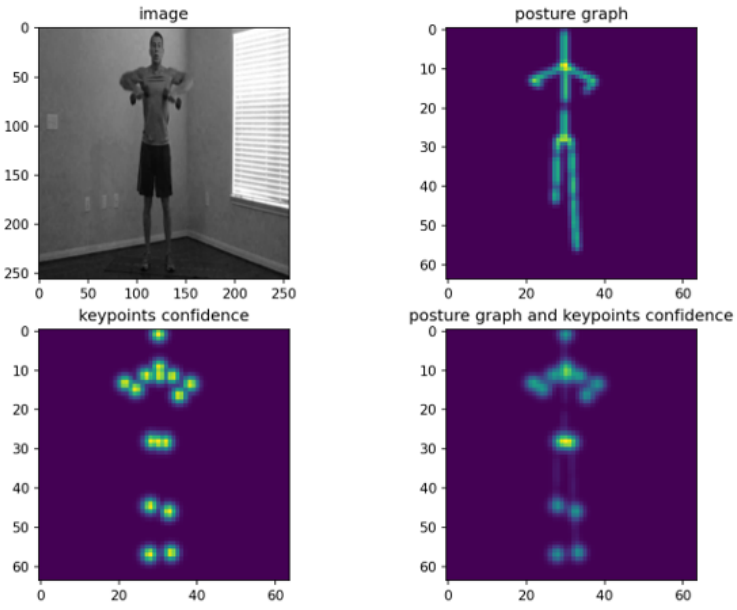


Figura 24: Exemplo dos vários mapas visualizáveis.

4 Implementação do DeepPoseKit em ambiente de fábrica

Tendo em conta o que vimos no capítulo anterior, a ferramenta *DeepPoseKit* será utilizada para resolver a tarefa de estimação de pose proposta nesta dissertação.

Tendo sempre em mente a estimação de pose de humanos em ambiente fabril, como num armazém, a avaliação destes modelos começará por testes utilizando pessoas sentadas numa secretária. Assim, foram obtidos vídeos de colaboradores da empresa Neadvance que, mais tarde, foram divididos em *frames* e se tornaram nas imagens disponíveis para a criação de *datasets*. Para a elaboração destes *datasets*, houve a necessidade de uma escolha meticulosa das imagens para que não surgissem imagens demasiado semelhantes nem repetidas, obtendo assim o melhor treino possível. Para além disto, para as filmagens que eram feitas apenas de um lado da pessoa, foi aplicada uma reflexão para que se obtivessem imagens das duas perspetivas, direita e esquerda, e foi ainda aplicada uma redução a todas as imagens para o tamanho $(320, 448)$ píxeis.

As figuras 25 e 26 ilustram o tipo de imagens usadas para a criação dos *datasets* de treino e teste.

Figura 25: Exemplo de imagem usada para criar os *datasets*.

Figura 26: Exemplo de imagem usada para criar os *datasets*.

4.1 Keypoints

De forma a alcançar o objetivo traçado na subsecção 3.5.3, é necessária a existência de um ficheiro *.csv* que fornecerá informação sobre os *keypoints* escolhidos. Para além de indicar os *keypoints* a ser previstos, neste ficheiro está estipulada uma hierarquia entre eles, isto é, qual é a sua origem ou "pai", e o simétrico de cada um, como por exemplo o simétrico do pulso esquerdo seria o direito.

De notar que a possibilidade de distinguir os *keypoints* entre direito e esquerdo é uma mais valia no reconhecimento das ações do funcionário no chão de fábrica. Esta é uma das características que levou à escolha desta ferramenta.

Com esta estrutura do esqueleto que se pretende construir na imagem, se o modelo estiver em dificuldades para definir um certo *keypoint*, que esteja oculto por exemplo, pode sempre auxiliar-se dos outros *keypoints* para obter uma melhor previsão para o *keypoint* escondido.

Tendo em conta as figuras apresentadas anteriormente, os *keypoints* definidos foram: tórax, ombro esquerdo e direito, cotovelo esquerdo e direito e pulso esquerdo e direito. Assim, segue-se a representação destes *keypoints* e das informações referidas anteriormente na seguinte figura.

	name	parent	swap
0	thorax	NaN	NaN
1	wristR	elbowR	wristL
2	elbowR	shoulderR	elbowL
3	shoulderR	thorax	shoulderL
4	shoulderL	thorax	shoulderR
5	elbowL	shoulderL	elbowR
6	wristL	elbowL	wristR

Figura 27: Ficheiro *.csv* representante da informação sobre a estrutura da pose.

De reparar que o tórax, sendo o ponto central de todo o corpo e sem pontos de referências acima dele para ajudar na sua identificação, será crucial para a estimação do resto dos pontos. A sua má localização pode interferir muito negativamente na deteção dos restantes *keypoints*, assim como a sua correta localização será um grande passo para uma ótima estimação. De um ponto de vista um pouco menos decisivo, posições dos ombros serão importantes para a estimação do cotovelo, e assim sucessivamente para os cotovelos e pulsos.

4.2 Anotações e criação do dataset

Depois de definidos os *keypoints* necessários, o próximo passo é a criação de um *dataset* para treino e outro para teste. Seguindo o protocolo de *Supervised-learning*, é necessária a existência de um *label* para os dados, sendo, neste caso, necessário, termos imagens com um determinado *label*, ou seja, imagens com a localização de cada um dos *keypoints*.

Estes *labels* são conseguidos à custa de anotação de imagens, isto é, através da *GUI* referida na secção 3.5 que o *DeepPoseKit* disponibiliza, um utilizador selecionará manualmente numa imagem as posições de cada um dos *keypoints* definidos no ficheiro *.csv* anterior. É possível observar o *layout* da *GUI* na imagem seguinte (figura 28).

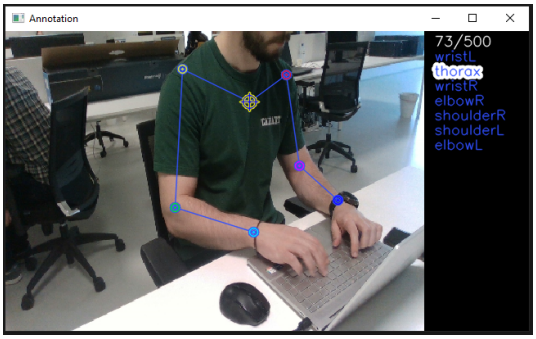


Figura 28: Utilização da *GUI*.

Um dos desafios que surgiu na criação dos *datasets* foi o método de anotação dos *keypoints* não visíveis. Numa primeira fase foi utilizada uma funcionalidade da *GUI* que permitia identificar um certo *keypoint* como oculto. Assim, estes *keypoints* seriam classificados com a posição $(-99999, -99999)$. No entanto, nas primeiras experiências realizadas, todos os *keypoints* estavam a ser previstos exatamente para o mesmo local e, por isso, foi necessário alterar a estratégia. O passo seguinte foi alterar esta funcionalidade para colocar os *keypoints* não visíveis na posição $(-1, -1)$. Apesar da melhoria, os resultados continuaram a não ser bons o suficiente e, portanto, tomou-se a decisão de anotar os *keypoints* tapados nas posições que o utilizador suspeitava estarem.

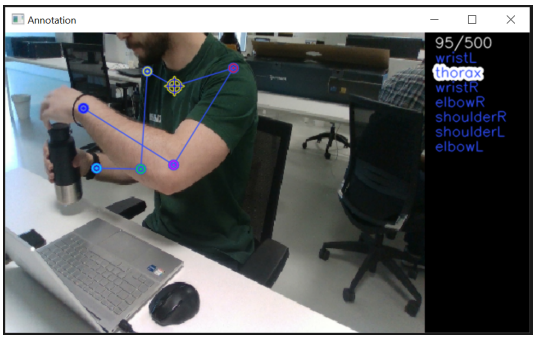


Figura 29: Exemplificação da anotação de um *keypoint* oculto.

Na figura 29 verifica-se que o cotovelo direito não é visível na imagem e, seguindo o pensamento estipulado, anotou-se a sua posição mais provável na imagem.

4.3 Treino

Depois de se obter e organizar os *datasets*, estes serão passados ao modelo para que este possa aprender e estimar corretamente a pose em imagens que não façam parte do dataset de treino.

Na fase de treino de um modelo as escolhas de hiperparâmetros são cruciais, tanto para a parte de aprendizagem, como para a avaliação e análise de resultados deste. Assim, começou por se definir que o número de épocas seria sempre 5000. O *batch size* começou por ser 4, quando se estava a treinar os modelos com *datasets* mais pequenos, mas, com o aumento de dados, este foi alterado para 8. O *optimizer* foi sempre o mesmo, o *adam*, assim como a *learning rate*, que foi sempre 0.001.

A função utilizada para *loss function* foi a *MSE (Mean Squared Error)* ou em português, erro quadrado médio. A *MSE* é a função de *loss* mais comummente usada para regressão. Esta é uma métrica usual para tarefas de estimação de pose.

As métricas usadas para avaliar a qualidade dos resultados previstos foram, além da *MSE*, a *MAE*, a *RMSE* e a distância/norma euclidiana definida por:

$$\sqrt{\sum_{i=1}^{N}(p_i - q_i)^2}. \tag{13}$$

Foi ainda analisada a confiança de previsão dos vários modelos para cada *keypoint*, o número de *keypoints* que seriam anotados se houvesse um determinado valor mínimo de confiança e os valores das

métricas para estes mínimos de confiança, onde apenas os *keypoints* que possuem um valor superior a esse mínimo de confiança são contabilizados. De notar que estas métricas serão a média de cada uma das métricas obtidas entre todos os pontos escolhidos, ou seja, calcula-se uma métrica para todos os pontos individualmente e, depois, calcula-se a média entre eles.

Para auxiliar o treino, foram ainda utilizados alguns *callbacks*. Os *callbacks* são funções que são aplicadas em certas fases do treino para obter estatísticas e estados atuais dos modelos. Assim, foram utilizados o *Early Stopping*, *Reduce LR On Plateau*, *Model Checkpoint* e *TensorBoard*.

O *Early Stopping* é um método de regularização usado para prevenir o *overfitting*. A partir do momento que uma métrica deixa de melhorar, o treino é interrompido. Para além da métrica escolhida, é também definido o número de épocas que se espera sem qualquer melhoria desta métrica. Neste estudo, foram definidas 100 épocas de paciência e a *val_loss* como métrica. De notar que a *val_loss* é o valor da *loss* obtida para os dados de validação, e a *loss* é o valor da função da *loss* para os dados de treino.

O *Reduce LR On Plateau* reduz a *learning rate* quando uma métrica escolhida deixou de melhorar. Para tal, escolhe-se um fator, definido neste caso como 0.2, e, tal como no caso anterior, um número de épocas de espera até executar as alterações, com o valor de 20 neste caso.

O *Model Checkpoint* guarda o modelo, ou apenas os seus pesos, com uma certa frequência. Tem ainda a opção de guardar apenas o melhor modelo até ao momento, o que foi selecionado nas fases de treino dos resultados que mais tarde serão analisados. A classificação de melhor ou pior advém do resultado da métrica escolhida, mais uma vez, a *val_loss*.

Por fim, o *Tensorboard* é uma ferramenta que permite visualizar gráficos sobre a evolução da *loss* e das métricas, histogramas dos pesos, entre outras opções de visualização.

Para além de todas estas mais valias adicionadas ao treino, foi ainda aplicada *data augmentation* às imagens. A *data augmentation* é uma técnica de transformação (rotações, translações, mudanças de cor, mudanças de contraste) dos dados existentes para se obter mais dados de treino e tornar o modelo o mais genérico possível. Assim, foram aplicadas modificações na luminosidade e nos canais de cor em algumas das imagens dos *datasets* de treino.

4.4 Desenvolvimento do dataset e resultados

4.4.1 Fase 1

O primeiro *dataset* de treino desenvolvido era composto por 501 imagens. Dele fizeram parte 4 pessoas, todas sentadas como nas figuras 25 e 26. Para além desta perspetiva da esquerda, incluiu-se também imagens de uma perspetiva mais frontal, como na figura 30, e a partir da direita, através da aplicação da reflexão das imagens.

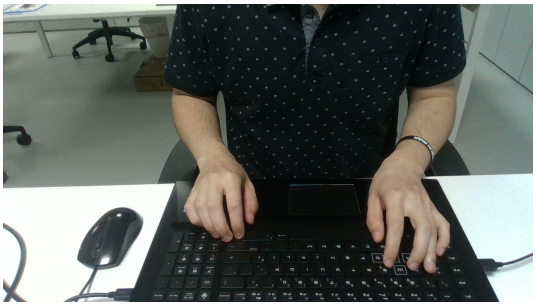


Figura 30: Exemplo de imagem utilizada para treino e teste numa perspetiva frontal.

Este *dataset* serviu apenas para experiências iniciais e, por isso, o único modelo testado foi *DeepLabCut* com *backbone densenet121*.

O seu treino foi avaliado com um *dataset* de teste formado por 169 imagens com 6 pessoas, contendo, assim, 2 pessoas que não estavam presentes no treino.

Tabela 1: Média da confiança do modelo na previsão de cada *keypoint*.

modelo	backbone	thorax	wristR	elbowR	shoulderR	shoulderL	elbowL	wristL
DeepLabCut	densenet121	0,19	0,26	0,22	0,25	0,24	0,26	0,22

Na tabela 1 é possível observar a média da confiança do modelo na previsão de cada *keypoint* neste primeiro teste. Como já era esperado devido à estruturação da hierarquia dos *keypoints*, o tórax foi a parte do corpo que o modelo teve mais dificuldade em encontrar. Por outro lado, os *keypoints* que seriam mais fáceis para o modelo encontrar são o cotovelo esquerdo e o pulso direito.

Nas próximas tabelas 2, 3 e 4 serão apresentados os resultados das métricas. A diferença entre as tabelas é a confiança mínima para os *keypoints* serem contabilizados.

Tabela 2: Valor das métricas contabilizando apenas *keypoints* com confiança ≥ 0.

modelo	backbone	euclidean	mae	mse	rmse	n de kpts
DeepLabCut	densenet121	66,53	41,9	6499,95	47,05	1183

Tabela 3: Valor das métricas contabilizando apenas *keypoints* com confiança ≥ 0.1.

modelo	backbone	euclidean	mae	mse	rmse	n de kpts
DeepLabCut	densenet121	65,11	40,8	6568,89	46,04	814

Tabela 4: Valor das métricas contabilizando apenas *keypoints* com confiança ≥ 0.2.

modelo	backbone	euclidean	mae	mse	rmse	n de kpts
DeepLabCut	densenet121	57,35	35,7	5490,36	40,56	538

Com o aumento da confiança requerida ao modelo, é natural a diminuição do número de *keypoints* a serem considerado. Também existe, de uma forma geral, uma redução dos valores das métricas com a seleção dos *keypoints* que, à partida, estarão mais corretos. Apenas a métrica *MSE* aumenta um pouco o seu valor na passagem da confiança 0 para 0.1.

A métrica da distância euclidiana indicará a distância média à posição correta, ou seja, o valor médio de píxeis em linha reta que as posições previstas estão das posições reais.

São agora apresentadas as previsões deste modelo. As imagens terão dois tipos de anotações: a verde as posições reais dos *keypoints* e, a azul, as posições onde o modelo previu *keypoints* e qual deles.

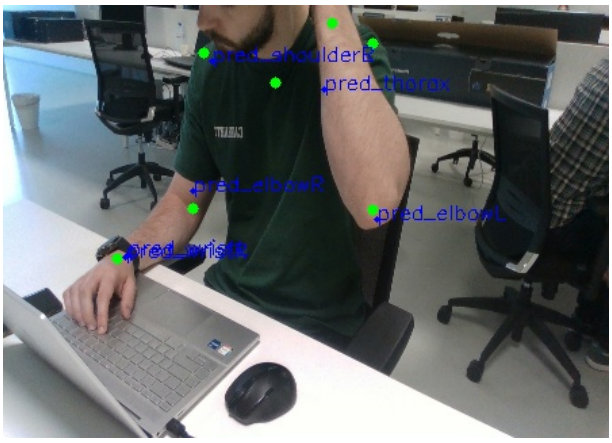


Figura 31: Representação das previsões com confiança superior a 0.

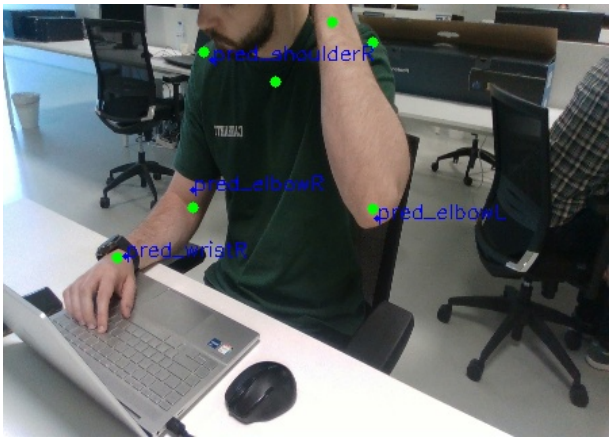


Figura 32: Representação das previsões com confiança superior a 0.1.

Figura 33: Representação das previsões com confiança superior a 0.2.

Na primeira figura, representante das previsões com confiança superior a 0, ou seja, todos os *keypoints* serão anotados, é visível o erro do modelo na localização do pulso esquerdo que foi colocado no mesmo local que o direito. Também os ombros foram colocados no mesmo sítio e o tórax foi colocado bastante ao lado do que era suposto.

Ao passar para a segunda figura, os *keypoints* que não tivessem pelo menos uma confiança de 0.1 não foram desenhados. Assim, sobraram apenas 4, os que, neste caso, estão mais corretamente anotados.

Por fim, na passagem para a seleção dos *keypoints* com confiança superior a 0.2, o cotovelo esquerdo deixou de ser representado. De reparar que, apesar do cotovelo esquerdo parecer melhor anotado do que o direito, que se encontra um bocado acima do que deveria, o modelo tinha menos confiança nele e por isso não foi ilustrado.

4.4.2 Fase 2

Para obter resultados mais precisos, quer nas confianças quer nas métricas, melhorou-se o *dataset* de treino. Foram acrescentadas 311 imagens com mais uma pessoa diferente, completando assim um *dataset* de 812 imagens de 5 pessoas sentadas e das mesmas 3 perspetivas, frontal, do lado esquerdo e do lado direito. O *dataset* de teste utilizado foi o mesmo do primeiro teste.

Com este *dataset* melhorado foram treinados 6 modelos: *DeepLabCut* com as *backbones densenet121*, *resnet50* e *mobilenetv2*, o *Stacked Hourglass*, o *LEAP* e o *Stacked DenseNet*. De

notar que os modelos *DeepLabCut* foram sempre inicializados com os pesos pré-treinados no *dataset ImageNet*.

Este foi o primeiro teste feito para encontrar o melhor modelo possível e, por isso, a experimentação de vários modelos era essencial.

Tabela 5: Média da confiança dos modelos na previsão de cada *keypoint*.

modelo	backbone	thorax	wristR	elbowR	shoulderR	shoulderL	elbowL	wristL
DeepLabCut	densenet121	0,27	0,33	0,31	0,32	0,24	0,27	0,28
DeepLabCut	resnet50	0,01	0,01	0,01	0	0,02	0,01	0,01
DeepLabCut	mobilenetv2	0,1	0,23	0,13	0,12	0,1	0,14	0,08
Stacked Hourglass		0,22	0,33	0,26	0,4	0,32	0,28	0,29
LEAP		0,25	0,38	0,28	0,31	0,3	0,26	0,29
Stacked DenseNet		0,41	0,49	0,48	0,43	0,42	0,43	0,44

A primeira observação a fazer é relativa ao modelo *DeepLabCut* com *resnet50*. Como se pode ver nos valores da tabela 5, e como será confirmado nas tabelas seguintes, o treino deste modelo falhou completamente. Assim, estes dados servem apenas para demonstrar que por vezes os modelos podem simplesmente escolher o caminho errado no seu treino e terminar com resultados bastantes maus. Tendo em conta esta situação, este modelo não será alvo de mais análises nesta fase.

De um ponto de vista mais geral, a confiança dos *keypoints* aumentou em relação ao primeiro teste. O tórax continuou a ser dos *keypoints* que o modelo tem mais dificuldade, ou menos confiança, a identificar, e, no outro extremo, o pulso direito parece ser o mais facilmente assinalável.

Mais concretamente, começando a análise pelo *DeepLabCut* com *densenet121*, as alterações no *dataset* demonstraram claras melhorias em relação ao primeiro teste feito precisamente com este modelo. A confiança em todos os *keypoints* manteve-se ou aumentou.

O *DeepLabCut* com *mobilenetv2* foi o modelo com menos sucesso. Aliás, apresentou piores confianças do que o modelo anterior com o *dataset* mais reduzido.

O *Stacked Hourglass* e o *LEAP* alcançaram valores semelhantes ao *DeepLabCut* com *densenet121*, sendo nalguns casos até melhores. De sublinhar a confiança de 0.4 do *Stacked Hourglass*

para o cotovelo direito e de 0.38 do *LEAP* para o pulso direito.

Por fim, o *Stacked DenseNet* é o modelo mais promissor uma vez que atingiu os maiores valores de confiança para todos os *keypoints*.

Depois desta primeira análise, serão agora explorados os dados relativamente às métricas.

Tabela 6: Valor das métricas contabilizando apenas *keypoints* com confiança ≥ 0.

modelo	backbone	euclidean	mae	mse	rmse	n de kpts
DeepLabCut	densenet121	24,06	15,14	1323,24	17,02	1183
DeepLabCut	resnet50	148,63	92,67	12930,53	105,1	1183
DeepLabCut	mobilenetv2	78,16	49,4	6754,17	55,27	1183
Stacked Hourglass		26,17	16,63	1558,68	18,5	1183
LEAP		46,94	29,66	3776,19	33,19	1183
Stacked DenseNet		15,06	9,65	561,47	10,65	1183

Tabela 7: Valor das métricas contabilizando apenas *keypoints* com confiança ≥ 0.1.

modelo	backbone	euclidean	mae	mse	rmse	n de kpts
DeepLabCut	densenet121	20,35	12,85	1028,57	14,39	963
DeepLabCut	resnet50	148,63	92,67	12930,53	105,1	0
DeepLabCut	mobilenetv2	63,5	40,29	5244,06	44,9	497
Stacked Hourglass		22,48	14,21	1169,71	15,9	972
LEAP		42,72	26,92	3310,2	30,21	924
Stacked DenseNet		13,71	8,80	460,77	9,69	1104

Ao analisar a tabela 6, surgem 3 modelos com valores interessantes para as métricas, o *Stacked DenseNet*, o *Stacked Hourglass* e o *DeepLabCut* com *densenet121*. Algo que surpreendeu foi também a diferença dos valores relativos ao *LEAP* para o *Stacked Hourglass* e o *DeepLabCut* com *densenet121*. Uma vez que apresentava confianças semelhantes aos outros dois, era expectável que as métricas fossem mais equilibradas, como acontece entre os outros dois modelos.

Tabela 8: Valor das métricas contabilizando apenas *keypoints* com confiança ≥ 0.2.

modelo	backbone	euclidean	mae	mse	rmse	n de kpts
DeepLabCut	densenet121	17,64	11,17	836,53	12,47	804
DeepLabCut	resnet50	148,63	92,67	12930,53	105,1	0
DeepLabCut	mobilenetv2	54,89	34,73	4460,98	38,82	270
Stacked Hourglass		20,54	12,9	1001,31	14.52	814
LEAP		38,87	24,19	2971,68	27,49	715
Stacked DenseNet		12,86	8,24	410,89	9,09	1014

Ao avançar para as tabelas 7 e 8, é importante analisar a variação do número de *keypoints* considerados. O *Stacked DenseNet*, devido às suas grandes médias de confiança apresentadas, foi capaz de manter um elevado valor de *keypoints* mesmo quando a confiança mínima era 0.2. O *LEAP*, que apresentava confianças interessantes, apresenta menos 100 *keypoints* que os dois modelos que obtiveram confianças semelhantes a ele, o *Stacked Hourglass* e *DeepLabCut* com *densenet121*, quando se estipula uma confiança superior a 0.2. Isto pode querer dizer que, existe uma grande variação na confiança das suas previsões, isto é, existem pontos que têm confianças muito elevadas, e outros que têm confianças bastante baixas, o que provoca este desaparecimento de anotações e caracteriza este modelo como inconstante, explicando também a sua inferioridade em termos de métricas em relação aos dois referidos anteriormente.

De forma a visualizar os resultados, são agora apresentadas as previsões do melhor e do pior modelo, tendo em conta as métricas, ou seja, a *Stacked DenseNet* e o *DeepLabCut* com *mobilenetv2*.

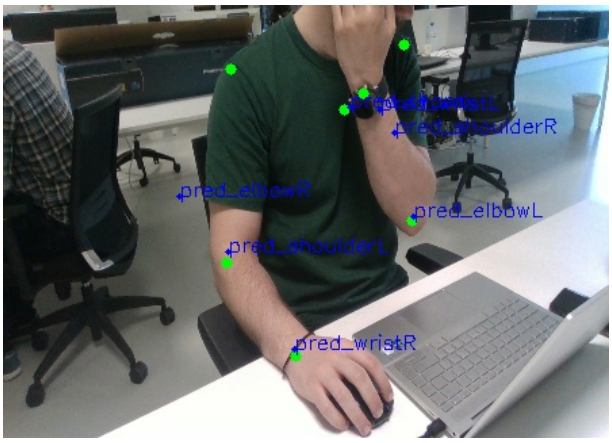


Figura 34: Representação das previsões com confiança superior a 0 do *DeepLabCut* com *mobilenetv2*.

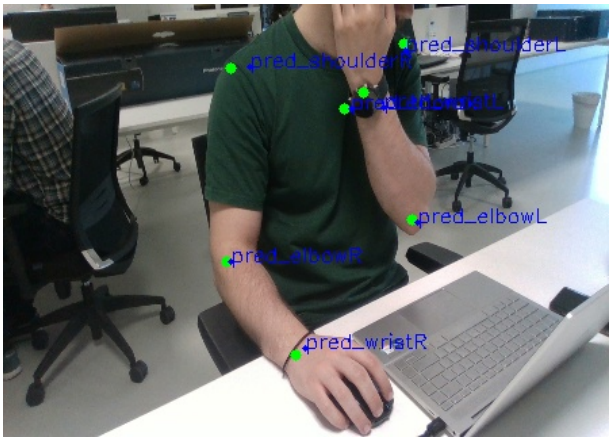


Figura 35: Representação das previsões com confiança superior a 0 do *Stacked DenseNet*.

Nesta primeira representação das previsões, nas figuras 34 e 35, é clara a diferença de acerto nas previsões dos dois modelos. A versão *mobilenetv2* apenas acertou 4 dos 7 *keypoints*, falhando por uma grande margem os restantes 3, enquanto que, a *Stacked DenseNet* obteve excelentes previsões para todos.

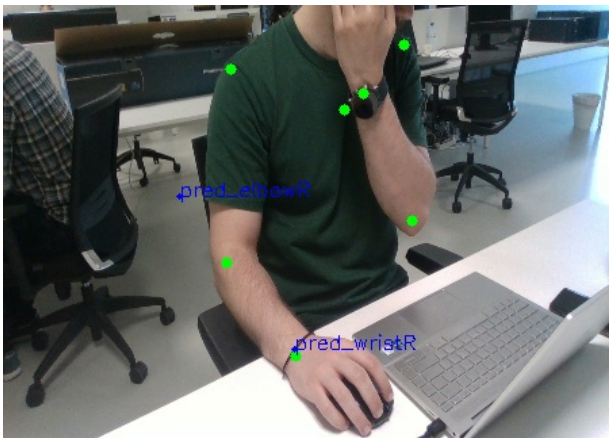


Figura 36: Representação das previsões com confiança superior a 0.1 do *DeepLabCut* com *mobilenetv2*.

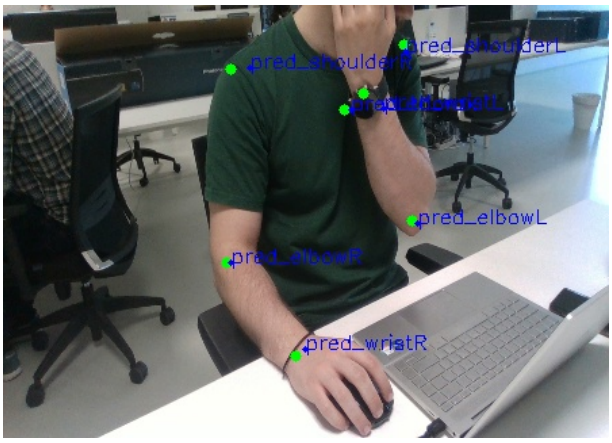


Figura 37: Representação das previsões com confiança superior a 0.1 do *Stacked DenseNet*.

Com o aumento da confiança necessária para serem representados os *keypoints*, rapidamente as representações da *mobilenetv2* desapareceram mantendo apenas 2, e, além das várias perdas, removeu os que estavam corretos e manteve 1 dos que estavam bastante errados. Quase que se pode encarar esta situação como uma questão de sorte nos pontos acertados na figura anterior, tendo em conta isto que sucedeu. Por outro lado, o *Stacked DenseNet* demonstrou a sua robustez e demonstrou que tinha confiança superior a 0.1 para todos os *keypoints*.

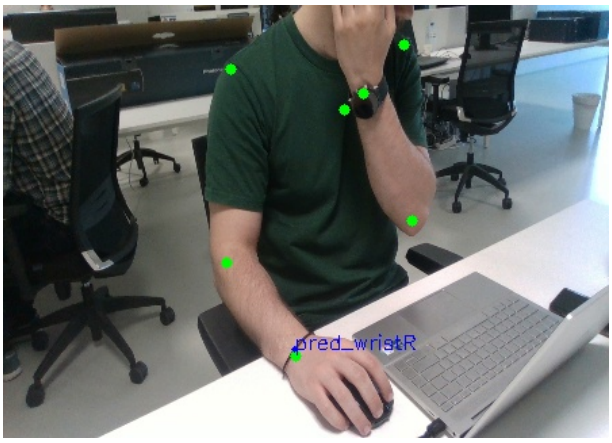


Figura 38: Representação das previsões com confiança superior a 0.2 do *DeepLabCut* com *mobilenetv2*.

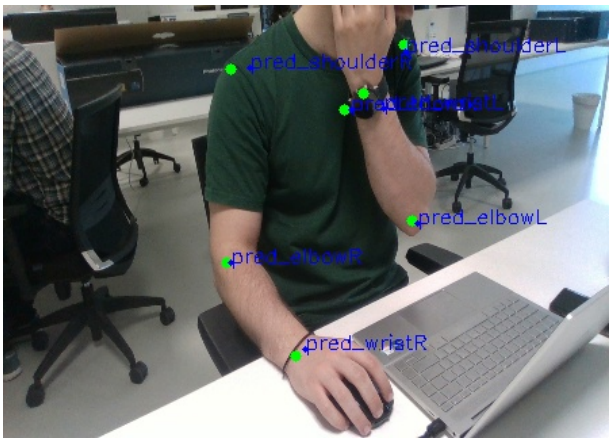


Figura 39: Representação das previsões com confiança superior a 0.2 do *Stacked DenseNet*.

Por fim, com o novo aumento da confiança, o *mobilenetv2* manteve apenas o pulso direito, o que dá a entender que foi o único *keypoint* que, de facto, a sua posição foi intencional. O *Stacked DenseNet* demonstrou o porquê de ser o melhor modelo desta fase ao conseguir ainda manter todos *keypoints* apesar do aumento.

Assim, conclui-se que o *Stacked DenseNet* foi o modelo com os melhores resultados. O *DeepLabCut* com *mobilenetv2* não obteve o desempenho esperado, mas com a evolução do *dataset* pode ser que se adapte melhor.

4.4.3 Fase 3

Para os segundos testes, ao *dataset* de treino foram apenas adicionadas mais 65 imagens das mesmas pessoas mas com roupas diferentes. O *dataset* de teste foi o mesmo, que era composto por 169 imagens. Nesta fase foram testados os modelos: *Stacked DenseNet*, o *LEAP*, o *Stacked Hourglass* e o *DeepLabCut* com as *backbones densenet121*, *resnet50* e *mobilenetv2*

Tabela 9: Média da confiança dos modelos na previsão de cada *keypoint*.

modelo	backbone	thorax	wristR	elbowR	shoulderR	shoulderL	elbowL	wristL
Stacked DenseNet		0,40	0,45	0,42	0,38	0,37	0,40	0,37
DeepLabCut	resnet50	0,33	0,45	0,45	0,40	0,39	0,42	0,41
DeepLabCut	mobilenetv2	0,34	0,49	0,51	0,47	0,41	0,52	0,48
DeepLabCut	densenet121	0,22	0,35	0,40	0,34	0,27	0,36	0,28
LEAP		0,26	0,33	0,41	0,35	0,25	0,33	0,29
Stacked Hourglass		0,27	0,36	0,36	0,31	0,24	0,29	0,28

Com o aumento do número de dados usados, mesmo que por uma pequena quantidade, voltou a melhorar a confiança dos modelos para previsão dos *keypoints*, com a exceção de um caso. De notar a extrema evolução do *DeepLabCut* com *mobilenetv2*, o considerado pior modelo da fase anterior. Por outro lado, e como caso único, o *Stacked DenseNet*, o melhor modelo da fase anterior, diminuiu a confiança dos seus *keypoints*.

É interessante observar que, apesar das dependências dos *keypoints* através do esqueleto hierárquico, o *DeepLabCut* com *mobilenetv2*, que não foi o que obteve maior confiança para o tórax, era o que tinha valores superiores em todos os outros pontos. De notar ainda que o cotovelos parecem ser os *keypoints* que os modelos, por norma, têm mais confiança. Ora, uma vez que é possível utilizar dois pontos como referência, o pulso e ombro, e sabendo que será sempre o ponto intermédio entre estes, percebe-se a facilidade que surge na previsão deste.

Uma situação que se destaca na tabela 10 é o modelo *DeepLabCut* com *densenet121* ser um dos com melhores valores de métricas apesar dos seus valores inferiores de confiança. Sendo os 3 modelos do *DeepLabCut* os com melhores métricas, isto demonstra consistência e robustez do modelo, apesar da

Tabela 10: Valor das métricas contabilizando apenas *keypoints* com confiança ≥ 0.

modelo	backbone	euclidean	mae	mse	rmse	n de kpts
Stacked DenseNet		26,70	16,74	1560,21	18,88	1183
DeepLabCut	resnet50	13,82	8,66	595,75	9,77	1183
DeepLabCut	mobilenetv2	15,39	9,65	702,80	10,89	1183
DeepLabCut	densenet121	15,85	10,10	817,91	11,20	1183
LEAP		39,35	24,57	2942,70	27,83	1183
Stacked Hourglass		20,01	12,97	1028,65	14,15	1183

Tabela 11: Valor das métricas contabilizando apenas *keypoints* com confiança ≥ 0.1.

modelo	backbone	euclidean	mae	mse	rmse	n de kpts
Stacked DenseNet		25,61	16,12	1524,37	18,11	1041
DeepLabCut	resnet50	12,17	7,59	455,53	8,60	1090
DeepLabCut	mobilenetv2	13,11	8,24	482,01	9,27	1081
DeepLabCut	densenet121	13,42	8,62	587,73	9,49	1025
LEAP		35,29	22,18	2632,68	24,95	974
Stacked Hourglass		18,79	12,20	973,77	13,28	1051

Tabela 12: Valor das métricas contabilizando apenas *keypoints* com confiança ≥ 0.2.

modelo	backbone	euclidean	mae	mse	rmse	n de kpts
Stacked DenseNet		24,22	15,28	1426,34	17,13	903
DeepLabCut	resnet50	11,23	7,01	377,46	7,94	959
DeepLabCut	mobilenetv2	11,84	7,47	389,24	8,37	969
DeepLabCut	densenet121	12,26	7,90	497,47	8,67	891
LEAP		30,46	19,27	2183,18	21,54	760
Stacked Hourglass		18,54	12,08	1048,05	13,11	825

variação de *backbones*. Por outro lado, as avaliações do *LEAP* e *Stacked Hourglass* não foram bem conseguidas e podem ser modelos que serão postos de parte em futuros testes. Também os valores das métricas do *Stacked DenseNet* ficaram aquém do esperado, destacando ser um dos piores na distância euclidiana. Ainda sobre este, a pouca perda de *keypoints* com o aumenta da confiança parece indicar que o modelo por vezes tem confianças muito baixas para certos *keypoints* inflacionando muito as suas métricas.

4.4.4 Fase 4

Após os vários testes realizados anteriormente, decidiu-se aplicar uma nova estratégia para tentar melhorar os resultados obtidos. Para ajudar a eliminar o *background* e toda a informação desnecessária sobre a pose que o modelo precisa de obter, foi utilizada uma rede *Yolov4* para obter a identificação de uma pessoa e, a partir da *bounding box* resultante, aplicar um corte à imagem original para que se fique apenas com o conteúdo da *bounding box*. Tal processo é ilustrado de seguida (figuras 40 e 41).

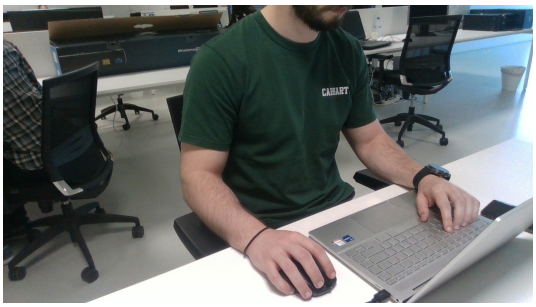


Figura 40: Imagem antes de ser aplicado o *crop*.

Figura 41: Imagem após a aplicação do *crop*.

Foi reconstruido o *dataset* de treino anteriormente utilizado (fase 3). Este novo *dataset* é composto pelas imagens do anterior, depois de aplicado o *crop*, e por mais 149 novas imagens das mesmas pessoas. Estas novas imagens continham novas roupas, diferentes *backgrounds*, uma vez que num escritório existem sempre alterações nas mesas e materiais nas secretárias, e também diferentes luminosidades devido à variação desta com as diferentes estações. Também o *dataset* de teste sofreu as mesmas alterações relativamente ao corte pelas *bounding boxes* mas manteve a sua quantidade.

Tendo em conta os resultados anteriores, e como foi referido anteriormente, a consistência de resultados do *DeepLabCut* foi um dos aspetos que mereceu destaque. Assim, com os novos dados serão testados apenas os modelos *DeepLabCut* com as *backbones densenet121*, *mobilenetv2* e *resnet50*

Será agora feita a análise dos resultados destes modelos com o novo *dataset*.

Tabela 13: Média da confiança dos modelos na previsão de cada *keypoint*.

modelo	backbone	thorax	wristR	elbowR	shoulderR	shoulderL	elbowL	wristL
DeepLabCut	densenet121	0,32	0,38	0,37	0,33	0,31	0,36	0,35
DeepLabCut	mobilenetv2	0,42	0,48	0,46	0,35	0,38	0,38	0,42
DeepLabCut	resnet50	0,26	0,34	0,27	0,25	0,29	0,29	0,27

Tendo em conta a utilização do mesmo modelo e apenas a variação das *backbones*, a referência aos modelos será feita através da *backbone* apenas.

De uma forma geral, os resultados obtidos para a confiança ainda não demonstram uma evolução significativa que justifique as alterações nos *datasets*. O *mobilenetv2* destacou-se claramente, sendo o que tem valores superiores em todos os *keypoints*. Relativamente ao *resnet50*, acredita-se que o baixo valor para o tórax poderá ser a origem do problema para os restantes *keypoints*.

Tabela 14: Valor das métricas contabilizando apenas *keypoints* com confiança ≥ 0.

modelo	backbone	euclidean	mae	mse	rmse	n de kpts
DeepLabCut	densenet121	15,64	9,86	850,13	11,06	1386
DeepLabCut	mobilenetv2	20,99	13,29	1365,32	14,84	1386
DeepLabCut	resnet50	18,13	11,54	1140,59	12,82	1386

Tabela 15: Valor das métricas contabilizando apenas *keypoints* com confiança ≥ 0.1.

modelo	backbone	euclidean	mae	mse	rmse	n de kpts
DeepLabCut	densenet121	14,25	9,02	725,22	10,07	1287
DeepLabCut	mobilenetv2	16,62	10,52	905,21	11,75	1210
DeepLabCut	resnet50	15,59	9,95	863,95	11,02	1244

Tabela 16: Valor das métricas contabilizando apenas *keypoints* com confiança ≥ 0.2.

modelo	backbone	euclidean	mae	mse	rmse	n de kpts
DeepLabCut	densenet121	12,53	7,94	532,54	8,86	1135
DeepLabCut	mobilenetv2	14,22	9,00	681,40	10,06	1071
DeepLabCut	resnet50	14,75	9,47	800,59	10,43	982

O primeiro ponto a destacar é a evolução dos *keypoints* a serem contabilizados. Como foi dito anteriormente, ainda não tinham sido apresentados resultados que tivessem justificado as alterações nos *datasets*. No entanto, a gradual diminuição das previsões que são removidas com o aumento da confiança necessária mostra que, apesar de os modelos não terem uma confiança média superior, estão a

obter valores mais estáveis, ou seja, valores de confiança mais parecidos uns com os outros, um fator muito mais importante do que obter alguns pontos com muita confiança e outros com muito baixa. Ainda sobre este tópico, nota-se a falta de confiança do *resnet50* ao perder 262 *keypoints* da confiança 0.1 para a 0.2. De notar também a rápida descida do valor das métricas do *mobilenetv2* com o aumento da confiança.

Nas previsões surgiu uma situação muito nítida nas representações visuais e que foi comum em todos os modelos. A localização do cotovelo direito na pessoa que não faz parte do *dataset* de treino estava muita vezes errada sendo colocada junto à manga da camisola.

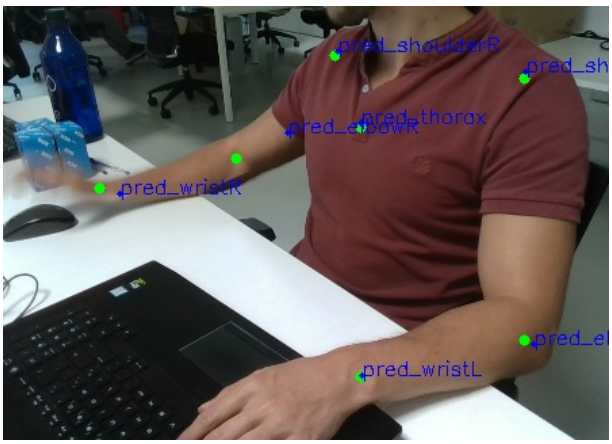


Figura 42: Exemplificação do erro de localização do cotovelo direito.

Na figura 42 é possível verificar a situação descrita anteriormente. Curiosamente, para todas as outras pessoas isto não aconteceu. Uma possível causa será o comprimento da manga utilizada pelo indivíduo, uma vez que as outras pessoas presentes no *dataset*, por norma, registavam comprimentos de manga superiores. Assim, ficou decidido que, posteriormente, seriam necessárias imagens que tivessem pessoas também com mangas compridas para tentar solucionar esta situação.

4.4.5 Fase 5

Após alcançar modelos que pareciam mais estáveis, o próximo passo foi obter imagens que fossem mais relacionadas com o aspeto do ambiente fabril e, por conseguinte, mais apropriadas para a estimação de pose proposta para este trabalho. Assim, foram adquiridas imagens de colaboradores da Neadvance

a construir peças numa porta, usando várias perspetivas. Para colmatar o problema das mangas, foram utilizadas camisas brancas (nalgumas das imagens) que tapassem todo o braço e, assim, levassem o modelo a deixar de associar o cotovelo direito ao limite da manga. Foi ainda aplicado o *crop*, já descrito em fases anteriores, uma vez que levaram a uma evolução na consistência dos modelos.

Figura 43: Exemplo de imagem obtida na construção de peças numa porta.

Na figura 43 estão representadas duas das imagens obtidas. De realçar a variação de luminosidade e das perspetivas para que se obtenha o modelo mais genérico possível.

Ao *dataset* anterior foram adicionadas 482 imagens deste género e originou um novo *dataset* com 1508 imagens. Ao manter os diferentes contextos das imagens é esperado que o modelo seja capaz de reconhecer os vários *keypoints* em qualquer situação e em várias poses que os indivíduos possam estar. Também ao *dataset* de teste foram adicionadas algumas das novas imagens sendo agora composto por 323 imagens.

Para os testes foram utilizados os mesmos modelos: o *DeepLabCut* com as *backbones densenet121*, *mobilenetv2* e *resnet50*.

De um ponto de vista mais geral, os valores das confianças desceram, ou seja, a tentativa de aumentar a capacidade de generalização dos modelos pode ter resultado em maior confusão sobre o que aprender e, por isso, retira confiança ao modelo ao fazer previsões. O *mobilenetv2* apresentou os melhores resultados. Para além disto, equilibrou a confiança na previsão dos dois cotovelos, algo que não aconteceu no teste anterior.

Tabela 17: Média da confiança dos modelos na previsão de cada *keypoint*.

modelo	backbone	thorax	wristR	elbowR	shoulderR	shoulderL	elbowL	wristL
DeepLabCut	densenet121	0,24	0,30	0,25	0,16	0,18	0,22	0,25
DeepLabCut	mobilenetv2	0,36	0,39	0,38	0,29	0,31	0,37	0,35
DeepLabCut	resnet50	0,12	0,13	0,13	0,10	0,19	0,18	0,15

Tabela 18: Valor das métricas contabilizando apenas *keypoints* com confiança ≥ 0.

modelo	backbone	euclidean	mae	mse	rmse	n de kpts
DeepLabCut	densenet121	27,02	16,76	1961,20	19,10	2261
DeepLabCut	mobilenetv2	27,51	17,23	1939,91	19,45	2261
DeepLabCut	resnet50	202,64	130,04	25220,01	143,29	2261

Tabela 19: Valor das métricas contabilizando apenas *keypoints* com confiança ≥ 0.1.

modelo	backbone	euclidean	mae	mse	rmse	n de kpts
DeepLabCut	densenet121	20,59	12,84	1244,70	14,56	1663
DeepLabCut	mobilenetv2	21,06	13,21	1234,65	14,89	1798
DeepLabCut	resnet50	201,66	129,46	25011,00	142,59	2093

Tabela 20: Valor das métricas contabilizando apenas *keypoints* com confiança ≥ 0.2.

modelo	backbone	euclidean	mae	mse	rmse	n de kpts
DeepLabCut	densenet121	17,31	10,83	961,61	12,24	1231
DeepLabCut	mobilenetv2	18,28	11,47	963,95	12,93	1558
DeepLabCut	resnet50	204,37	130,93	25552,94	144,51	107

Com a diminuição das confianças, naturalmente os valores das métricas subiram. Para além disso, quando contabilizados os *keypoints* com confiança superior a 0.2, o *densenet121* e *mobilenetv2* conseguiram manter uma distância euclidiana entre as localizações previstas e as reais relativamente

baixa. É ainda preocupante a quantidade de *keypoints* que são perdidos com o aumento da confiança. Em particular, no último aumento restam apenas 107 no *resnet50*, para além da perda de 1030 pelo *densenet121* e de 703 pelo *mobilenetv2*.

Na fase de anotação das imagens algumas pessoas estavam de costas para a câmara, o que provocou algumas alterações no método de anotação do tórax. Com a tentativa de o anotar mesmo assim, foi necessário colocar a sua posição no meio das costas, o que poderá ter confundido o modelo sobre a sua posição mais correta e, posteriormente, pode ter levado aos resultados menos bem conseguidos apresentados. Esta conclusão foi construida através da análise das seguintes figuras.

Figura 44: Exemplo de imagem obtida de costas.

Figura 45: Exemplo de imagem obtida de costas.

Ainda assim, numa perspetiva mais positiva, o problema do cotovelo parece ter ficado resolvido com o acrescento de imagens de manga comprida, como podemos ver na figura seguinte.

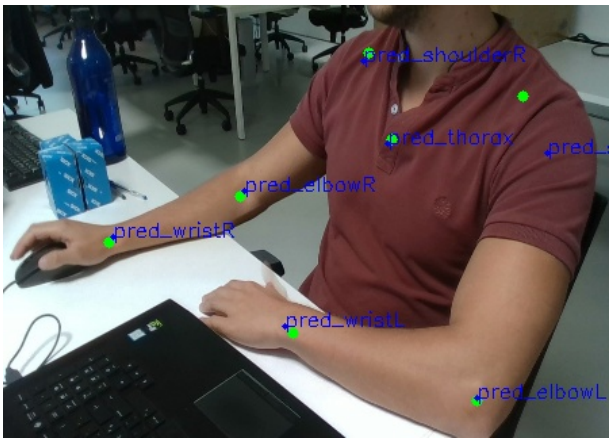


Figura 46: Exemplo de uma imagem depois de aplicada a correção para o cotovelo direito.

4.4.6 Fase 6

Depois da observação sobre o método de anotação do tórax, e considerando a diminuição da qualidade dos resultados apresentados, nesta última fase decidiu-se remover dos *datasets* de treino e teste todas as imagens que tivessem de uma perspetiva de trás. Como foi dito anteriormente, acredita-se que provocavam confusão na anotação do tórax e, por conseguinte, na aprendizagem do modelo. Disto resultou um *dataset* de treino com 1362 imagens e um de teste com 283. Quanto aos modelos, foram novamente avaliados com o objetivo de tentar melhorar os resultados obtidos.

Comecemos então por analisar a confiança obtida com este novo *dataset*.

Tabela 21: Média da confiança dos modelos na previsão de cada *keypoint*.

modelo	backbone	thorax	wristR	elbowR	shoulderR	shoulderL	elbowL	wristL
DeepLabCut	densenet121	0,28	0,29	0,24	0,21	0,20	0,25	0,28
DeepLabCut	mobilenetv2	0,27	0,16	0,23	0,21	0,20	0,19	0,20
DeepLabCut	resnet50	0,22	0,15	0,22	0,07	0,09	0,19	0,07

Os valores da tabela 21 não aparentam de todo indicar que as modificações aplicadas aos *datasets* tenham sido bem sucedidas. Relativamente ao tórax, dois dos modelos subiram as suas confianças e apenas um desceu. Uma vez que o foco era ajudar na previsão do tórax, a tarefa foi parcialmente bem

sucedida.

Já para os outros *keypoints*, o *resnet50* manteve a sua dificuldade na localização do ombro direito e piorou bastante para o esquerdo. O *mobilenetv2*, que diminuiu a confiança do tórax, diminuiu os seus valores para todos os pontos. O *densenet121*, contrariamente aos outros dois modelos, foi o único que de facto aproveitou estas alterações ao subir a sua confiança para vários *keypoints*.

Tabela 22: Valor das métricas contabilizando apenas *keypoints* com confiança ≥ 0.

modelo	backbone	euclidean	mae	mse	rmse	n de kpts
DeepLabCut	densenet121	20,73	13,17	1348,36	14,66	1981
DeepLabCut	mobilenetv2	43,09	27,60	3792,76	30,47	1981
DeepLabCut	resnet50	237,47	151,70	33690,13	167,92	1981

Tabela 23: Valor das métricas contabilizando apenas *keypoints* com confiança ≥ 0.1.

modelo	backbone	euclidean	mae	mse	rmse	n de kpts
DeepLabCut	densenet121	16,40	10,44	863,21	11,60	1586
DeepLabCut	mobilenetv2	30,88	19,75	2376,97	21,84	1280
DeepLabCut	resnet50	235,24	150,74	33138,64	166,34	1216

Tabela 24: Valor das métricas contabilizando apenas *keypoints* com confiança ≥ 0.2.

modelo	backbone	euclidean	mae	mse	rmse	n de kpts
DeepLabCut	densenet121	14,51	9,23	704,59	10,26	1243
DeepLabCut	mobilenetv2	24,76	15,83	1739,92	17,5	890
DeepLabCut	resnet50	237,83	153,43	34126,31	168,17	454

Tal como aconteceu na análise das confianças, também para as métricas a situação se repete, apenas o *densenet121* melhorou os seus valores. Mais, é de notar que, apesar da diminuição de imagens, ou seja, a diminuição de *keypoints* possíveis de anotar, quando analisada a tabela com confianças superiores

a 0.2, mesmo com menos pontos possíveis, o *densenet121* conseguiu manter mais pontos do que na fase anterior. Isto leva a crer que, com estas alterações aos *datasets*, o modelo ficou mais consistente nas suas previsões.

Por outro lado, o *resnet50* obteve valores para a distância euclidiana desastrosos, tal como na fase anterior, mas ainda piorou. Uma vez que estes desempenhos do *resnet50* apenas surgiram após a adição das imagens de montagem da porta, dá a entender que o modelo não foi capaz de generalizar tal variação de ambientes e posições.

Por fim, o *mobilenetv2*, que se tinha destacado pela quantidade de *keypoints* com confiança superior a 0.2 na fase anterior, terminou com valores bastante fracos, principalmente no que tinha sido o seu ponto forte anteriormente (quantidade de pontos anotados com uma confiança relativamente alta).

5 Conclusões e trabalho futuro

5.1 Conclusão

O principal objetivo desta dissertação é a melhoria de processos que ocorrem no chão de fábrica através do estudo e desenvolvimento de soluções inteligentes baseadas em *Machine Learning*, para obter a pose dos funcionários presentes.

Para tal, foi definido um procedimento composto por duas fases, a primeira destas de execução única, e a segunda iterativa e com constantes execuções. Na primeira fase criou-se uma estrutura de esqueleto personalizada adequada ao problema proposto. Na segunda, foram sucessivamente criados *datasets* de treino e teste que eram enviados para 6 modelos diferentes: *DeepLabCut*, com as 3 *backbones densenet121*, *resnet50* e *mobilenetv2*, *LEAP*, *Stacked DenseNet* e *Stacked Hourglass*. Esta fase era iterativa uma vez que, conforme os resultados dos modelos, eram aplicadas alterações aos *datasets*, como a remoção de imagens desnecessárias e adição de imagens em novos ambientes. Para a avaliação dos modelos foram utilizadas 4 métricas, 3 das quais mais conhecidas, *MAE (Mean Average Error)*, *MSE (Mean Squared Error)* e *RMSE (Root Mean Squared Error)*, e a distância euclidiana. Para além disto também se teve em conta a média de confiança dos modelos para cada *keypoint* e o número de *keypoints* que possuíam uma confiança superior aos valores 0, 0.1 e 0.2.

As imagens que compunham os *datasets* eram de funcionários da empresa Neadvance sentados nas suas secretárias e diferentes ângulos destes na montagem de uma peça numa porta.

Numa primeira etapa de todo o processo, o modelo que se destacou foi o *Stacked DenseNet*. Nesta altura os *datasets* eram apenas compostos por imagens dos funcionários sentados.

Ao aumentar o número de imagens, mais precisamente 65 novas imagens, os *DeepLabCuts* demonstraram que produziam os modelos mais robustos, isto é, tinham um maior equilíbrio e consistência nos seus resultados, daí os menores erros nesta fase. Tendo em conta estes resultados, para os testes seguintes, foram apenas testados os 3 *DeepLabCut*.

De forma a ajudar o modelo a aprender apenas o estritamente necessário e evitar aprender características presentes no *background*, foi aplicado um *crop* às imagens de treino e teste. Apesar das confianças obtidas não justificarem a alteração, quando foi analisado o número de *keypoints* concluímos que este

crop compensou o trabalho realizado. Com estas alterações os modelos conseguiram obter confianças ainda mais estáveis.

Para aproximar agora o *dataset* do problema definido como objetivo, foram adicionadas 482 imagens de treino dos funcionários a montar peças numa porta, mantendo também as imagens anteriores. Os resultados obtidos anteriormente pioraram com a adição dos novos dados. Ao analisar as previsões, notou-se que a possível causa para isto foi a existência de imagens com pessoas de costas que obrigava a anotação do tórax, um dos *keypoints* mais importantes, nas costas, o que podia alterar toda a estrutura do esqueleto.

Assim, na última etapa de testes, foram removidas as imagens que se acreditavam estar a causar o problema e obteve-se um *dataset* de treino de 1362 imagens. Apenas o modelo *DeepLabCut* com *densenet121* revelou melhorias nos resultados.

Em suma, pode-se concluir que para os *datasets* com apenas pessoas sentadas conseguiram-se obter resultados bastante interessantes.

Quando foram alterados os ambientes do chão de fábrica, surgiram alguns problemas de deteção. Problema a resolver no futuro.

5.2 Trabalho Futuro

Para um trabalho futuro, existem vários caminhos que podiam ser bastante benéficos para os modelos e para a tarefa em questão, que serão agora enumerados:

- Remoção de imagens de pessoas sentadas nas secretárias e treino e teste apenas com as imagens de montagem de peças na porta;
- Adição de novos *keypoints* facilmente identificáveis para ajudarem na previsão do tórax. Por exemplo, um *keypoint* para o pescoço e outro para a barriga poderiam ser bastante úteis servindo de referência para o tórax. Outro exemplo seria o pescoço e as duas ancas que fornecia ainda mais referência para encontrar o *keypoint* central do esqueleto utilizado nas experiências;
- A adição de imagens de treino é também importante, assim como a variação de indivíduos presentes nessas imagens e as suas roupas, ou seja, fazer treinos com *datasets* mais elaborados.

Referências

[1] https://www.cs.toronto.edu/ lczhang/aps360_20191/lec/w02/terms.html. (consultado em 10/07/2022).

[2] https://www.ibm.com/cloud/learn/neural networks. (consultado em 14/07/2022).

[3] https://wandb.ai/site/articles/fundamentals-of-neural networks. (consultado em 14/07/2022).

[4] https://www.datarobot.com/blog/introduction-to-computer-vision-what-it-is-and-how-it works/. (consultado em 20/07/2022).

[5] https://medium.com/analytics-vidhya/an-introduction-to-image-analysis-using-opencv 2ce8db47e05c. (consultado em 22/07/2022).

[6] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way 3bd2b1164a53. (consultado em 22/07/2022).

[7] https://www.v7labs.com/blog/convolutional-neural-networks guide. (consultado em 22/07/2022).

[8] https://medium.com/low-code-for-advanced-data-science/introduction-to-convolutional-neural-networks-and-computer-vision 72b2d85dd1c0. (consultado em 23/07/2022).

[9] Ce Zheng, Wenhan Wu, Chen Chen, Taojiannan Yang, Sijie Zhu, Ju Shen, Nasser Kehtarnavaz, and Mubarak Shah. Deep learning-based human pose estimation: A survey, 2020.

[10] Tewodros Legesse Munea, Yalew Zelalem Jembre, Halefom Tekle Weldegebriel, Longbiao Chen, Chenxi Huang, and Chenhui Yang. The progress of human pose estimation: A survey and taxonomy of models applied in 2d human pose estimation. *IEEE Access*, 8:133330–133348, 2020.

[11] David H. Hubel and Torsten N. Wiesel. Receptive fields of single neurons in the cat's striate cortex. *Journal of Physiology*, 148:574–591, 1959.

[12] Lawrence Roberts. *Machine Perception of Three-Dimensional Solids*. 01 1963.

[13] S. Papert. *The Summer Vision Project*. AI memo. Massachusetts Institute of Technology, Project MAC, 1966.

[14] David Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. Henry Holt and Co., Inc., New York, NY, USA, 1982.

[15] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[16] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[17] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[18] Jianbo Shi and Jitendra Malik. Normalized cut and image segmentation. Technical Report UCB/CSD-97-940, EECS Department, University of California, Berkeley, May 1997.

[19] D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, 1999.

[20] Paul Viola and Michael Jones. Robust real-time object detection. volume 57, 01 2001.

[21] M. Everingham, A. Zisserman, C. K. I. Williams, and L. Van Gool. The PASCAL Visual Object Classes Challenge 2006 (VOC2006) Results. http://www.pascal-network.org/challenges/VOC/voc2006/results.pdf.

[22] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[23] M. Fischler and R. Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on Computers*, 22(01):67–92, jan 1973.

[24] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[26] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[27] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014.

[28] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style, 2015.

[29] Aditya M. Deshpande, Anil Kumar Telikicherla, Vinay Jakkali, David A. Wickelhaus, Manish Kumar, and Sam Anand. Computer vision toolkit for non-invasive monitoring of factory floor artifacts. *Procedia Manufacturing*, 48:1020–1028, 2020. 48th SME North American Manufacturing Research Conference, NAMRC 48.

[30] Ray Y. Zhong, Xun Xu, Eberhard Klotz, and Stephen T. Newman. Intelligent manufacturing in the context of industry 4.0: A review. *Engineering*, 3(5):616–630, 2017.

[31] Seung Hyun Kim, Su Chang Lim, and Do Yeon Kim. Intelligent intrusion detection system featuring a virtual fence, active intruder detection, classification, tracking, and action recognition. *Annals of Nuclear Energy*, 112:845–855, 2018.

[32] A. Murynin-V. Kunzetsov-P. Ivanov M. Jin, L. Jin and I.-J. Jeong. *Method of intruder detection and device thereof - US 7,088,243 B2 - PatentSwarm*. 2006.

[33] G. Aravamuthan, P. Rajasekhar, R. Verma, S. Shrikhande, Somitra Kar, and Suresh Babu. *Physical Intrusion Detection System Using Stereo Video Analytics*, pages 173–182. 01 2020.

[34] Ashutosh Dekhne, Greg Hastings, John Murnane, and Florian Neuhaus. Automation in logistics: Big opportunity, bigger uncertainty. *McKinsey Q*, pages 1–12, 2019.

[35] John Howard. Artificial intelligence: Implications for the future of work. *American Journal of Industrial Medicine*, 62(11):917–926, 2019.

[36] Deloitte. The warehouse of tomorrow, today ubiquitous computing, internet of things, machine learning. Technical report, 2017.

[37] Cyril Alias, Çağdaş Özgür, and Bernd Noche. Monitoring production and logistics processes with the help of industrial image processing. In *27th Annual POMS Conference*, pages 2280–2292, 2016.

[38] Chu Wang, Marcello Pelillo, and Kaleem Siddiqi. Dominant set clustering and pooling for multi-view 3d object recognition, 2019.

[39] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34, 2018.

[40] Kimberly Fessel. 5 significant object detection challenges and solutions. 2019.

[41] Andriy Burkov. *The Hundred-Page Machine Learning Book*.

[42] Ethem Alpaydin. *Introduction to Machine Learnng - Second Edition*. The MIT Press.

[43] Daniel Kirsch Judith Hurwitz. *Machine Learning for dummies*.

[44] CRC Press Taylor Francis Group. *Explorations in Artificial Intelligence and Machine Learning*.

[45] Mu Li Alexander J. Smola Aston Zhang, Zachary C. Lipton. *Dive into Deep Learning*.

[46] Y Reddy, Viswanath Pulabaigari, and Eswara B. Semi-supervised learning: a brief review. *International Journal of Engineering Technology*, 7:81, 02 2018.

[47] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997.

[48] Avner Friedman, Alla Borisyuk, Bard Ermentrout, and David Terman. *Introduction to Neurons*, volume 1860, pages 1–20. 01 2005.

[49] Akash Kumar Bhoi, Pradeep Kumar Mallick, Chuan-Ming Liu, and Valentina E. Balas, editors. *Bio-inspired Neurocomputing*. Springer Singapore, 2021.

[50] Roza Dastres and Mohsen Soori. Artificial neural network systems. *International Journal of Imaging and Robotics*, 21:13–25, 03 2021.

[51] Alexei Botchkarev. A new typology design of performance metrics to measure errors in machine learning regression algorithms. *Interdisciplinary Journal of Information, Knowledge, and Management*, 14:045–076, 2019.

[52] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[53] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[54] Ranjay Krishna. Computer vision: Foundations and applications. 2017.

[55] Anirudha Ghosh, A. Sufian, Farhana Sultana, Amlan Chakrabarti, and Debashis De. *Fundamental Concepts of Convolutional Neural Network*, pages 519–567. 01 2020.

[56] Snehanshu Saha, Nithin Nagaraj, Archana Mathur, Rahul Yedida, and Sneha H R. Evolution of novel activation functions in neural network training for astronomy data: habitability classification of exoplanets. *The European Physical Journal Special Topics*, 229(16):2629–2738, nov 2020.

[57] Bin Wang Wenqing Zhng Qi Dang, Jianqin Yin. Deep learning based 2d human pose estimation: A survey. 24(6), 2019.

[58] Alexander Toshev and Christian Szegedy. Deeppose: Human pose estimation via deep neural networks. *CoRR*, abs/1312.4659, 2013.

[59] Benjamin Sapp and Ben Taskar. Modec: Multimodal decomposable models for human pose estimation. In *In Proc. CVPR*, 2013.

[60] Sam Johnson and Mark Everingham. Clustered pose and nonlinear appearance models for human pose estimation. In *bmvc*, volume 2, page 5, 2010.

[61] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christopher Bregler. Efficient object localization using convolutional networks, 2014.

[62] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2d human pose estimation: New benchmark and state of the art analysis, June 2014.

[63] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines, 2016.

[64] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation, 2016.

[65] Joao Carreira, Pulkit Agrawal, Katerina Fragkiadaki, and Jitendra Malik. Human pose estimation with iterative error feedback, 2015.

[66] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1302–1310, 2017.

[67] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Doll'a r, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[68] Jacob M Graving, Daniel Chae, Hemal Naik, Liang Li, Benjamin Koger, Blair R Costelloe, and Iain D Couzin. Deepposekit, a software toolkit for fast and robust animal pose estimation using deep learning. *eLife*, 8:e47994, 2019.

[69] Alexander Mathis, Pranav Mamidanna, Kevin M. Cury, Taiga Abe, Venkatesh N. Murthy, M. Mathis, and Matthias Bethge. Deeplabcut: markerless pose estimation of user-defined body parts with deep learning. *Nature Neuroscience*, 21:1281–1289, 2018.

[70] T.D. Pereira, D. E. Aldarondo, L. Willmore, M. Kislin, S. S.-H. Wang, M. Murthy, and J. W. Shaevitz. Fast animal pose estimation using deep neural networks. *bioRxiv*, 2018.

[71] Simon Jégou, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 11–19, 2017.

[72] Manuel Guizar-Sicairos, Samuel T. Thurman, and James R. Fienup. Efficient subpixel image registration algorithms. *Opt. Lett.*, 33(2):156–158, Jan 2008.

[73] Eldar Insafutdinov, Leonid Pishchulin, Bjoern Andres, Mykhaylo Andriluka, and Bernt Schiele. Deepercut: A deeper, stronger, and faster multi-person pose estimation model, 2016.

[74] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, 2017.

[75] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[76] L. Y. Pratt. Discriminability-based transfer between neural networks. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992.

[77] Alexander Mathis and Richard Warren. On the inference speed and video-compression robustness of deeplabcut. *bioRxiv*, 2018.

[78] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[79] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation, 2015.

[80] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3296–3297, 2017.

[81] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[82] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.