UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

IGOR MUNIZ SOARES

Uma abordagem bottom-up completa para reconhecimento de atividades humanas em imagens através da pose estimada com redes convolucionais

Goiânia
2019

sistema de bibliotecas ufg

PRPG
PRÓ-REITORIA DE PÓS-GRADUAÇÃO

UFG

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TESES E DISSERTAÇÕES NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei nº 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou *download*, a título de divulgação da produção científica brasileira, a partir desta data.

1. Identificação do material bibliográfico: **[X] Dissertação** **[ ] Tese**

2. Identificação da Tese ou Dissertação:

Nome completo do autor: **Igor Muniz Soares**

Título do trabalho: **Uma abordagem bottom-up completa para reconhecimento de atividades humanas em imagens através da pose estimada com redes convolucionais**

3. Informações de acesso ao documento:

Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF da tese ou dissertação.

Igor Muniz Soares

Assinatura do(a) autor(a)[2]

Ciente e de acordo:

Assinatura do(a) orientador(a)[2]

Data: 19 / 11 / 19

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Os dados do documento não serão disponibilizados durante o período de embargo.

Casos de embargo:

- Solicitação de registro de patente;
- Submissão de artigo em revista científica;
- Publicação como capítulo de livro;
- Publicação da dissertação/tese em livro.

[2] A assinatura deve ser escaneada.

IGOR MUNIZ SOARES

Uma abordagem bottom-up completa para reconhecimento de atividades humanas em imagens através da pose estimada com redes convolucionais

Dissertação apresentada à Escola de Engenharia Elétrica, Mecânica e de Computação da Universidade Federal de Goiás para obtenção do título de Mestre em Ciências pelo Programa de Pós-graduação em Engenharia Elétrica e de Computação.

Orientador: Prof. Dr. Gelson da Cruz Júnior

Coorientador: Prof. Dr. Cássio Vinhal

Goiânia

2019

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Muniz Soares, Igor
Uma abordagem bottom-up completa para reconhecimento de atividades humanas em imagens através da pose estimada com redes convolucionais [manuscrito] / Igor Muniz Soares. - 2019.
127 f.: il.

Orientador: Prof. Dr. Gelson da Cruz Júnior; co-orientador Dr. Cássio Dener Noronha Vinhal.
Dissertação (Mestrado) - Universidade Federal de Goiás, Escola de Engenharia Elétrica, Mecânica e de Computação (EMC), Programa de Pós-Graduação em Engenharia Elétrica e de Computação, Goiânia, 2019.
Bibliografia. Anexos. Apêndice.
Inclui siglas, gráfico, tabelas, algoritmos, lista de figuras, lista de tabelas.

1. Visão Computacional. 2. Aprendizagem Profunda. 3. Reconhecimento de Atividade em Imagens. 4. Estimação de Pose. 5. Detecção de Pessoas. I. da Cruz Júnior, Gelson, orient. II. Título.

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO
COORDENAÇÃO DE PÓS-GRADUAÇÃO

Ata de Dissertação de Mestrado

Ata da sessão de julgamento da Dissertação de Mestrado em Engenharia Elétrica e de Computação, área de concentração Engenharia de Computação, do candidato **Igor Muniz Soares,** realizada em 27 de setembro de 2019.

Aos vinte e sete dias do mês de setembro de dois mil e dezenove, às 14:30 horas, na na Sala Caryocar Brasiliensis, Bloco A da Escola de Engenharia Elétrica e de Computação (EMC), Universidade Federal de Goiás (UFG), reuniram-se os seguintes membros da Comissão Examinadora designada pela Coordenadoria do Programa de Pós-graduação em Engenharia Elétrica e de Computação: Os Doutores , Gelson da Cruz Junior – Orientador (EMC/UFG), Deborah Silva Alves Fernandes – INF/UFG, Wesley Pacheco Calixto - EMC/UFG e Cássio Dener Noronha Vinhal – EMC/UFG, para julgar a Dissertação de Mestrado de **Igor Muniz Soares,** intitulada **"Uma Abordagem Bottom-up Completa Para Reconhecimento De Atividades Humanas Em Imagens Através Da Pose Estimada Com Redes Convolucionais"**, apresentada pelo Candidato como parte dos requisitos necessários à obtenção do grau de Mestre, em conformidade com a regulamentação em vigor. O Professor Doutor Gelson da Cruz Junior da Comissão, abriu a sessão e apresentou o candidato que discorreu sobre seu trabalho, após o que, foi arguido pelos membros da Comissão na seguinte ordem: Deborah Silva Alves Fernandes, Wesley Pacheco Calixto e Cássio Dener Noronha Vinhal. A parte pública da sessão foi então encerrada e a Comissão Examinadora reuniu-se em sessão reservada para deliberar. A Comissão julgou então que o Candidato, tendo demonstrado conhecimento suficiente, capacidade de sistematização e argumentação sobre o tema de sua Dissertação, foi considerado **aprovado** e deve satisfazer as exigências listadas na Folha de Modificação de Dissertação de Mestrado, em anexo a esta Ata, no prazo máximo de 60 dias, ficando o professor orientador responsável por atestar o cumprimento dessas exigências. Os membros da Comissão Examinadora descreveram as justificativas para tal avaliação em suas respectivas Folhas de Avaliação, anexas a esta Ata. Nada mais havendo a tratar, o presidente da Comissão declarou encerrada a sessão. Nos termos do Regulamento Geral dos Cursos de Pós-graduação desta Universidade, a presente Ata foi lavrada, lida e, julgada conforme, segue assinada pelos membros da Comissão supracitados e pelo Candidato. Goiânia, 27 de setembro de 2019.

Comissão Examinadora Designada:

Gelson da Cruz Junior – Orientador (EMC/UFG) (Avaliação: APROVADO)

Deborah Silva Alves Fernandes – INF/UFG (Avaliação: Aprovado)

Wesley Pacheco Calixto – EMC/UFG (Avaliação: APROVADO)

Cássio Dener Noronha Vinhal – EMC/UFG (Avaliação: Aprovado)

Candidato:

Igor Muniz Soares

Resumo

SOARES, Igor Muniz. **Uma abordagem bottom-up completa para reconhecimento de atividades humanas em imagens através da pose estimada com redes convolucionais** 2019. 125 f. Dissertação (Mestrado) – Escola de Engenharia Elétrica, Mecânica e de Computação, Universidade Federal de Goiás, Goiânia, 2019.

Nos últimos anos, houveram significantes melhorias no campo da visão computacional, tornando-se possível obter importantes informações de imagens. Alguns dos desafios para melhor entendimento de uma cena são a detecção de pessoas e reconhecimento das atividades que eles estão executando. Este trabalho propõe um único modelo ponto-a-ponto capaz de detectar pessoas, estimar as poses destas, e reconhecer a atividade que eles estão executando através da pose. Os experimentos mostram que nosso modelo alcançou o estado da arte nas tarefas de detecção de pessoa e estimação de pose no MSCOCO dataset 2017, e é capaz de reconhecer as atividades andando, correndo, sentado e parado com um F1 score de 0.7344. O modelo é executado em tempo real com uma inferência de aproximadamente 20 frames/segundo.

Palavras-chaves: estimação de pose; reconhecimento de atividades; detecção de pessoas; redes neurais convolucionais.

Abstract

SOARES, Igor Muniz. **A complete bottom-up approach to recognizing human activities in images through the estimated pose with convolutional networks.** 2019. 125 p. Dissertation (Master of Science) – School of Electrical, Mechanical, and Computer Engineering, University Federal of Goiás, Goiânia, 2019.

In the last few years, significant improvements in the computer vision were made, making it possible to obtain important information from images. Some of the challenges for a better understanding of a scene are the detection of people and the recognition of the activities they are performing. This work propose a single end-to-end model able to detect people, estimate their pose, and recognize each one of their activities by their pose. The experiments show that the model has reached the state of the art in the tasks of person detection and pose estimation on MSCOCO Dataset 2017, and can recognize walking, running, sitting, and standing activities with an F1 score of 0.7344. The model is real-time with an inference rate of approximately 20 frames/sec.

Keywords: multi-person pose estimation; human activity recognition; person detection; convolutional neural networks.

Lista de figuras

Lista de algoritmos

Lista de tabelas

Lista de abreviaturas e siglas

AP	Average Precision (Precisão média)
CNN	Convolutional Neural Networks (Redes Neurais Convolucionais)
FPN	Feature Pyramid Network (Rede de pirâmides de característica)
FPS	Frames por segundo
GB	Gigabyte
GHz	Giga hertz
IoU	Intersection over Union (Intersecção sobre a união)
mAP	mean Average Precision (média da precisão média) [M] Milhão [ms] milissegundos
MLP	Multi Layer Perceptron (Perceptron de múltiplas camadas)
OKS	Object Keypoint Similarity (Similaridade de objetos ponto chave)
PRN	Pose Residual Network (Rede Residual de Poses)
ResNet	Residual Networks (Redes Residuais)
RNA	Redes neurais artificiais
s	segundo
SSD	Solid State Drive

Sumário

[3] De acordo com a Associação Brasileira de Normas Técnicas. NBR 6023.

1 Introdução

Com avanços no campo de aprendizagem profunda, o termo inteligência artificial ganhou força nos últimos anos, trazendo produtos inovadores como carros autônomos e sistemas de reconhecimento facial, e consequentemente mais investimento para pesquisas nessa área.

No entanto, a ideia de inteligência demonstrada por máquinas em contraste a inteligência humana (RUSSELL; NORVIG, 2009) não é algo novo, mas sim datada de 1956 quando em um *workshop* em *Dartmouth College* surge o conceito de inteligência das máquinas e se inicia as pesquisas na área. Em seguida, pesquisadores percebem a necessidade de máquinas compreenderem imagens e vídeos, passo necessário para continuar desenvolvendo inteligência em máquinas, e assim o campo da visão computacional se expande.

Na década de 1970, apesar das grandes expectativas em redes neurais artificiais, a limitação de *hardware* para treinamento dos algoritmos e a dificuldade de aplicação em problemas não lineares fizeram com que as pesquisas em inteligência artificial diminuíssem consideravelmente. Em 1989, LeCun e colaboradores propuseram uma solução para reconhecimento de caracteres escritos à mão aplicando redes neurais com camadas convolucionais para extração de características (LECUN; BOSER; HENDERSON, 1989). Mas só em 2012, Krizhevsky e colaboradores chamaram a atenção da comunidade científica ao obter resultados consideravelmente melhores em classificação de imagens utilizando redes neurais convolucionais profundas (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

Desde então, problemas de visão computacional frequentemente empregam redes convolucionais, e graças a melhoria nas tecnologias de *hardware* e especialmente o uso de unidades gráficas de processamento (GPUs) para treinamento de algoritmos, arquiteturas de redes profundas alcançaram o estado da arte nos principais *datasets* de problemas da visão computacional (LIN; ZITNICK; DOLL, 2014; EVERINGHAM *et al.*, 2010; GEIGER *et al.*, 2013; DENG *et al.*, 2009) para classificação de imagens (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), detecção de objetos (ZHAO; ZHENG, 2019) e segmentação de cenas (GARCIA-GARCIA *et al.*, 2017).

Combinando os resultados de problemas isolados, trabalhos mais complexos surgiram com um objetivo mais ambicioso: a compreensão de uma cena em um dado contexto. Tal

compreensão é fundamental para uma melhor interação homem-máquina em campos como robótica e carros autônomos. Uma das principais tarefas para contextualizar uma cena é reconhecer a atividade de humanos, o que é hoje em dia uma das áreas mais pesquisadas em visão computacional, dado o grande número de aplicações como monitoramento e segurança, interação homem-máquina, veículos autônomos, compreensão de vídeos e imagens, e tomada de decisões em sistemas inteligentes (POPPE, 2010).

Este trabalho apresenta um estudo para reconhecimento de atividades de pessoas em imagens pela pose estimada de cada pessoa. Para isso, é necessário primeiro estimar a pose de cada pessoa presente na cena, um problema que pode ser dividido em outros dois: a) detecção de pontos chaves (*keypoints*) ou juntas (ombros, joelhos, cotovelos, mãos...) e b) agrupamento destes pontos para cada indivíduo, uma tarefa não trivial quando múltiplas pessoas estão presentes em cena. A seguir são apresentados os trabalhos mais recentes em cada uma dessas tarefas.

1.1 Revisão Bibliográfica

1.1.1 Estimação da Pose Humana

Para estimar a pose de uma pessoa, é preciso localizar certas partes do corpo humano. Os primeiros métodos focavam em obter a pose de uma única pessoa em um *frame*, como realizado por Toshev et al. aplicando redes neurais profundas como regressores das coordenadas de cada parte do corpo (TOSHEV, 2014).

O desafio de obter a pose de múltiplas pessoas apareceu como um problema mais complexo dado a necessidade de se associar os pontos detectados a cada pessoa presente na cena. Em um *frame*, existe um número desconhecido de pessoas, em posições e escalas não definidas, capazes de interagir entre si ou com partes do corpo escondidas.

Os primeiros trabalhos trataram essa dificuldade primeiro detectando todas pessoas em cena, um problema já resolvido com aprendizagem profunda, e em seguida, reduzindo o problema a estimação de pose de uma única pessoa para cada uma das detectadas. Esta metodologia ficou conhecida como *top-down*.

Métodos *top-down* tem seu tempo de execução proporcional ao número de pessoas detectadas, tornando-se extremamente lentos em cenas comuns (ruas, parques, etc.), e dependem de um bom detector de objetos, dado o fato de que pessoas não detectadas não

passam para o próximo passo de terem suas poses reconhecidas. Entretanto, este método consegue analisar cada instância de pessoa por vez, permitindo melhores refinamentos para a localização dos *keypoints*. Tais métodos possuem os melhores resultados na *MS COCO Keypoints Challenge* (LIN; ZITNICK; DOLL, 2014) [1].

He e colaboradores desenvolveram a Mask R-CNN, uma rede profunda para detecção de objetos baseado em máscaras que também pode ser estendida para detectar *keypoints*, onde K juntas possuem K máscaras a serem preditas (HE *et al.*, 2017). Xiao e colaboradores (XIAO; WU; WEI, 2018) propuseram um método simples adicionando camadas deconvolucionais à ResNet, uma rede profunda para classificação de imagens (HE *et al.*, 2015), alcançando melhores resultados. Fang e colaboradores usando *Symmetric Spatial Transformer Network* (SSTN) propuseram um *framework* capaz de lidar com detecção de pessoas de baixa acurácia, melhorando também a estimação de pose de métodos *top-down* (FANG *et al.*, 2017).

Dada a limitação de tempo de inferência dos métodos *top-down*, pesquisadores focaram em obter algoritmos capazes de estimar as coordenadas de todas as partes de corpo humano de uma única vez e então associá-las a cada instância de pessoa presente na imagem. Chamado de métodos *bottom-up*, eles são consideravelmente mais rápidos que os métodos *top-down*, porém apresentam mais dificuldades em detectar os *keypoints* e associá-los de maneira correta, apresentando acurácias um pouco menores.

Nesta metodologia *bottom-up*, Cao e colaboradores apresentaram o primeiro sistema no estado da arte e *real time* para estimação de poses de múltiplas pessoas utilizando *Part Affinity Fields* para associar os *keypoints* (CAO *et al.*, 2017). Li e colaboradores combinaram o método *bottom-up* com o detector de pessoas do método *top-down* (LI *et al.*, 2018), usando as regiões delimitadas pelo regressor do detector de cada pessoa como uma restrição para associar os *keypoints* a cada pessoa. Kocabas e colaboradores propuseram uma rede residual capaz de combinar as coordenadas das partes de corpo humano com as regiões de restrição do detector para predizer poses usando uma função de aprendizagem previamente treinada com poses conhecidas (KOCABAS; KARAGOZ; AKBAS, 2018).

[1] Os resultados podem ser visualizados em http://cocodataset.org/keypoints-leaderboard

1.1.2 Reconhecimento de Atividade Humana

Reconhecer atividades em imagens é uma das principais áreas de pesquisas atuais da visão computacional, não limitada apenas em identificar que ação as pessoas estão realizando, mas também detectar objetos próximos e a relação entre todos em cena, classificando atividades mais complexas.

Os primeiros métodos propuseram reconhecer atividades em imagens e vídeos extraindo características de cada *frame* e classificando-as com técnicas de reconhecimento de padrões (POPPE, 2010; CHEN *et al.*, 2018). Logo se percebeu a necessidade de informações no tempo e trabalhos com camadas convolucionais 3D ou redes recorrentes com memória (LSTM) melhoraram os resultados obtendo características espaço-temporais (CARREIRA; ZISSERMAN, 2017; JI *et al.*, 2013). Qiu e colaboradores propuseram uma pseudo 3D rede residual para aprender as características espaço-temporais (QIU; YAO; MEI, 2017) e Tran e colaboradores fizeram um estudo detalhado em (TRAN *et al.*, 2018) sobre as diferenças entre redes 2D e 3D para reconhecimento de atividades, propondo um novo bloco convolucional chamado de R(2+1)D.

Entretanto, reconhecer a atividade de uma cena extraindo características como um todo, não permite analisar diferentes atividades em uma mesma cena ou nem mesmo aprofundar em detalhes da atividade de cada pessoa. Dado a representatividade da pose em determinadas atividades humanas, trabalhos surgiram com reconhecimento de ações através da pose (WANG; WANG; YUILLE, 2013; YANG; WANG; MORI, 2010). Evangelidis e colaboradores desenvolveram um codificador para representar o esqueleto baseado na posição relativa dos keypoints (EVANGELIDIS; SINGH; HORAUD, 2014).Song e colaboradores propuseram um modelo espaço-temporal com atenção para reconhecer atividades a partir do esqueleto do corpo humano(SONG *et al.*, 2017). Luvizon e colaboradores propuseram um modelo de multi-tarefas, capaz de estimar a pose de pessoas e reconhecer a atividade agregando as características visuais, mapas de probabilidade e a pose estimada (LUVIZON; PICARD; TABIA, 2018)

1.2 Justificativa

O intuito deste trabalho é desenvolver um algoritmo capaz de reconhecer a atividade de múltiplas pessoas baseado em suas poses 2D através de um único *frame*. Reconhecer a atividade de pessoas ajuda a predizer as próximas ações que elas irão tomar e consequentemente as ações que um sistema inteligente também deve tomar.

A tarefa de reconhecimento de atividade pode ser usada para diversas aplicações, mas uma das motivações para este trabalho foi a aplicação em robôs móveis e/ou carros autônomos, permitindo os mesmos tomarem decisões como parar ou alterar trajetória baseadas na detecção de pessoas e classificação do seu comportamento, uma medida de segurança necessária e ainda não desenvolvida.

Nesse contexto, o estudo aqui presente é responsável por desenvolver um algoritmo capaz de detectar pessoas, detectar as partes do corpo de cada pessoa, estimar a pose, e a partir dessa pose estimada classificar possíveis atividades que interessam para a situação presente de um robô móvel.

1.3 Definição do Problema

Tendo uma imagem ou um *frame* como entrada, o algoritmo deve ser capaz de reconhecer as atividades de pessoas pela pose estimada de cada, sendo assim, o problema completo pode ser divido em outros subproblemas: detecção de pessoas em imagens, estimação de pose de humanos em imagens e reconhecimento de padrões para classificação. Todos esses problemas apresentam estudos recentes da visão computacional com avanços em resultados utilizando técnicas de aprendizagem profunda. Embora geralmente são tratados como problemas separados e com soluções distintas, neste trabalho é abordado uma maneira de resolvê-los com uma única arquitetura de rede neural. A seguir são detalhados os subproblemas.

1.3.1 Detecção de Pessoa

Detecção de pessoas ou também detecção de pedestres, é uma tarefa da visão computacional em que é necessário classificar e localizar pessoas em determinada imagem.

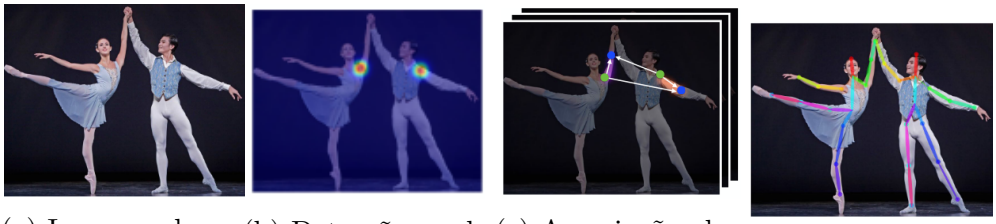
Recentes arquiteturas de *deep learning* para detecção de objetos como Mask-R CNN (HE *et al.*, 2017), RetinaNet (LIN; AI; DOLL, 2018) e Yolo (REDMON *et al.*, 2015) alcançaram bons resultados, na fronteira do que poderia ser considerado o estado da arte. A Figura 1.1 exemplifica o resultado da tarefa em que para cada detecção de um objeto de determinada classe que o algoritmo foi treinado é retornado uma caixa delimitadora (em azul na imagem) e a probabilidade de conter o objeto dentro dessa delimitação

Figura 1.1 – Detecção de pessoas utilizando-se a rede RetinaNet.

Fonte: Igor Soares, 2019

1.3.2 Estimação de Pose Humana

Estimar a pose de pessoas consiste em primeiro detectar pontos chaves do corpo humano, chamados de *keypoints*. Esses pontos são partes do corpo essenciais para se identificar a pose como: cabeça, ombros, mãos, joelhos, etc. Em seguida, é necessário associar tais pontos de maneira correta a cada pessoa presente na imagem. A Figura 1.2 retrata o fluxo do algoritmo em primeiro detectar os pontos chaves (b) de uma imagem (a) e em seguida associá-los a cada pessoa (c) e então estimar a pose (d).



(a) Imagem de entrada (b) Detecção de keypoints (c) Associação dos pontos (d) Pose estimada

Figura 1.2 – *Pipeline* de estimação de pose.

Fonte: Adaptado de Cao *et al.* (2017)

1.3.3 Reconhecimento de Atividades através de Padrões

Após a detecção de pessoas e extração de suas respectivas poses, o problema se resume a reconhecer as atividades através dos padrões apresentados pela pose. Reconhecer padrões é uma tarefa de aprendizado de máquina onde uma entrada qualquer é classificada em rótulos pré-treinados e essa classificação se dá a partir de correlações identificadas entre os dados, nesse caso as poses, e as classes ou rótulos desejados.

Ao final desta etapa, o algoritmo é capaz de detectar pessoas, estimar poses e classificá-las em alguma atividade. É possível ver o fluxo do algoritmo completo desenvolvido na Figura 1.3 para classificação de quatro atividades: correndo, andando, sentado ou parado.

Figura 1.3 – Reconhecimento de atividade através da pose.

Fonte: Igor Soares, 2019

1.4 Objetivos do Trabalho

1.4.1 Objetivo Geral

Este estudo tem como objetivo desenvolver um algoritmo de aprendizagem profunda, mais precisamente uma arquitetura única de rede neural convolucional multitarefas, capaz de em uma única inferência detectar pessoas, com metodologia *bottom-up*, detectar *keypoints*, estimar a pose em duas dimensões de cada pessoa e reconhecer uma das seguintes atividades através dos padrões de pose: andando, correndo, sentado e parado.

1.4.2 Objetivos Específicos

Para alcançar o objetivo geral, alguns pontos necessários foram traçados. Esses pontos foram destacados como objetivos ou tarefas específicas para conseguir resultados satisfatórios. Mais detalhadamente tais objetivos podem ser delineados como:

- pesquisar e definir as melhores arquiteturas para cada problema e a possibilidade de se conectarem em uma única arquitetura;
- implementação e treinamento de uma arquitetura para detecção de pessoas e estimação de pose;
- gerar um conjunto de treinamento para classificação das atividades definidas através das poses extraídas com a arquitetura anteriormente implementada;
- adicionar um classificador de poses a estrutura da rede de estimação de pose e treinar a arquitetura completa;
- realizar experimentos com os principais *datasets* públicos disponíveis para as tarefas de detecção de pessoas e estimação de pose e comparar com trabalhos anteriores;
- verificar a eficácia de se reconhecer atividades através de poses.

2 Fundamentação Teórica

Neste capítulo são abordados os principais conceitos de redes neurais artificiais, desde o neurônio artificial até redes convolucionais. Dada a proposta do trabalho de desenvolvimento de uma arquitetura completa de uma rede convolucional, para melhor compreensão é necessário uma base teórica e matemática das principais técnicas utilizadas. Tópicos mais avançados serão discutidos quando se fizerem necessários durante a metodologia no capítulo 3.

2.1 Redes Neurais Artificiais

Redes neurais artificiais são uma analogia às redes neurais biológicas, ou a maneira como acredita-se que o cérebro dos mamíferos funcione. Tal sistema procura simular o aprendizado do cérebro através de exemplos. A estrutura primária de uma rede neural artificial é o neurônio artificial, onde uma entrada é processada e levada à saída. A Figura 2.1 mostra o funcionamento do neurônio onde os dendritos captam sinais e os enviam para o corpo do neurônio onde esses sinais são processados. O resultado é passado adiante através do axônio.

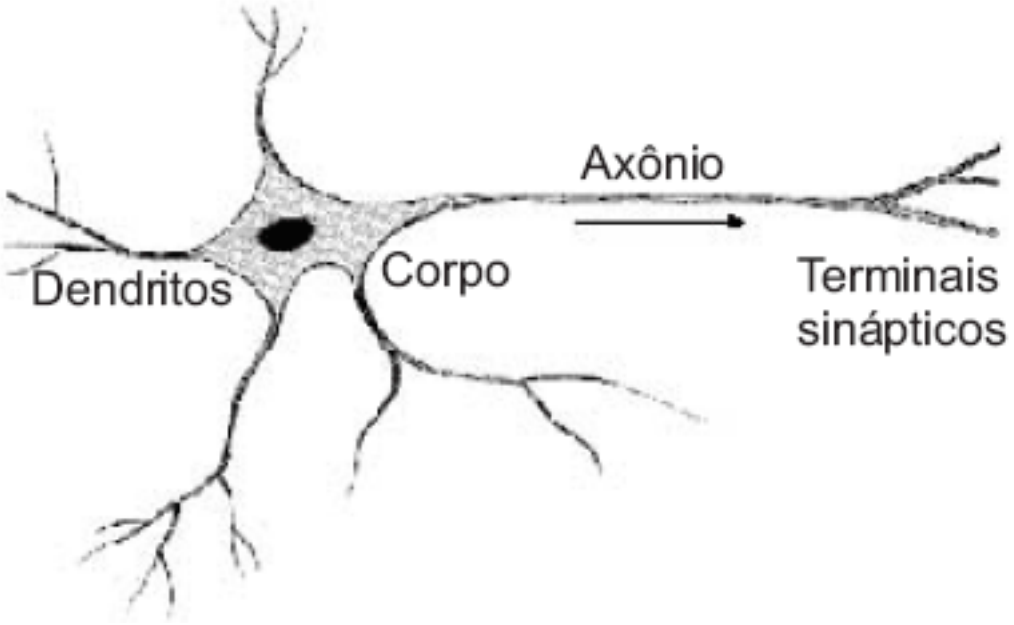


Figura 2.1 – Neurônio biológico.

Fonte: (FERNEDA, 2006)

Semelhante ao neurônio biológico, o neurônio artificial, demonstrado na Figura 2.2, capta sinais de entrada e atribui pesos a estes que em seguida são processados e a saída é gerada. O sinal de saída é repassado para o próximo neurônio em que o processo se repete, construindo a rede neural artificial.

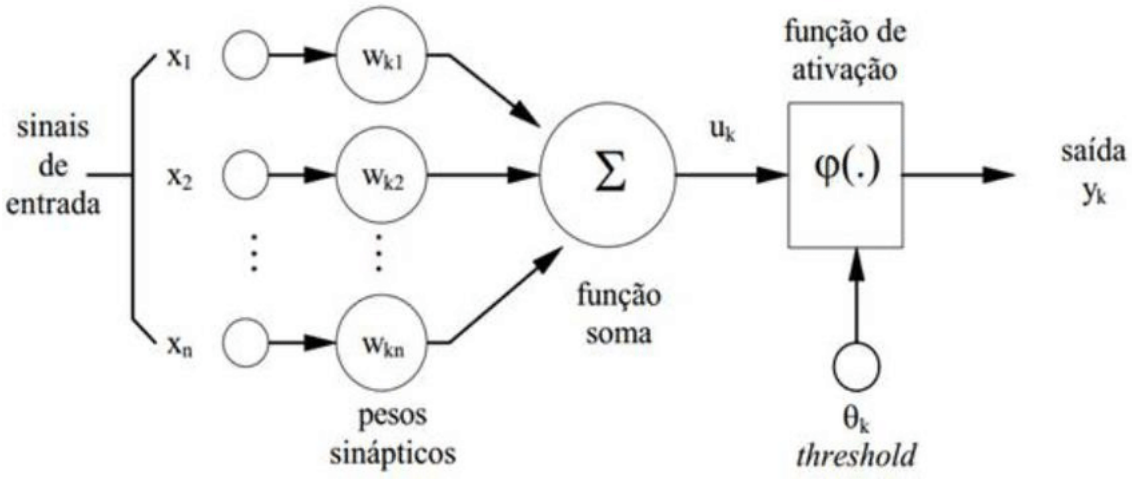


Figura 2.2 – Neurônio artificial.

Fonte: (HAYKIN, 2001)

Matematicamente, um neurônio artificial j é descrito como:

$$y_j = \varphi(\sum_{i=1}^{m} w_{ij}x_j + b_j) \tag{2.1}$$

onde $x_1, x_2, x_3....x_m$ são as entradas do neurônio; $w_{j1}, w_{j2}, w_{j3}...w_{jm}$ são os respectivos pesos do neurônio j; $\varphi(.)$ é a função de ativação; b_k é o bias; e y_k é a saída do neurônio.

A função de ativação é quem define a saída final e seu principal objetivo é limitar a amplitude do sinal de saída ou adicionar não-linearidade ao sistema. O *bias* (viés) pode ser ativado ou não, aplicando uma transformação de translação na saída.

Para o treinamento do *perceptron*, um único neurônio artificial, seja a atualização de pesos dada por:

$$w_i(t+1) = w_i(t) + r(d_j - y_j(t))x_{j,i} \tag{2.2}$$

onde d_j é a saída desejada para a entrada j; e o erro calculado como:

$$\frac{1}{s} = \sum_{j=1}^{s} |d_j - y_j(t)| \tag{2.3}$$

os passos do treinamento de um *perceptron* são mostrados no algoritmo 2.1.

Algoritmo 2.1 Algoritmo de treinamento de um perceptron

```
1: inicializa os pesos com valores pequenos e randômicos
2:
3: while erro maior que limiar do
4:    propaga a entrada e calcula a saída ( 2.1)
5:    atualiza os pesos ( 2.2)
6:    calcula o erro ( 2.3)
```

2.2 Tipos de Aprendizagem

A forma como uma rede aprende depende do seu tipo de aprendizagem. De acordo com Russell e Norvig (2009) podemos classificar a aprendizagem em três paradigmas distintos: aprendizagem supervisionada, aprendizagem não supervisionada e aprendizagem por reforço.

A aprendizagem supervisionada, como o nome indica, necessita de um supervisor, ou em outras palavras, é necessário que seja apresentado ao algoritmo a resposta esperada. Essa supervisão é necessária apenas durante a etapa de treinamento. Com a resposta esperada e resposta obtida pelo algoritmo em cada iteração, é calculado o erro e esse erro é utilizado para modificar a rede afim de otimizá-la. O treinamento do perceptron e o algoritmo *backpropagation* são exemplos de treinamentos supervisionados.

Por sua vez, a aprendizagem não supervisionada não possui nenhuma informação da saída desejada. A rede reconhece padrões nos dados de entrada através de correlações estatísticas entre estes, agrupando os dados com características próximas em uma mesma categoria ou classe.

A aprendizagem por reforço também não possui um supervisor, mas conta com uma função de recompensa, responsável por dar "bonificações"ao algoritmo por aproximar seu resultado ao objetivo esperado ou dar "punições"caso o algoritmo esteja indo na direção contrária. Assemelha-se ao aprendizado humano por repetição, onde é sabido o objetivo final mas não exatamente como chegar lá.

2.3 Redes Neurais de Múltiplas Camadas

Redes neurais de múltiplas camadas são um tipo de rede neural *feedforward*, ou seja, a entrada é apenas propagada para frente. Neste tipo de rede temos pelo menos três camadas, sendo uma de entrada, uma ou mais camadas escondidas e uma camada de

saída. Com exceção da camada de entrada, todas camadas são compostas por n neurônios artificiais e conectados entre si. Quando há conexão entre todos os neurônios, essa rede é comumente chamada de *fully connected*. A quantidade de neurônios da camada de saída indica a quantidade de classes que o problema tem. A Figura 2.3 apresenta uma rede neural artificial de múltiplas camadas com três possíveis saídas.

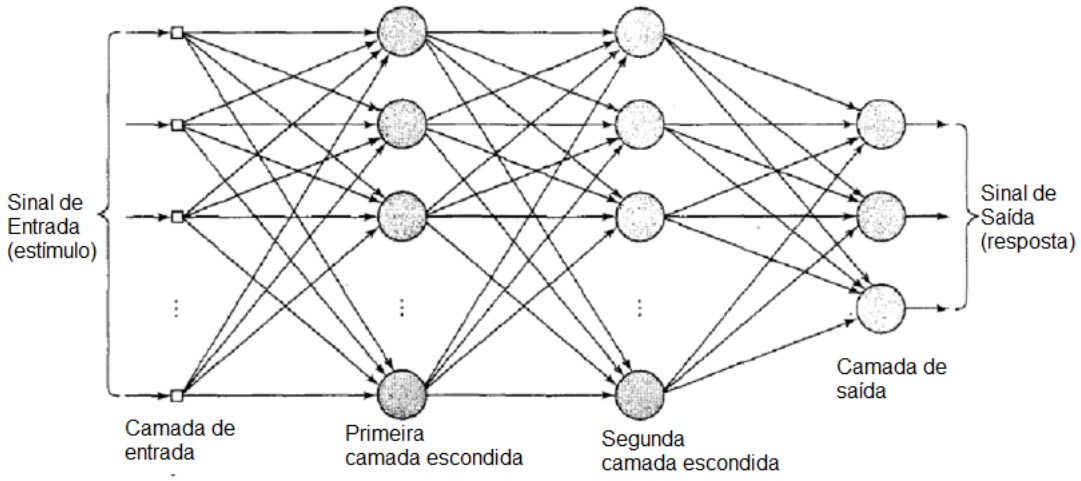


Figura 2.3 – Rede neural perceptron de múltiplas camadas.

Fonte: (HAYKIN, 1998)

Redes MLP podem ser treinadas de maneira supervisionada utilizando-se o algoritmo *backpropagation*, uma de suas variações ou métodos alternativos desenvolvidos desde os anos 80. Considerado por muitos o principal algoritmo de redes neurais e aprendizagem profunda, o *backpropagation* teve sua idealização na década de 1970, mas só em 1986 com Rumelhart, Hinton e Williams (1986) seu uso se popularizou ao ser demonstrada com rigor matemático o algoritmo, sua eficiência e a capacidade de treinar redes neurais de múltiplas camadas para resolver problemas antes insolúveis.

De forma simplificada, Goodfellow, Bengio e Courville (2016) descrevem o *backpropagation* no livro Deep Learning como sendo um método iterativo, recursivo e eficiente de se calcular a atualização dos pesos para otimizar a rede até que ela seja capaz de executar a tarefa para qual está sendo treinada.

Semelhante ao treinamento do *perceptron*, o *backpropagation* atualiza os pesos de acordo com o erro entre a entrada e a resposta desejada, porém dado a quantidade de neurônios e camadas, o erro deve ser expresso em função da posição dos neurônios e seus respectivos pesos, ou seja, o erro é retropropagado em direção a entrada. Para isso é necessário obter a derivada da função de ativação em cada neurônio, aplicando-se

o gradiente descendente para otimização da função de custo, atualizando-se os pesos e minimizando o erro. A descrição matemática do *backpropagation* é dada a seguir.

Seja um conjunto de entrada denotado por $(\boldsymbol{x}(n), \boldsymbol{d}(n)$, onde $\boldsymbol{x}(n))$ é o vetor de entrada aplicado a camada de entrada e $\boldsymbol{d}(x)$ é vetor de respostas desejada, o campo de indução local (a saída antes da aplicação de função de ativação) $v_j^{(l)}$ de um neurônio j na camada l é

$$v_j^{(l)}(n) = \sum_i w_{ji}^{(l)}(n) y_i^{(l-1)}(n) \tag{2.4}$$

onde $y_i^{(l-1)}(n)$ é o sinal de saída do neurônio i na camada anterior $l-1$ na iteração n, e $w_{ij}^{(l)}(n)$ é o peso sináptico do neurônio j na camada l que é alimentado pelo neurônio i na camada $l-1$. Assumindo uma função de saída φ, o sinal de saída do neurônio j na camada l é

$$y_j^{(l)} = \varphi_j(v_j(n)) \tag{2.5}$$

Calcular o erro da saída final:

$$e_j(n) = d_j(n) - o_j(n) \tag{2.6}$$

onde $d_j(n)$ é o j-ésimo elemento do vetor de respostas desejada $\boldsymbol{d}(n)$ e $o_j(n)$ é a saída do neurônio j na última camada.

O gradiente local da rede é então definido por:

$$\delta_j^{(l)}(n) = e_j^{(L)}(n)\varphi_j'(v_j^{(L)}(n)) \tag{2.7}$$

para um neurônio j na camada de saída L;

$$\delta_j^{(l)}(n) = \varphi_j'(v_j^{(l)}(n)) \sum_k \delta_k^{(l+1)}(n) w_{kj}^{(l+1)}(n) \tag{2.8}$$

para um neurônio j na camada escondida l;

O ajuste dos pesos sinápticos da rede na camada l é realizado pela regra delta generalizada:

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha[\Delta w_{ji}^{(l)}(n-1)] + \eta \delta_j^{(l)}(n) y_i^{(l-1)}(n) \tag{2.9}$$

Para melhor compreensão, um sumário gráfico é apresentado na Figura 2.4 onde a parte de cima (em preto) é propagação do sinal e a de baixo (em azul) a retropropagação com o ajuste dos pesos.

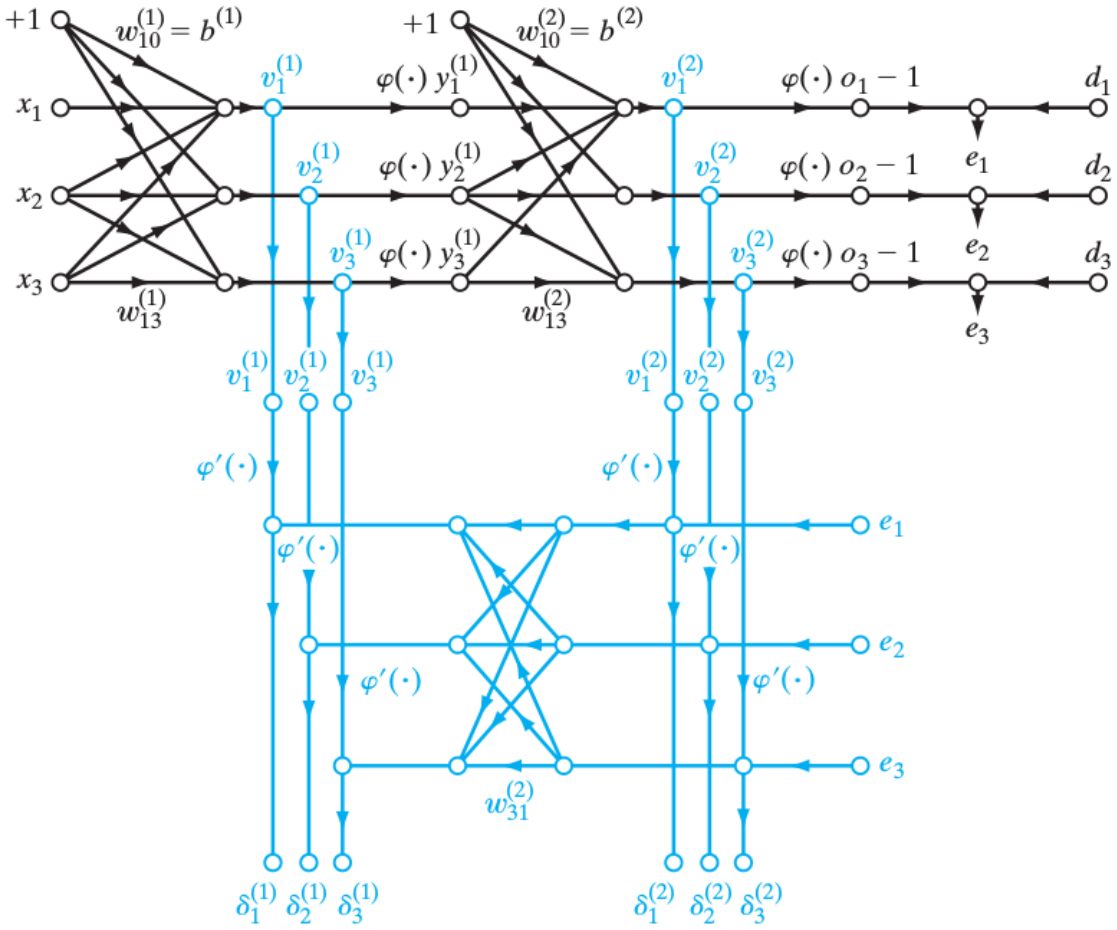


Figura 2.4 – Sumário gráfico do aprendizado com *backpropagation*.

Fonte: (HAYKIN, 2009)

Para cada amostra de treinamento propaga-se o sinal de entrada calculando-se a saída de cada neurônio em cada camada de acordo com as expressões 2.1 e 2.5, onde a saída dos neurônios de determinada camada servem de entrada para a próxima camada. Com a saída dos neurônios da última camada, calcula-se o sinal de erro de acordo com 2.6, terminando a primeira etapa do algoritmo, chamada de propagação. Em seguida, inicia-se o processo de retropropagação do erro, primeiro calculando o gradiente local de acordo com as expressões 2.8 e 2.7 e então atualizando-se os pesos da rede (2.2). O processo de propagação e retropropagação itera sobre todas as entradas e repete-se até que um critério de parada seja alcançado, podendo ser por número de iterações, erro suficientemente pequeno, etc.

2.4 Redes Neurais Convolucionais

Uma rede convolucional ou CNN (*Convolutional Neural Network*) é uma rede profunda capaz de adicionar informações extras para a rede *fully connected* (classificação), sendo geralmente usada pra problemas de visão computacional (imagens). É dita ser uma rede profunda por causa da quantidade de camadas e parâmetros consideravelmente superiores às redes multicamadas.

Goodfellow, Bengio e Courville (2016) descrevem redes convolucionais como sendo redes neurais que empregam operações de convolução, um tipo de operação linear. São treinadas também utilizando-se *backpropagation* (LECUN; BOSER; HENDERSON, 1989) e nos últimos anos podem ser consideradas o estado da arte para a maioria dos problemas de visão computacional.

Usualmente, uma rede convolucional demanda pouco pré-processamento dos dados de entrada, usando diretamente imagens. Antes de sua chegada, algoritmos tradicionais de visão computacional aplicavam uma série de filtros nas imagens para extração de características e então essas características eram classificadas com algum algoritmo de *machine learning*. Sem a necessidade de se aplicar filtros antes da entrada em uma rede convolucional, e com a obtenção de resultados melhores, percebeu-se que o treinamento das operações convolucionais ensinam à rede uma combinação de filtros melhor que as geradas por uma pessoa. Essa automatização da extração de característica para classificação é sua principal vantagem. A Figura 2.5 visualiza os diferentes filtros convolucionais obtidos em um treinamento de rede convolucional.

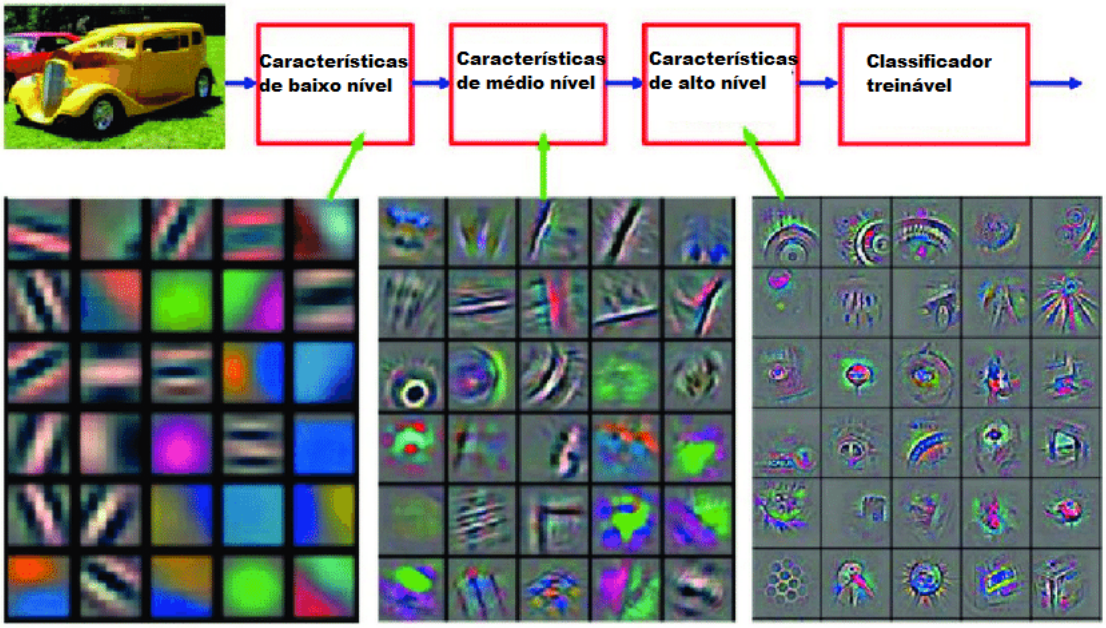


Figura 2.5 – Visualização dos filtros convolucionais em diferentes profundidades da rede.
Fonte: (ØRSTAVIK; MIDTBø, 2017)

Redes convolucionais possuem necessariamente camadas convolucionais, mas não se limitam apenas a estas. Ao longo dos anos, surgiram algumas técnicas que são aplicadas entre camadas convolucionais para melhores resultados e diminuição do tempo de treinamento, sendo este último fator uma das principais desvantagens de redes profundas. Entre estas técnicas destaca-se: *pooling*, *batch normalization* e *dropout*.

2.4.1 Camada Convolucional

Camadas convolucionais formam os principais blocos de uma rede convolucional e é nelas que a maior parte dos cálculos são computados. Responsáveis pelas operações de convoluções, os parâmetros de uma camada convolucional são filtros com aprendizagem, ou seja, os valores desses filtros são atualizados durante o treinamento, ajustando cada filtro a um valor que melhore a saída.

Todo filtro é uma matriz de dimensões espaciais pequenas, mas com profundidade igual o tamanho da profundidade da entrada. O filtro percorre toda imagem, produzindo um mapa de ativação de duas dimensões com os valores do filtro em cada posição espacial. O mapa de ativação de todos os filtros são sobrepostos produzindo a entrada da próxima camada convolucional.

Os hiperparâmetros de uma camada convolucional são: o número de filtros a serem utilizados, o tamanho espacial dos filtros ou tamanho do *kernel*, o número de *strides* e a quantidade de *zero paddings*, ou seja:

Stride é o número de passos em que o filtro se movimenta ao percorrer a entrada. Caso seja valor 1 (um), todos *pixels* da imagem são percorridos um por um.

Zero padding é a quantidade de colunas com valores nulos que é preciso adicionar a imagem para que possamos controlar o tamanho espacial da saída.

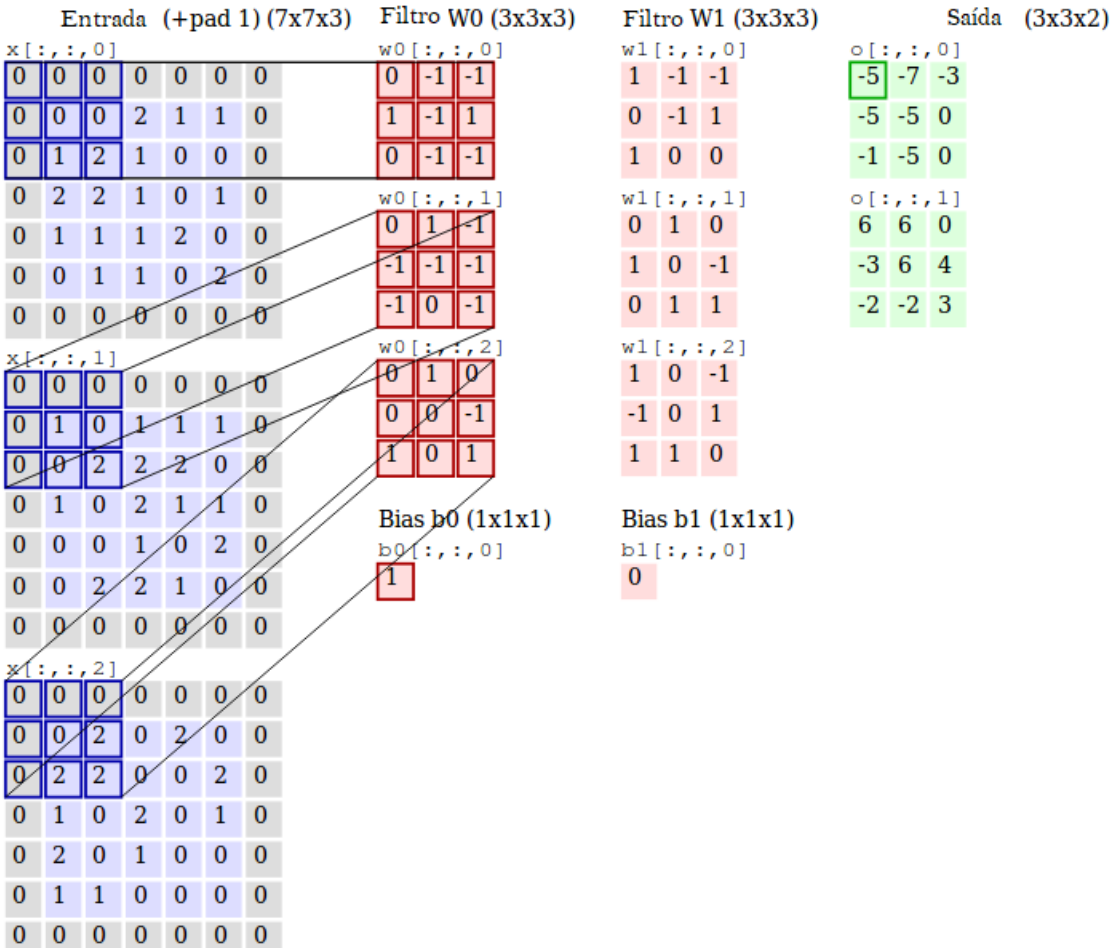


Figura 2.6 – Demonstração de uma convolução com filtro de *kernel* 3x3.

Fonte: Repositório digital [1]

[1] Disponível em: http://cs231n.github.io/convolutional-networks/conv. Acesso em julho 2019

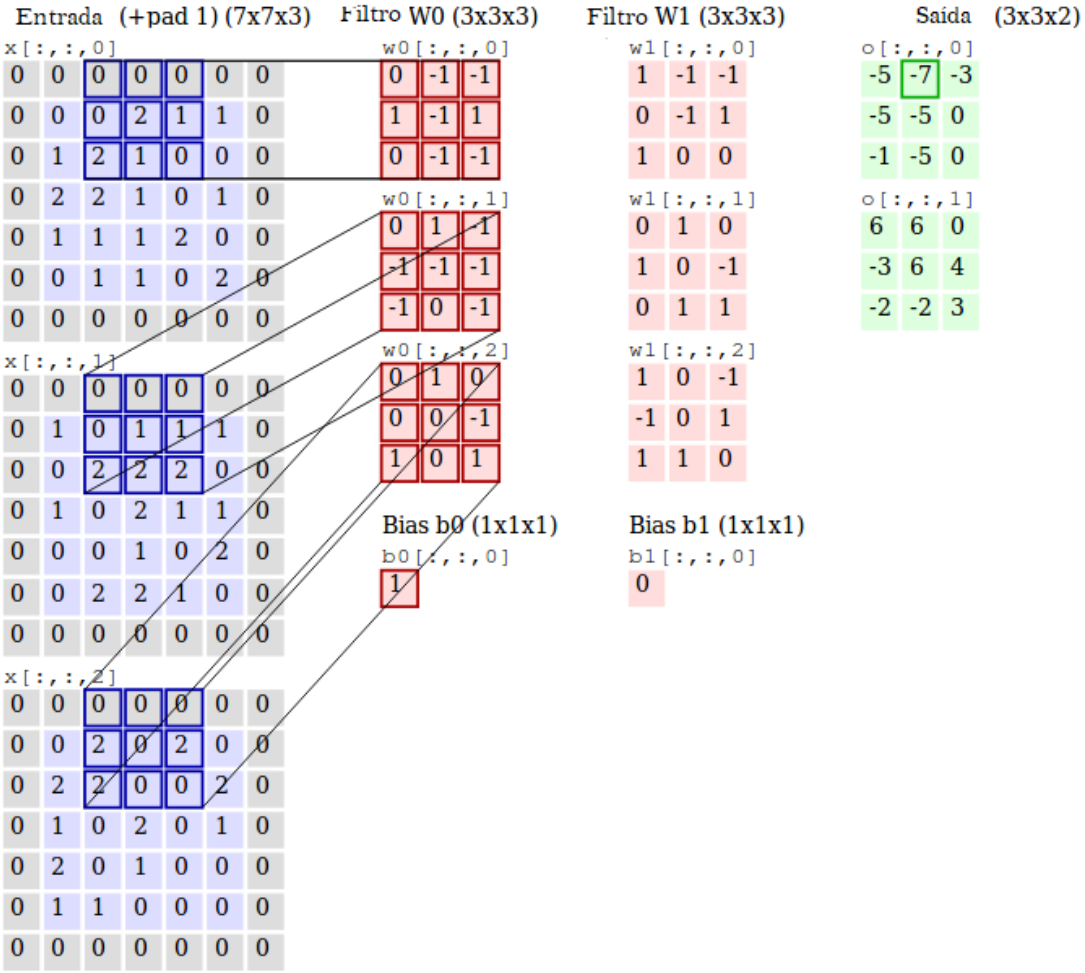


Figura 2.7 – Deslocamento do filtro sobre a entrada com *stride* 2.

Fonte: Repositório digital [2]

Em resumo, para uma entrada $W \times H \times D$, define-se o número de filtros K, o tamanho do *kernel* F, o tamanho do *stride* S e a quantidade de *zero padding* P e calcula-se o volume de saída $W_s \times H_s \times D_s$ como:

$$W_s = (W - F + 2P)/S + 1 \tag{2.10}$$

$$H_s = (H - F + 2P)/S + 1 \tag{2.11}$$

$$D_s = K \tag{2.12}$$

[2] Disponível em: http://cs231n.github.io/convolutional-networks/conv. Acesso em julho 2019

2.4.2 Pooling

A camada de *pooling* reduz a dimensão espacial do mapa de características (saída da camada convolucional), preservando a dimensão de profundidade (SCHERER; MüLLER; BEHNKE, 2010). Isso é feito combinando a saída de um grupo de neurônios de uma camada em apenas um neurônio na próxima camada.

Existem dois tipos de *pooling*, local e global, podendo executar dois tipos de operações diferentes: valor máximo e valor médio.

O *local pooling* executa suas operações em pequenas janelas de tamanhos fixos, geralmente 2×2, percorrendo todo mapa de características, semelhante ao *kernel* da camada convolucional. Já o *pooling* global atua em todos neurônios da camada convolucional (CIREsAN *et al.*, 2011). Genericamente, o *pooling* local reduz a dimensionalidade espacial de uma entrada $W \times H \times D$ de acordo com:

$$W_s = (W - F)/S + 1 \tag{2.13}$$

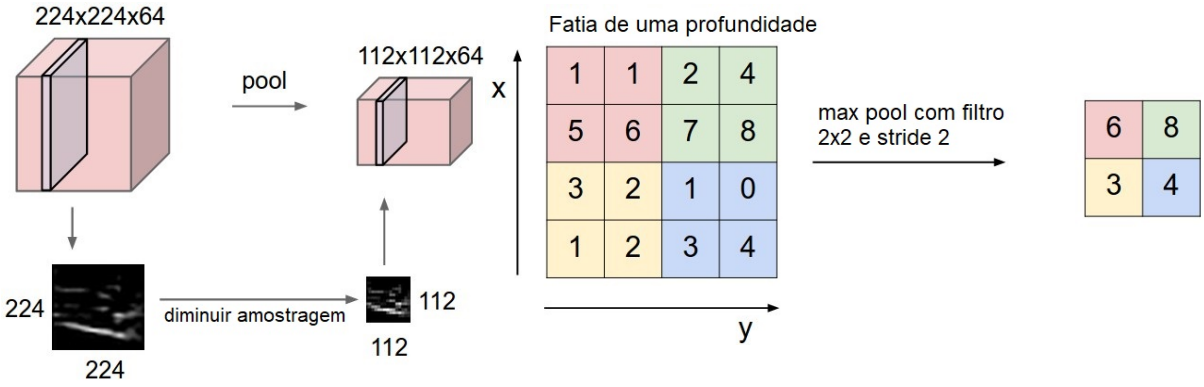
$$H_s = (H - F)/S + 1 \tag{2.14}$$

$$D_s = (D - F)/S + 1 \tag{2.15}$$

onde F é o tamanho da janela de *pooling* e S o tamanho do seu *stride*. Quanto a operação, o *pooling* pode operar com o máximo valor da janela ou a média dos valores presentes na janela.

A utilização do *pooling* para redução de dimensionalidade espacial faz-se necessária para ganho de performance computacional, uma vez que com menos parâmetros o treinamento é executado mais rápido. Além disso, há diminuição da possibilidade de *overfitting*, um problema de baixa ou nenhuma capacidade de generalização das redes neurais (CARUANA; LAWRENCE; GILES, 2000), além de capturar invariâncias nos dados dos mapas de características (SCHERER; MüLLER; BEHNKE, 2010).

A Figura 2.8 demonstra visualmente o funcionamento do *pooling* onde, em (a), uma entrada de $224 \times 224 \times 64$ sofre *pooling* de janela 2×2 e stride 2 tendo sua amostragem reduzida para $112 \times 112 \times 64$. A profundidade de 64 não é alterada. Já em (b), temos um exemplo da operação de *max pooling* (valor máximo).



(a) (b)

Figura 2.8 – Exemplo de pooling local.

Fonte: Repositório digital [3]

2.4.3 Batch Normalization

Ioffe e Szegedy (2014) propuseram uma técnica para aumentar a velocidade, a performance e a estabilidade do treinamento de redes neurais artificiais. Chamada de *batch normalization*, acreditava-se que essa técnica suavizava o deslocamento de covariância interna, onde a inicialização aleatória de parâmetros e mudanças na distribuição dos dados de entrada atrapalhavam a taxa de aprendizagem da rede. Contudo, estudos recentes mostraram que na verdade o *batch normalization* suaviza a função objetivo aumentando a performance da rede (SANTURKAR *et al.*, 2018).

A camada de *batch normalization* apresenta um passo de normalização que arruma a média e a variância das entradas de cada camada. A normalização é treinada para cada *mini-batch*, um conjunto pequeno de entradas.

Seja B um *mini-batch*, a média e a variância de B são denotadas respectivamente por 2.16 e 2.17.

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.16}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{2.17}$$

[3] Disponível em: http://cs231n.github.io/convolutional-networks/conv. Acesso em julho 2019

Para uma camada com entrada de d dimensões, $x = (x^{(1)}, ..., x^{(d)})$, cada dimensão é normalizada separadamente com:

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{{\sigma_B^{(k)}}^2 + \epsilon}} \tag{2.18}$$

onde $k \in [1, d]$, $i \in [1, m]$ e ϵ é uma constante arbitrária pequena para estabilidade numérica.

A transformação da camada de *batch-normalization* é então dada por:

$$y_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)} \tag{2.19}$$

onde $\gamma^{(k)}$ e $\beta^{(k)}$ são parâmetros com aprendizado durante o processo de otimização.

É importante notar que a camada de *batch-normalization* apresenta comportamentos diferentes durante a fase de treinamento e a fase de inferência. Durante o treinamento, calcula-se a média e a variância do *mini-batch* de entrada e realiza-se o ajuste dos pesos porém, na fase de testes, não existe garantia de que a entrada tenha a mesma distribuição do conjunto de treinamento e por isso usa-se a mesma média e variância do treinamento. Caso contrário, utilizar os valores de média e variância da entrada no teste com pesos treinados em valores diferentes resultaria em resultados piores.

2.4.4 Dropout

Dropout é uma técnica de regularização de redes neurais para evitar *overfitting*. Ao contrário de outras técnicas de regularização, o *dropout* não depende da modificação da função de custo. Sua idéia é retirar unidades de neurônios de determinadas camadas aleatoriamente, prevenindo que a rede se adapte demais aos dados de treinamento e perca sua capacidade de generalização (HINTON, 2014).

Semelhante ao *batch-normalization*, o *dropout* funciona diferente durante o teste, na verdade não realizando nenhuma operação. Assim, as camadas tem seus neurônios retirados apenas durante o treinamento.

Na Figura 2.9 é possível ver o funcionamento do *dropout* (à direita) em comparação com uma mesma rede sem *dropout* (à direita).A quantidade de neurônios a serem retirados é definido como parâmetro em porcentagem.

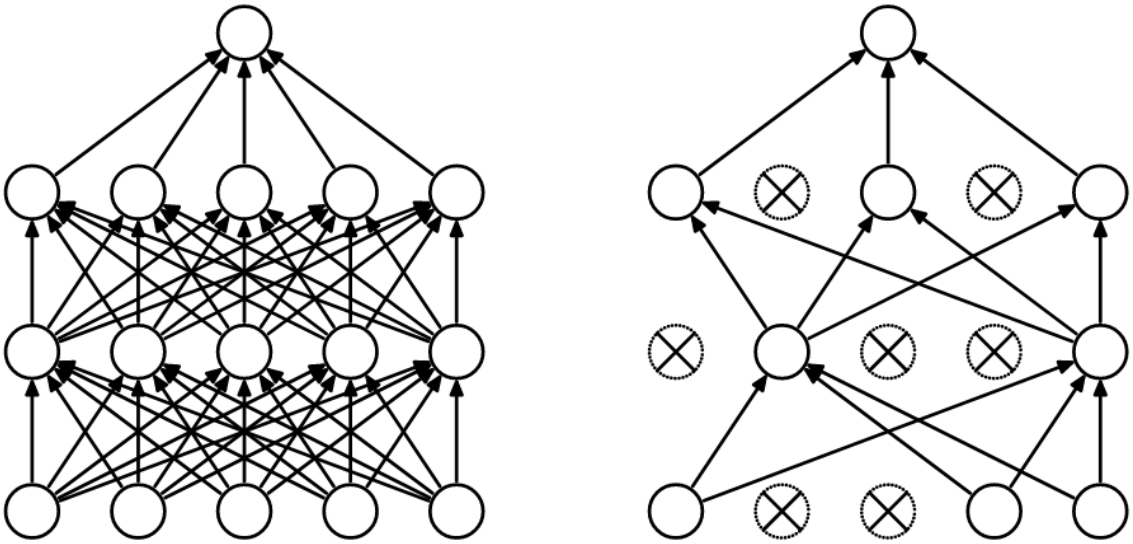


Figura 2.9 – Exemplo de dropout.

Fonte: (HINTON, 2014)

2.4.5 Camada fully connected

Camadas totalmente conectadas tem todos seus neurônios conectados entre si, em todas as camadas. Na prática, são as mesmas camadas de uma rede neural multicamadas (MLP). Essas camadas estão no topo das redes profundas, tendo como entrada os mapas de características "achatados", ou seja, a saída em três dimensões das redes convolucionais transformadas em uma única dimensão.

Essa parte da rede é responsável pela classificação e reconhecimento de padrões em todas características extraídas pelas camadas anteriores (convolucionais, *pooling*,etc). Uma vez que uma rede convolucional seja treinada em determinada atividade, é possível congelar os parâmetros da parte convolucional e retreinar apenas as camadas *fully connected* para execução da mesma atividade em classes diferentes das previamente treinadas. Esse processo é conhecido com transferência de aprendizagem e é bastante utilizado por reduzir consideravelmente o tempo de treinamento (YOSINSKI *et al.*, 2014).

Existem arquiteturas em que as camadas *fully connected* não se fazem necessárias, tendo como objetivo predizer em cima dos próprios mapas de características. O uso é mais comum é para segmentação semântica (SHELHAMER; LONG; DARRELL, 2017).

3 Desenvolvimento

Neste capítulo é detalhado todo o processo para o desenvolvimento do método proposto para classificar atividades através da pose. Uma única rede convolucional é apresentada, porém por ser multitarefas, tal rede é a combinação de diversas arquiteturas alinhadas para objetivos semelhantes. Todas essas arquiteturas são detalhadas nas próximas seções. A Figura 3.1 mostra o resultado final deste estudo, em que uma imagem de entrada qualquer tem suas características extraídas por convoluções que são igualmente compartilhadas para dois submodelos: um detector de pessoas e outro detector de keypoints. Após ambas detecções, os resultados são combinados em uma camada para estimação do pose e associação a cada pessoa. As poses finais são então classificadas por uma rede neural multicamadas em prováveis atividades.

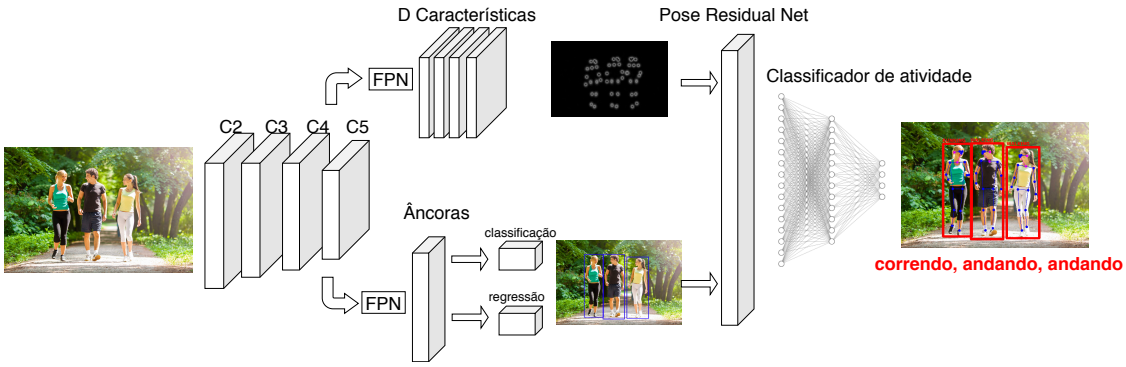


Figura 3.1 – Método proposto.

Fonte: Igor Soares, 2019

3.1 Definição de Arquiteturas a Serem Utilizadas

Para ser capaz de executar multitarefas ao mesmo tempo, é essencial que o algoritmo final apresente todas estruturas necessárias para cada tarefa e que seja possível a conexão entre essas estruturas. Assim, são analisadas as principais arquiteturas para as atividades de detecção de pessoas, estimação de pose e classificação de dados e suas semelhanças.

3.1.1 Detecção de Pessoas

Por motivos de tempo de inferência, o algoritmo proposto deve ser da metodologia *bottom-up*, portanto o submodelo de detecção de pessoas também deve possuir o melhor

tempo de execução possível, e nesse sentido, analisaremos os detectores chamados de *one-shot*.

Detectores de objeto *one-shot* recebem esse nome pois em apenas uma inferência são capazes de detectar múltiplos objetos em uma imagem (LIU *et al.*, 2015; REDMON *et al.*, 2015; LIN; AI; DOLL, 2018). Os detectores chamados de *two-shots*, primeiro necessitam identificar as regiões propostas (*Regional Proposal Networks - RPN*) e depois os objetos em cada uma dessas regiões (HE *et al.*, 2017; GIRSHICK, 2015).

O primeiro detector de objetos *one-shot* foi a rede Yolo (*You only look once*) (REDMON *et al.*, 2015). Contendo 24 camadas convolucionais seguidas de 2 camadas *fully connected* (*perceptrons* de múltiplas camadas), Redmon *et al.* (2015) alcançaram 63.4% mAP (*mean Average Precision*), 7 pontos abaixo da Faster R-CNN (REN *et al.*, 2015), mas quase sete vezes mais rápido (45 FPS contra 7 FPS).

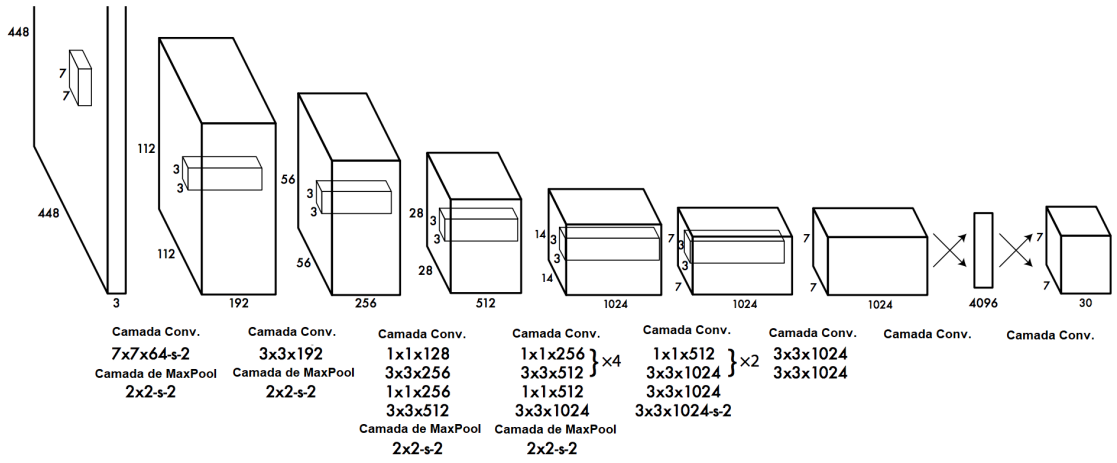


Figura 3.2 – Arquitetura da Yolo.

Fonte: (REDMON *et al.*, 2015)

Em seguida foi desenvolvida a rede SSD (*Single-Shot Detector* (LIU *et al.*, 2015), uma rede convolucional que produz uma coleção de caixas delimitadoras (*bounding boxes*) e avalia a probabilidade da existência de um objeto nestas caixas, seguido do algoritmo *non-maximum suppresion* para reduzir as caixas e produzir a detecção final. As primeiras camadas da rede são as camadas da VGG16 (SIMONYAN; ZISSERMAN, 2014), uma arquitetura para classificação de imagens, seguidas de camadas auxiliares para a detecção. Como resultado, obteve 74.3% de mAP e 59 FPS.

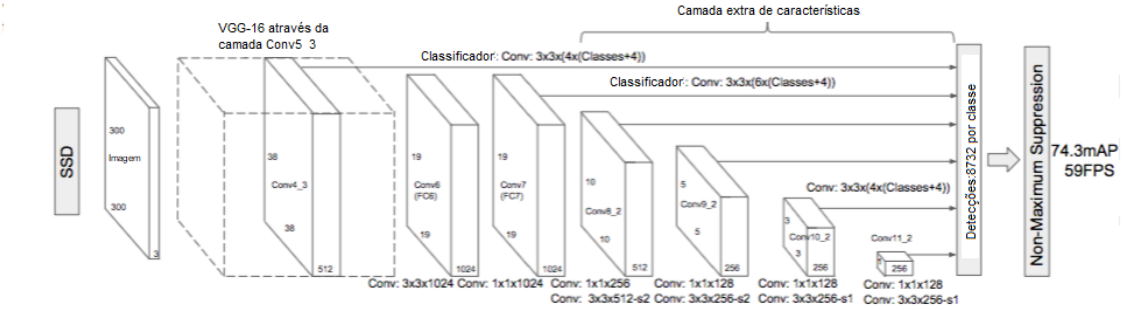


Figura 3.3 – Arquitetura da SSD.

Fonte: (LIU *et al.*, 2015)

Recentemente, o grupo de pesquisa em inteligência artificial do Facebook, *Facebook AI Research (FAIR)*, desenvolveu um novo detector de objetos *one-shot* chamado RetinaNet (LIN; AI; DOLL, 2018), no qual foi implementada uma nova função de custo para treinamento chamada de *focal loss*. Tal detector não possui tempo de inferência tão rápido quanto a Yolo e a SSD, porém possui os melhores resultados no *dataset* de detecção de objeto mais recente, o MS COCO (LIN; ZITNICK; DOLL, 2014) (32.5 mAP contra 28 mAP da SSD e 21.6 da YOLO). A rede é composta de camadas convolucionais para extração de características com uma rede piramidal para múltiplas escalas, seguidas de duas subredes, uma para classificação de objetos e outra para regressão dos pontos das *bounding-boxes*. A Figura 3.5 apresenta a comparação de tempo de inferência e mAP obtido pelos principais detectores *one-shot*.

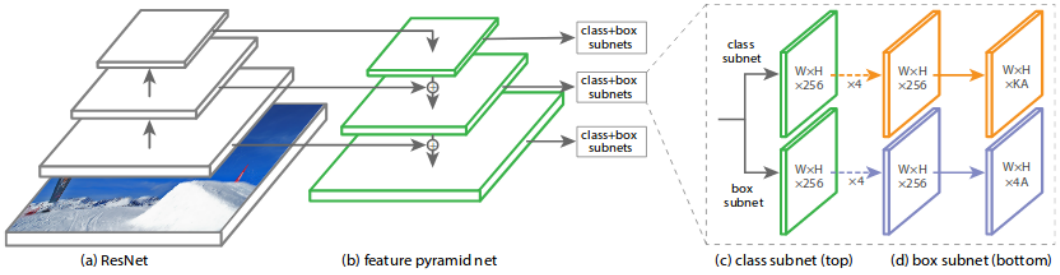


Figura 3.4 – Arquitetura da RetinaNet.

Fonte: (LIN; AI; DOLL, 2018)

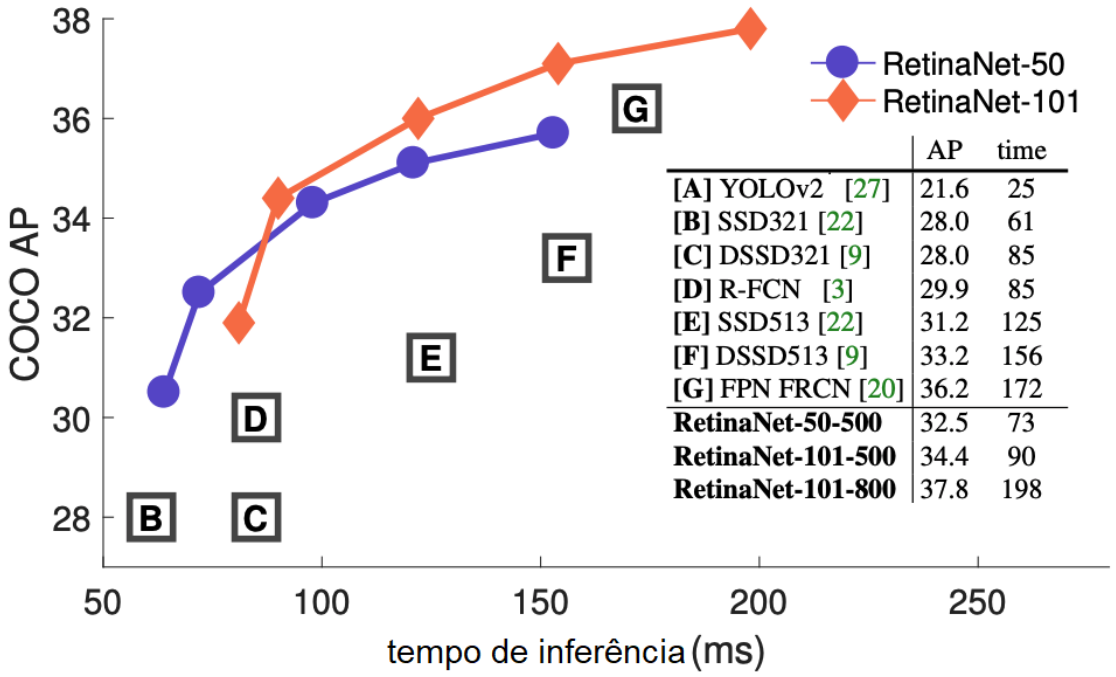


Figura 3.5 – Comparação entre RetinaNet e outros métodos one-shot.

Fonte: (LIN; AI; DOLL, 2018)

Como a metodologia *bottom-up* depende de um bom detector de pessoas, é utilizado a RetinaNet para compor a arquitetura final, que possui bons resultados na tarefa de detecção de objetos e um tempo de inferência razoável por ser *single-shot*. A teoria e matemática deste modelo é retratada na seção 3.2.2.

3.1.2 Estimação de Pose Humana

Comumente na tarefa de estimação de pose, utiliza-se um detector de pessoas na abordagem *top-down*, onde primeiro se detecta cada pessoa e depois realiza-se a detecção de *keypoints* e estimação da pose. Porém, este estudo visa uma abordagem *bottom-up*, devido ao menor tempo de inferência e possibilidade de utilização do algoritmo em trabalhos futuros com *hardware* mais modesto. Dois trabalhos na literatura estimam pose com um detector de pessoas, Li *et al.* (2018) e Kocabas, Karagoz e Akbas (2018).

Li *et al.* (2018) propuseram um modelo semelhante ao apresentado por Cao *et al.* (2017), predizendo os *keypoints* com mapas de confiança em uma rede multi-estágios, porém ao invés de *Part Affinity Fields*, Li *et al.* (2018) usaram campos vetoriais de direção entre cada *keypoint* para associação, juntamente com as delimitações das *bounding boxes* preditas por um detector de pessoas. A imagem de entrada é repassada para duas redes

distintas, uma para detecção de pessoas, neste caso a YOLO, e outra rede para detecção de *keypoints*. As duas saídas são agrupadas para estimar a pose. A Figura 3.6 mostra a visão completa do funcionamento e a Figura 3.7 retrata a arquitetura da rede, onde a imagem tem suas características extraídas por uma rede de classificação de imagens (ResNet) e repassadas para múltiplos estágios. Em cada estágio, a primeira parte prediz os mapas de confianças e a segunda os campos vetoriais

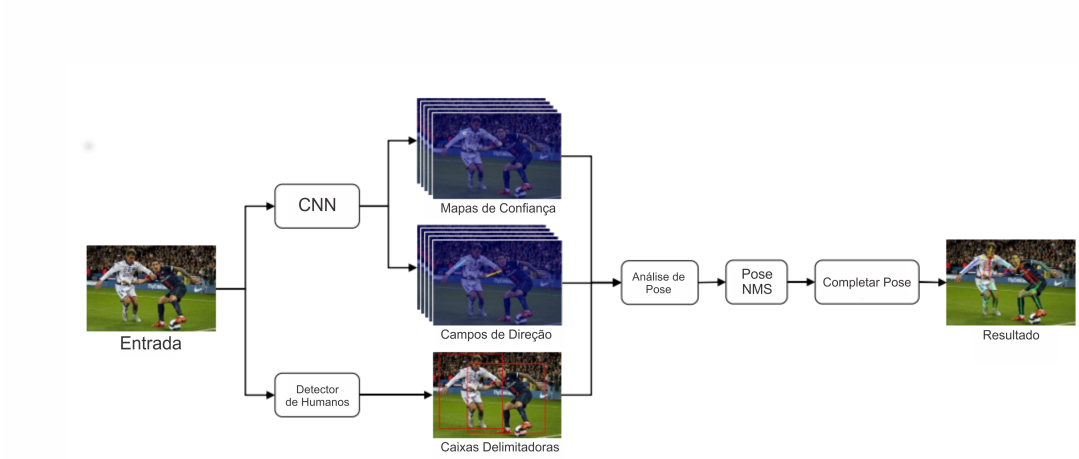


Figura 3.6 – Visão geral do fluxo do algoritmo de Li *et al.* (2018).

Fonte: (LI *et al.*, 2018)

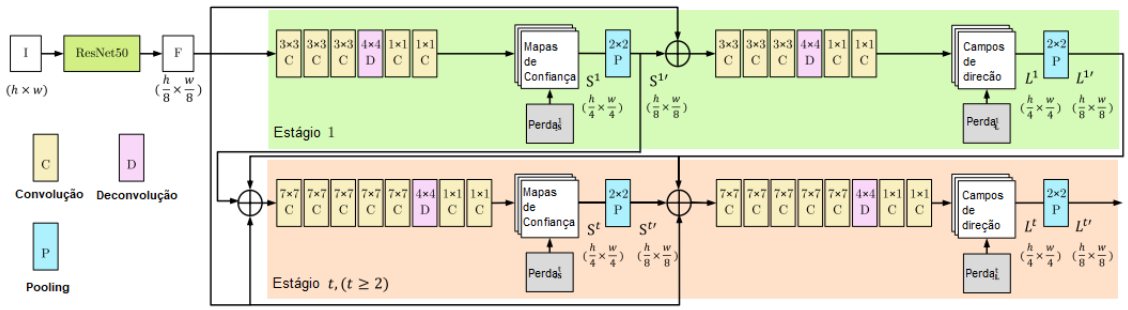


Figura 3.7 – Arquitetura da rede de Li *et al.* (2018) responsável por detectar *keypoints*.

Fonte: (LI *et al.*, 2018)

Por sua vez, Kocabas, Karagoz e Akbas (2018) apresentaram a MultiPoseNet, uma rede convolucional multi-tarefas capaz de detectar *keypoints* e pessoas com um mesmo *backbone* (extrator de características). Além disso, Kocabas, Karagoz e Akbas (2018) desenvolveram uma rede residual de poses, principal contribuição do artigo, capaz de estimar a pose com as saídas das subredes anteriores utilizando padrões de poses existentes pré-treinados.

A Figura 3.8 mostra o funcionamento da rede residual ao identificar padrões de poses presentes, e portanto possíveis de existirem, no *dataset* de treinamento e utilizar esses padrões para estimar novas poses, corrigindo a associação dos *keypoints* em poses improváveis.

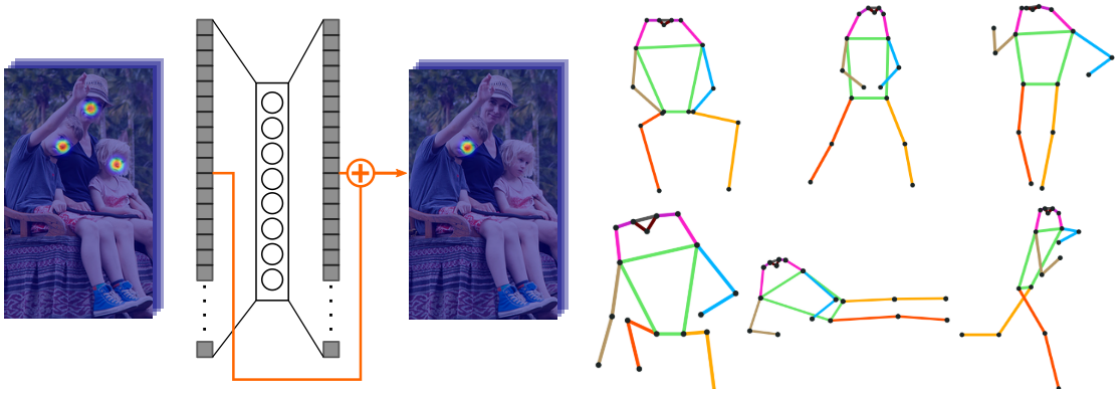


Figura 3.8 – Rede residual de poses.

Fonte: (KOCABAS; KARAGOZ; AKBAS, 2018)

Dadas as características desse modelo e a possibilidade da rede residual de poses também ser aplicada a arquitetura final, para estimar poses é utilizado o detector de *keypoints* de Kocabas, Karagoz e Akbas (2018) e melhor discutido em 3.2.3.

3.1.3 Reconhecimento de Atividade Através das Poses

Reconhecer uma atividade significa classificar determinada atividade de acordo com os dados de poses apresentados. Para isso, é necessário algum algoritmo de reconhecimento de padrões. Com a projeção de utilização de redes convolucionais em todas as etapas discutidas até aqui, para facilitação de conexão e elaboração de uma única rede neural final, o classificador será uma rede perceptron de múltiplas camadas.

3.2 O Modelo

Nesta seção são apresentadas a teoria e a matemática por trás de todos submodelos necessários e etapas para unificação em uma única arquitetura. A ordem de discussão é: *backbone*, detector de *keypoints* (MultiPoseNet), detector de pessoas (RetinaNet), rede residual de poses e classificador de atividades.

3.2.1 O Backbone

Nesta parte da arquitetura é feita a extração de características das imagens de entrada. Estas características são compartilhadas e utilizadas para o treinamento dos submodelos *Keypoint Subnet* e *Person Detection Subnet*.

O *backbone* é constituído de uma ResNet (HE *et al.*, 2017), uma rede convolucional residual mais fácil de ser otimizada, com duas *Feature Pyramid Networks* (FPN) (LIN *et al.*, 2017) conectadas, uma para cada submodelo. Essa estrutura foi escolhida para a extração de mapas de características piramidais, possibilitando assim uma representação de características em múltiplas escalas, onde todos os níveis de características geram informações úteis aos modelos conectados.

ResNet - Rede Residual

Redes neurais profundas são mais difíceis de serem treinadas. Quanto mais camadas, mais essa dificuldade aumenta. Um dos problemas mais discutidos na literatura são a explosão e desaparecimento de gradiente. Bengio, Simard e Frasconi (1994) apresentam esses problemas e Glorot e Bengio (2010) fazem uma discussão extensa sobre tal assunto. A explosão de gradiente acontece quando o erro cumulativo da rede resulta em valores muito grandes, levando a atualização de pesos brusca causando instabilidade no treinamento e consequentemente *overfitting*. Já o desaparecimento, principal problema de redes profundas, acontece devido a forma como o backpropagation funciona. Ao propagar a derivada do erro por várias camadas, as últimas camadas a serem treinadas (as primeiras da rede) recebem um gradiente quase nulo (algumas vezes praticamente nulo), tendo seus pesos não atualizados. Isso impede a otimização das redes, e por muito tempo, contra intuitivamente, aplicar mais camadas a uma rede piorava seus resultados.

He *et al.* (2015) desenvolveram a ResNet, uma rede com blocos residuais, capaz de não só contribuir para a solução destes problemas (na época da sua publicação já haviam alguns métodos que contornavam esses dois problemas também como o *batch normalization* já discutido e a função de ativação ReLU), como solucionar o problema de degradação, outro problema de redes profundas e até então sem solução. A degradação é um aumento na taxa de erro que se observou ao adicionar mais camadas. Como não acontecia em

redes mais "rasas", He *et al.* (2017) propuseram criar resíduos entre blocos de camadas convolucionais.

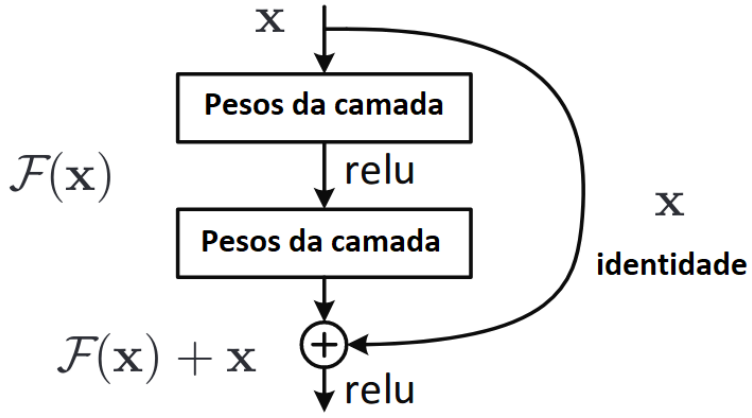


Figura 3.9 – Bloco residual.

Fonte: (HE *et al.*, 2015)

Esse mapeamento residual, nada mais é que do adicionar ao mapeamento convencional o resíduo anterior a este, criando conexões de atalho na propagação da rede. Formalmente, a propagação entre blocos é dada por $F(x) + x$.

A adição de resíduo não provoca aumento de parâmetros e nem complexidade computacional, e mantém normal o treinamento com o *backpropagation*. Graças a essas conexões de atalho, He *et al.* (2015) obtiveram os melhores resultados na competição do ImageNet (DENG *et al.*, 2009) em 2015, apresentando três redes residuais com 50, 101 e 152 camadas respectivamente, contra 22 camadas do ganhador de 2014.

Nome camada	Tamanho saída	18	34	50	101	152
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figura 3.10 – Arquitetura da ResNet em todas suas profundidades.

Fonte: Adaptado de (HE *et al.*, 2015)

FPN - Rede de Pirâmides de Características

Quando executada uma tarefa de detecção de objetos, é importante que o detector consiga obter informações em diferentes níveis de escala. Uma solução comum é apresentar a rede a mesma imagem em diferentes tamanhos, porém isso torna o processo muito mais lento.

(LIN *et al.*, 2017) exploraram a inerente hierarquia multi-escalas das redes convolucionais, em que no próprio processo reduzem o tamanho espacial do mapa de características, para desenvolver uma arquitetura com conexões laterais construindo um mapa de características em todas escalas.

Esta arquitetura, chamada *Feature Pyramid Network*, pega a última camada de cada estágio de uma rede convolucional, onde apresentam diferenças no tamanho espacial. No caso da ResNet, a FPN utiliza as saídas dos últimos quatro blocos convolucionais C_2, C_3, C_4, C_5, tamanhos $1x, \frac{1}{2}x, \frac{1}{4}x, \frac{1}{8}x$ respectivamente.

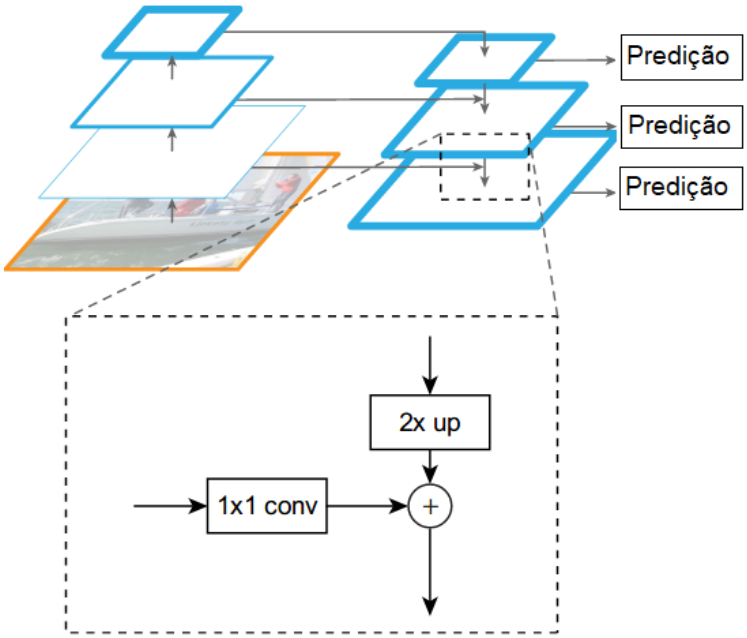


Figura 3.11 – Conexões laterais entre os blocos residuais da ResNet e a FPN.

Fonte: (LIN *et al.*, 2017)

É aplicado em cada uma dessas camadas uma convolução 1×1 de 256 filtros para reduzir a profundidade delas para 256. A camada de menor tamanho espacial C_5 torna-se M_5, então aumenta-se a amostragem (*upsampling*) em $2x$ utilizando *nearest neighbors* somando o resultado com o mapa de características do mesmo tamanho. O processo se

repete até a camada de maior tamanho. Em cada processo aplica-se uma convolução 3×3 para reduzir o efeito de *aliasing* do *upsampling*. A FPN resulta então em quatro mapas de características piramidais P_2, P_3, P_4, P_5. A Figura 3.12 ilustra esse processo.

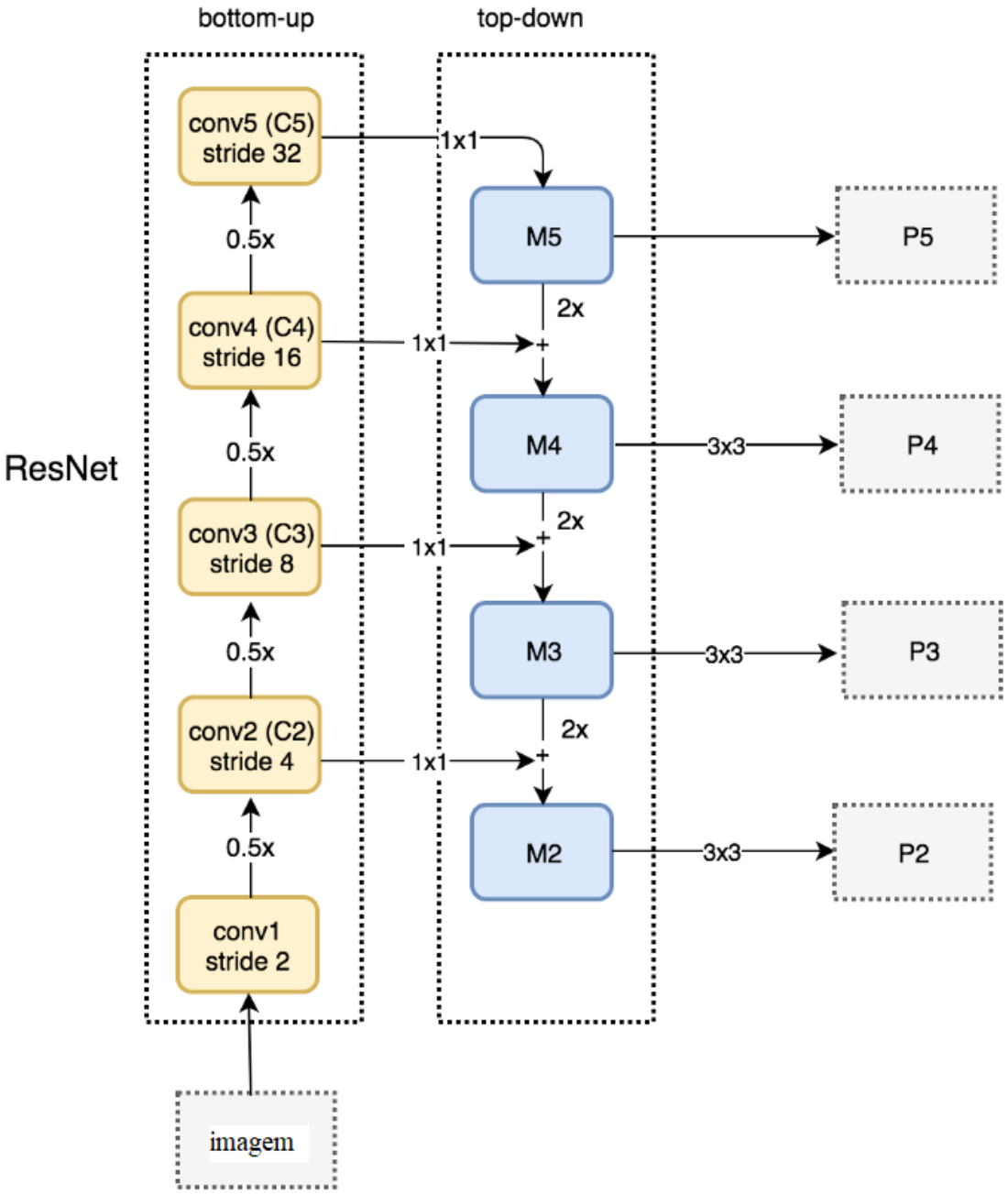


Figura 3.12 – Processo de geração dos mapas de características piramidais da FPN.

Fonte: Repositório digital [1]

[1] Disponível em: https://medium.com/@jonathan$_hui/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c.Acessoemjulho2019$

3.2.2 Rede para Detecção de Pessoas

Como o modelo final é a unificação de vários modelos em um só, para a tarefa de detecção de pessoas é acoplado uma rede ao *backbone*. Essa rede é a RetinaNet onde Lin e colaboradores propuseram uma nova função de perda chamada *focal loss* capaz de lidar com o problema de classes desbalanceadas em treinamentos e detectores de objetos (LIN; AI; DOLL, 2018). Além de rápida e de boa acurácia, a RetinaNet é compatível com o *backbone*, aplicando a rede FPN aos três últimos blocos residuais C_3, C_4 e C_5 da *ResNet*.

Lin, Ai e Doll (2018) não trabalharam com o mapa P_2 por motivos de custo computacional, mas desenvolveram dois novos mapas P_6 e P_7. P_6 é obtido via convolução 3×3 com *stride* 2 em C_5 e P_7 é computada aplicando-se uma função de ativação ReLU (AGARAP, 2018) seguida de uma convolução 3×3 com *stride* 2 em P_6. Segundo os autores essa modificação melhorou a performance mantendo a acurácia. O modelo então realiza operações convolucionais para classificação de objetos e regressão de *bounding box* a partir da saída do *backbone*.

A parte de classificação é uma rede totalmente convolucional anexada a cada nível da FPN. São quatro camadas convolucionais 3×3 com 256 filtros, seguidas da função de ativação ReLU. Uma útlima camada convolucional 3×3 é aplicada contendo $K \times A$ filtros, seguida da funcão de ativação sigmóide, onde K é o número de classes no treinamento; A é o número de *anchors*, caixas de diferentes tamanhos dispostas na imagem para a saber a probabilidade de existir um objeto dentro; e a função sigmóide dada por:

$$f(x) = \frac{1}{1 + e^{-\lambda x}} \tag{3.1}$$

A parte de regressão é idêntica a de classificação, exceto pela última camada constituída de uma convolução 3×3 com $4A$ filtros, ou seja, para cada *anchor* teremos o valor de quatro pontos (x_1, y_1, x_2, y_2), as coordenadas da caixa delimitadora (*bouding box*) do objeto detectado.

Durante o tempo de inferência, para cada imagem de entrada, existe $\sum_{l=3}^{7} W_l \times H_l \times A$ *anchor boxes* para cada nível de FPN. Para cada *anchor box* a rede classifica a probabilidade de o objeto pertencer a cada classe treinada e prediz 4 números indicando as coordenadas da caixa delimitadora final. Dado o grande número de *anchor boxes*, um objeto pode ser predito por múltiplas caixas. Para remover essa redundância, utiliza-se *non-maximum*

suppression, que por iteração escolhe a caixa com maior confiança (probabilidade) e elimina todas as outras que apresentam uma interseção de área maior que 0.5.

3.2.3 Rede para Detecção de Keypoints

Para detecção de partes do corpo humano, é acoplada uma outra rede ao *backbone*, predizendo o mapa de confianças onde cada mapa é a representação 2D da localização de determinada parte em certo pixel. A localização destes *keypoints* é representada como picos de Gauss chamados de *heatmaps*, onde cada *heatmap layer* pertence a uma classe de *keypoint* (nariz,ombro,pulso,joelho..). Seja uma imagem com k pessoas, deve-se existir j mapas com k picos, onde j é o número de partes do corpo humano a serem preditas.

A rede de detecção de *keypoints* obtém quatro *pyramidal features* $P_2 - P_5$ dos últimos quatro blocos $C_2 - C_5$ da ResNet reduzindo sua profundidade para 256. Para $i \in \{2, ..., 5\}$, nós aplicamos:

$$D_i = \phi(\phi(P_i)) \tag{3.2}$$

onde ϕ é uma camada convolucional com kernel 3×3 e 128 filtros. As camadas D_3, D_4 e D_5 são reescalonadas por 2,4 e 8 respectivamente para igualar o tamanho espacial de D_2 e todas são concatenadas em um *feature map* Z de 512 de profundidade, suavizado por uma camada convolucional 3×3 com ativação $ReLU$. Finalmente, o *heatmap* é obtido por uma convolução 1×1 com $(j + 1)$ filtros, onde o $+1$ é uma camada de *background* contendo todos *keypoints*.

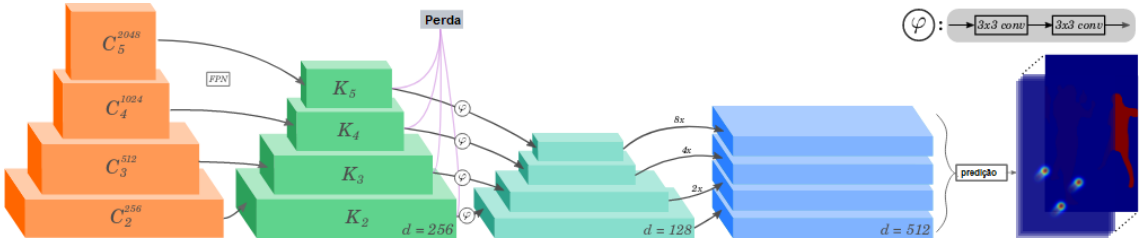


Figura 3.13 – Estrutura da rede para detecção de*keypoints*.

Fonte: (KOCABAS; KARAGOZ; AKBAS, 2018)

3.2.4 Rede para Correção de Poses - *Pose Residual Network*

Com o conteúdo desenvolvido até aqui, é possível detectar pessoas e obter a localização das partes dos corpos dessas pessoas e fazer uma associação simples. Porém, a associação dos *keypoints* a uma pessoa é um dos principais problemas da metodologia *bottom-up*, principalmente em situações em que há sobreposições de pessoas e consequentemente das suas *bounding-boxes* como mostrado na Figura 3.15.

A Figura 3.14 mostra o fluxo do algoritmo construído até aqui, onde compartilhando o mesmo *backbone*, há duas subredes, uma para detecção de *keypoints* e outra para detecção de pessoas. As duas saídas podem ser combinadas para associar os pontos dentro de uma *bounding box* a determinada pessoa.

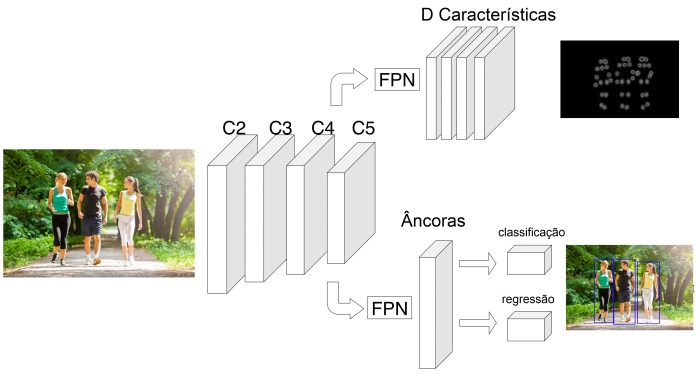


Figura 3.14 – Modelo desenvolvido até aqui.

Fonte: Igor Soares, 2019

Figura 3.15 – Sobreposição de caixas delimitadoras.

Fonte: Adaptado de (KOCABAS; KARAGOZ; AKBAS, 2018)

Pensando nisso, Kocabas, Karagoz e Akbas (2018) propuseram como solução uma rede residual capaz de aprender a estrutura de poses humanas conhecidas a partir do próprio *dataset* de treinamento. Eles chamaram esta rede de *Pose Redisual Network*(PRN). Assim, para cada *keypoint* j temos como entrada da rede $\boldsymbol{X} = \{x_1, x_2, \ldots, x_j\}$ onde $x_j \in R^{W \times H}$. Nós esperamos como saída $\boldsymbol{Y} = \{y_1, y_2, \ldots, y_j\}$ onde $y_j \in R^{W \times H}$, contendo as posições corretas dos *keypoints*. O método é descrito por:

$$y_j = \psi_j(\boldsymbol{X}) + x_j \tag{3.3}$$

onde ψ_j é uma correção no *keypoint* j. A Expressão 3.3 é implementada com uma rede residual de múltiplas camadas e a saída possui como ativação uma função *Softmax*, predizendo a probabilidade daquele *keypoint* de determinada classe pertencer ao conjunto total (pose).

Para acoplar essa rede no modelo desenvolvido até aqui (Figura 3.14), seja a saída da parte de detecção de *keypoints* $\boldsymbol{DK}$ e a saída da parte de detecção de pessoas $\boldsymbol{DP}$,a dimensão de $\boldsymbol{DK}$ é dada por $W \times H \times K$, onde W e H são as dimensões da imagem e K é o número de *keypoints* a serem preditos, e a dimensão de $\boldsymbol{DP}$ é dada por $N \times w \times h \times$, onde N é o número de pessoas detectadas; w e h são a largura e altura da *bounding box* com $w, h \in R^{W \times H}$, o seguinte algoritmo foi desenvolvido:

Algoritmo 3.1 Algoritmo de entrada para a rede residual de poses

1: Entrada: $\boldsymbol{DK}$ e $\boldsymbol{DP}$
2:
3: **for** **i** de 1 a N **do**
4: recorta $\boldsymbol{DK}$ nas coordenadas de $\boldsymbol{DP_i}$
5: reescalona o resultado para 56×36
6: aplica como entrada em 3.3
7:
8: **end for**
9: Saída: poses estimadas nas dimensões $N \times 56 \times 36 \times K$ =0

3.2.5 Reconhecimento de Atividade Através da Pose

Reconhecer uma ação pela pose da pessoa significa identificar padrões na disposição das partes do corpo que remetem a determinada atividade. Para o modelo ter essa

capacidade, foi adicionado uma rede MLP à saída da pose de cada indivíduo, onde as posições 2D de cada *keypoint* servem de entrada para o classificador.

Para cada instância de pessoa e seus respectivos *keypoints* detectados, a posição 2D das articulações é normalizada entre 0 e 1,e encadeada de forma a obter um vetor $[0_x, 0_y, 1_x, 1_y..., j_x, j_y]$ de tamanho $2j$, onde j é o número total de articulações. O vetor é a entrada de um modelo MLP com duas camadas intermediárias com 3072 e 128 neurônios cada e função de ativação $ReLU$, seguida de uma camada de saída com A neurônios e função de ativação *Softmax*, onde A é o número de classes de atividades a serem classificadas. Entre todas camadas densas, é adicionado uma camada de *BatchNormalization* (IOFFE; SZEGEDY, 2014) e *Dropout* (HINTON, 2014) com fator 0.3.

O modelo final (Figura 3.16) capaz de detectar pessoas, estimar a pose de cada pessoa e classificar suas atividades possui três saídas: a primeira saída de dimensões $N \times 4$ contendo as coordenadas da caixa de N pessoas; a segunda saída de dimensões $N \times 2 \times j$ contendo as coordenadas 2D de j *keypoints* de todas pessoas; e a terceira saída com a classificação da atividade de dimensões $N \times A$ contendo as probabilidades da atividade de N pessoas em A classes diferentes.

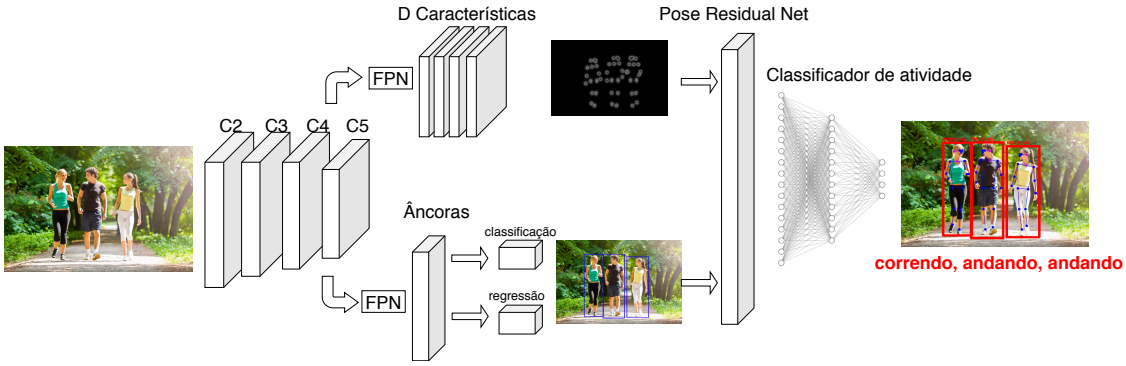


Figura 3.16 – Diagrama do modelo final.

Fonte: Igor Soares, 2019

4 Experimentos e Resultados

Neste capítulo é apresentado todos os testes e resultados obtidos com a metodologia descrita. Todos submodelos são analisados um por um, assim como o modelo completo final. Os resultados das tarefas de detecção de *keypoints* e detecção de pessoas são comparados a outros trabalhos em nível de estado da arte. O modelo final de classificação de pose 2D não possui trabalho semelhante e por isso não há como ser comparado, ao invés disso são apresentados os resultados de diferentes arquiteturas que levaram a escolha da melhor. Os *datasets* utilizados e desenvolvido neste trabalho são discutidos a seguir.

4.1 Detalhes de Implementação

A metodologia descrita foi toda implementada em Python utilizando-se as bibliotecas Keras (CHOLLET *et al.*, 2015) e Tensorflow (ABADI *et al.*, 2015), exceto a parte da RetinaNet, que contém uma implementação *open-source* disponibilizada por Gaiser *et al.* (2018)[1]. O código foi adaptado com algumas partes reescritas para encaixar a parte de detecção de pessoas ao resto do modelo. Detalhes são discutidos a seguir.[2]

4.1.1 Linguagem Utilizada e Ferramentas

Python

Python é uma linguagem interpretada, de alto nível, de proposito geral, multiparadigma, suportando programação funcional, orientada a objetos e procedural (KUHLMAN, 2011). Possui tipagem dinâmica e seu intuito é produzir códigos limpos, com pouca verbosidade e bem organizados, devido a identação obrigatória.

Graças a essas características, Python atualmente é a principal linguagem para desenvolvimento de aprendizado de máquina (CHOLLET, 2017). Entretanto, é importante notar a queda de performance ao se usar puramente Python, já que é uma linguagem interpretada, situação ruim para algoritmos que necessitam de grande processamento.

[1] Disponível em: https://github.com/fizyr/keras-retinanet/tree/master/keras$_r etinanet$

[2] O código completo da metodologia deste trabalho está disponível em: https://github.com/IgorMunizS/MultiPoseID

Devido ao interpretador do Python ser escrito em C/C++ (outra linguagem bastante utilizada e eficiente para *machine learning*), é possível integrar códigos em C++ com o Python através de bibliotecas. Com isso mantém-se as qualidades do Python de desenvolvimento rápido e código limpo com o rápido tempo de execução do C++.

Tensorflow

Tensorflow é uma biblioteca *open-source* para fluxo de dados (*dataflow*) e programação diferenciável. Desenvolvida pelo time da Google, o Google Brain, é uma biblioteca de matemática simbólica, com foco em aprendizagem de máquina e atualmente aprendizagem profunda. [3].

Seu código é escrito com uma mistura de C++ altamente otimizado junto com CUDA, uma plataforma de computação paralela desenvolvida pela Nvidia para processamento de códigos utilizando a GPU (NICKOLLS *et al.*, 2008). Entretanto fornece APIs de acesso em Python e C++ convencional, sendo utilizada a do Python neste trabalho.

A complexidade da programação simbólica com geração de grafos em *dataflow* para criação de redes profundas com centenas de camadas torna o código do Tensorflow difícil de dar manutenção e de procurar por erros. Com isso surgiu o Keras, uma API de alto nível que utiliza o Tensorflow como *backend*, com programação sequencial e mais abstrata. Entretanto, o Keras não oferece o nível de profundidade do Tensorflow e para algumas atividades é necessário utilizá-lo diretamente.

Keras

Keras é uma API de alto nível para redes neurais escrita em Python. É capaz de rodar sobre outras bibliotecas de *deep learning* como Tensorflow, CNTK[4] e Theano[5]. Foi desenvolvida para produzir resultados rápidos, tendo como principais características a interface amigável, modularização de estruturas de redes neurais permitindo a criação de modelos sequencialmente, facilidade em extender modelos e ser escrita em Python.[6]

[3] Informações disponíveis em: https://www.tensorflow.org/about/
[4] https://github.com/Microsoft/cntk
[5] https://github.com/Theano/Theano
[6] Informações disponíveis em: https://keras.io/

A maior parte do trabalho foi desenvolvido utilizando-se o Keras e essa escolha é justificada pela velocidade com que é possível desenvolver novas arquiteturas de redes neurais, permitindo ainda a integração com o Tensorflow para desenvolvimento de estruturas novas (não presentes nos módulos do Keras).

O Keras juntamente com o tensorflow são os *frameworks* com maior poder de desenvolvimento como indica a Figura 4.1 e por isso os mais utilizados na indústria e na comunidade científica, destacando-se também o Pythorch em constante crescimento.

Para o cálculo dessa pontuação, adquiriu-se os dados de número de vagas de emprego, de citações em artigos, de livros publicados, de projetos *open-source* e suas contribuições dos principais *frameworks* de *deep learning* disponíveis. O resultado mostra quais são os *frameworks* preferidos pela indústria e a comunidade científica em um contexto geral.

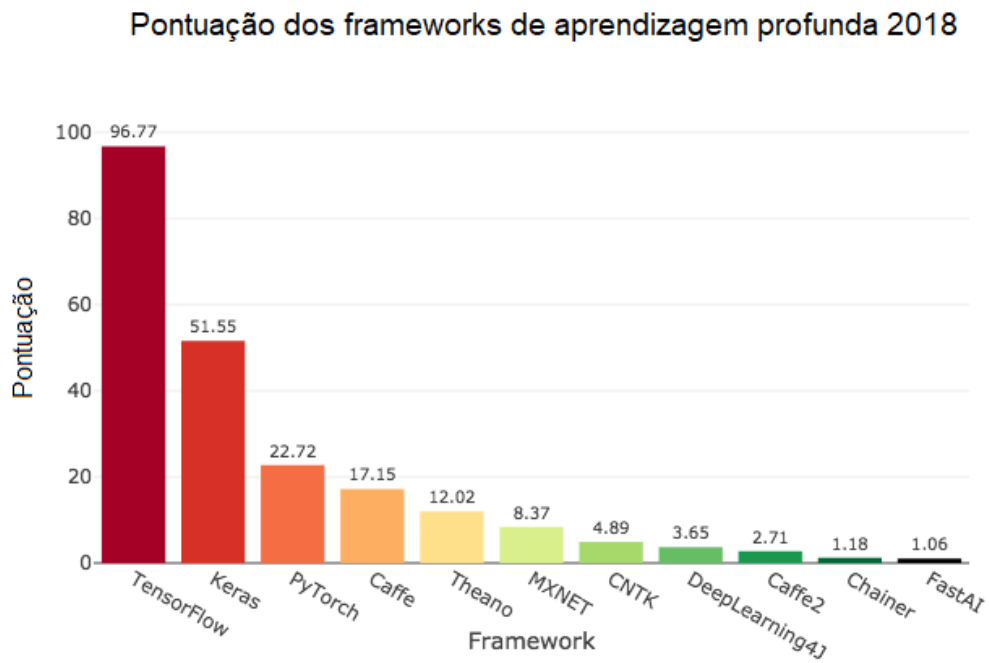


Figura 4.1 – Os *frameworks* de *deep learning* mais poderosos.

Fonte: Jeff Hale, 2018[7]

No apêndice B é apresentado alguns esboços de implementação necessários para reprodução do trabalho. Para o desenvolvimento completo da metodologia, os tópicos apresentados no capítulo anterior foram implementados parte a parte, reaproveitando o que há disponível *open-source*, no caso, a rede de detecção de objetos RetinaNet. O

[7] Disponível em: https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a. Acesso em julho 2019

sumário da arquitetura completa final pode ser vista no apêndice C e o código completo https://github.com/IgorMunizS/MultiPoseID.

4.1.2 Treinamento

Hardware Utilizado

Todas as partes (detecção de *keypoints*, detecção de pessoas, PRN e classificação de atividade) são treinadas separadamente devido as diferentes funções de perda em cada treinamento. O treinamento foi realizado em uma única GPU Nvidia Tesla V100 e durou, em média[8], aproximadamente 93 horas para treinar todos os submodelos. É importante notar que o tempo de treinamento é influenciado não apenas pela GPU, mas por outros componentes como o processador e o disco de armazenamento que são bastante utilizados durante a fase de carregar e preprocessar os dados de entrada. As especificações da GPU e do computador completo estão nas Tabelas 4.1 e 4.3, respectivamente.

Tabela 4.1 – Especificações da Nvidia Tesla V100[9]

	Tesla V100
CUDA Cores	5120
Tensor Cores	640
Double Precision	7 teraFLOPS
Single Precision	14 teraFLOPS
Deep Learning	112 teraFLOPS
Memory Bandwith	900GB/s
Memory Capacity	16GB

Tabela 4.2 – Especificações do computador utilizado no trabalho.[10]

	Tesla V100
Processador	8vCPUS
Memória RAM	24GB
GPU	Tesla V100
Armazenamento	SSD 64 GB

[8] Houveram vários testes, inclusive com mudanças na profundidade do backbone resultando em mais ou menos tempo de treinamento. Esse valor é a média do tempo de todos os treinamentos.

[9] Retirado de: https://www.nvidia.com/pt-br/data-center/tesla-v100/

[10] Máquina alugada no Google Cloud : https://cloud.google.com/

A rede convolucional final completa é então obtida com a junção de todas essas partes, carregando os pesos finais obtidos em cada treinamento, associando os valores a cada camada por seus respectivos nomes.

Ordem de treinamento

Rede de detecção de keypoints

Primeiro, deve-se treinar a rede para detecção de *keypoints* devido ao compartilhamento do *backbone* entre esta rede e a de detecção de pessoas, sendo preferível treinar todos os parâmetros na tarefa com mais complexidade de otimização. Para essa tarefa, usa-se como entrada recortes de imagens de dimensões 480×480 e centralizados na principal pessoa da cena. Para o cálculo do erro e atualização dos pesos, é gerado como *groudtruth* (o valor desejado na saída) os mapas de confiança a partir das anotações contidas no *dataset* das posições 2D dos *keypoints*. Cada mapa de confiança possui um pico de Gauss em determinada parte do corpo humano para k pessoas em uma cena. Seja $\mathbf{S}^*_{j,k}$ um mapa de confiança para o *keypoint* j da pessoa k, nós aplicamos a função de Gauss na posição da coordenada $\mathbf{x}_{j,k}$ e obtemos a confiança de uma localização $\mathbf{p}$ ser a junta com:

$$\mathbf{S}^*_{j,k} = \exp(-||\mathbf{p} - \mathbf{x}_{j,k}||^2/\sigma^2), \tag{4.1}$$

onde σ é um valor para controlar a propagação do pico.

A Figura 4.2c exemplifica uma imagem de entrada (a) e seus respectivo *heatmap* ou *ground truth* para o ponto chave "nariz"(b). Também é mostrado o *background* (c) com a junção de todos os pontos de todas pessoas presentes na cena.

(a) (b) (c)

Figura 4.2 – Imagem de entrada e heatmap para treinamento. (a) Imagem de entrada; (b) Mapa de confiança para o nariz. (c) Junção de todos os mapas de confiança para demonstração da localização de todos os pontos.

Fonte: Igor Soares,2019

Os dados são aumentados em tempo de treinamento, aplicando-se transformações na imagem para gerações de novas entradas a partir de uma única. Essa técnica, chamada de *data augmentation*, é essencial para a capacidade de generalização da rede convolucional que, por padrão, depende de muitos dados durante o treinamento. É em tempo de treinamento pois a cada iteração aplica-se uma variação na entrada, sem necessidade de salvar o resultado em disco. Para isso, utiliza-se rotações randômicas entre ± 40 graus, variação de escala entre $0.7 - 1.2$ e giro vertical com probabilidade de 30%. O treinamento é inicializado com os pesos do *backbone* pré-treinado no *dataset ImageNet* (DENG *et al.*, 2009) para facilitar a classificação de pessoas. Todas entradas são pré-processadas centralizando as cores em zero em cada canal (BGR).

(a) (b) (c)

Figura 4.3 – Exemplos de novas entradas geradas com *data augmentation*.

Fonte: Igor Soares, 2019

Para atualizar os pesos, deve-se usar a função de perda L_2 entre as predições e os mapas *groud truth*. Uma supervisão intermediária também foi aplicada a saída das características piramidais P para evitar a explosão do gradiente (*vanishing problem*) (WEI, 2016). Para $i \in \{2, ..., 5\}$ obtem-se Z_i aplicando uma convolução 2D com j filtros em P_i, onde j é o número de *keypoints*. As camadas $Z_3 - Z_5$ são reescalonadas para obterem o mesmo corpo do mapa predito. A função de perda final é dada por:

$$f = \mathbf{W} \cdot ||H_p - S_j^*||_2^2 + \sum_{i=2}^{5} \mathbf{W} \cdot ||Z_i - S_{j,k}^*||_2^2 \tag{4.2}$$

onde H_p é a saída da rede e $\mathbf{W}$ é uma máscara para eliminar pessoas nas cenas em que não há anotações dos pontos.

O modelo é otimizado com a função Adam (KINGMA; BA, 2015) com taxa de aprendizado inicial de $4e^{-5}$. Essa taxa é atualizada de acordo com Expressão4.3

$$Lr_n = 4e^{-5} \times 0.333^{n/2}, \tag{4.3}$$

onde $n \in N^*$ é o número da época de treinamento.

Rede de Detecção de Pessoas

A parte de detecção de pessoas é uma modificação da versão open source Keras RetinaNet(GAISER *et al.*, 2018). As principais modificações são a utilização do *backbone* aqui implementado que, apesar de possuir as mesmas características (ResNet + FPN) do utilizado pelos criadores da RetinaNet (LIN; AI; DOLL, 2018), deve possuir as camadas com o mesmo nome das camadas do *backbone* utilizado na rede de detecção de *keypoints* para ser possível o compartilhamento no modelo final, e alteração da última camada para lidar apenas com a detecção de um objeto (pessoas).

Parte da rede, mais precisamente as camadas do *backbone*, são inicializadas com os pesos resultantes do treinamento de detecção de *keypoints* e em seguida a atualização de pesos dessas camadas é congelada. Isso garante o compartilhamento do *backbone* entre as duas redes sem alterar os resultados do primeiro treinamento, forçando a RetinaNet a aprender a detectar objetos com as características extraídas em outro problema de semântica parecida (KOCABAS; KARAGOZ; AKBAS, 2018). O compartilhamento é fundamental para um modelo multitarefa, permitindo duas subredes utilizarem a mesma entrada e suas características, além de resultar em menor tempo de inferência. O congelamento é necessário pois se houvesse alteração no *backbone* no treinamento de detecção de pessoas, os pesos das outras camadas obtidos no treinamento anterior não teriam mais relação com o problema.

As imagens de entrada são redimensionadas para que seu menor lado tenha 800 pixels e a função de custo para treinamento é a descrita por Lin, Ai e Doll (2018), chamada de *Focal Loss* e descrita em 4.4.

$$FL(p_t) = -\alpha_t(1-p_t)^{\gamma} log(p_t) \tag{4.4}$$

Esta função é formulada para abaixar o peso de exemplos fáceis e focar em exemplos difíceis de serem treinados. P_t é a probabilidade estimada de ser uma classe, e quando sua certeza está próxima de 1, o valor resultante é pequeno ajustando poucos os pesos; γ é o parâmetro de foco da função voltado para exemplos mais difíceis e α é um *offset* para tratar classes desbalanceadas baseado no número de exemplos.

Neste segundo treinamento também utiliza-se o otimizador Adam, com taxa de aprendizagem inicial de $1e^{-5}$ e com fator de redução de 0.1 *on plateau*, significando que toda vez que não houver diminuição do erro entre duas épocas, a taxa de aprendizagem é divida por 10.

Rede Residual de Poses

A *Pose Residual Network* é treinada com os dados de *ground truth* do próprio *dataset*. Para cada instância de *bounding boxes*, nós geramos um *heatmap* de entrada de tamanho fixo 36×56 contendo todos os *keypoints* delimitados por aquela caixa e um *heatmap* de saída, do mesmo tamanho do de entrada, com apenas os *keypoints* da pessoa em questão. Dessa forma, a rede aprende a identificar poses reais a partir de dados reais já anotados, excluindo pontos que não fazem parte de uma pose em questão.

Durante o treinamento utiliza-se o otimizador Adam, com taxa de aprendizagem de $1e^{-3}$ e com fator de redução de 0.5 *on plateau*, e função de perda *binary crossentropy*.

Um exemplo de correção de pose com essa rede residual pode ser visualizado na Figura 4.4, em que para cada recorte da imagem onde é detectado uma pessoa (à esquerda), tem-se como entrada a mesma região delimitada no *heatmap* dos *keypoints* (centro). A saída esperada é uma pose que contenha os pontos apenas da pessoa desejada (à direita).

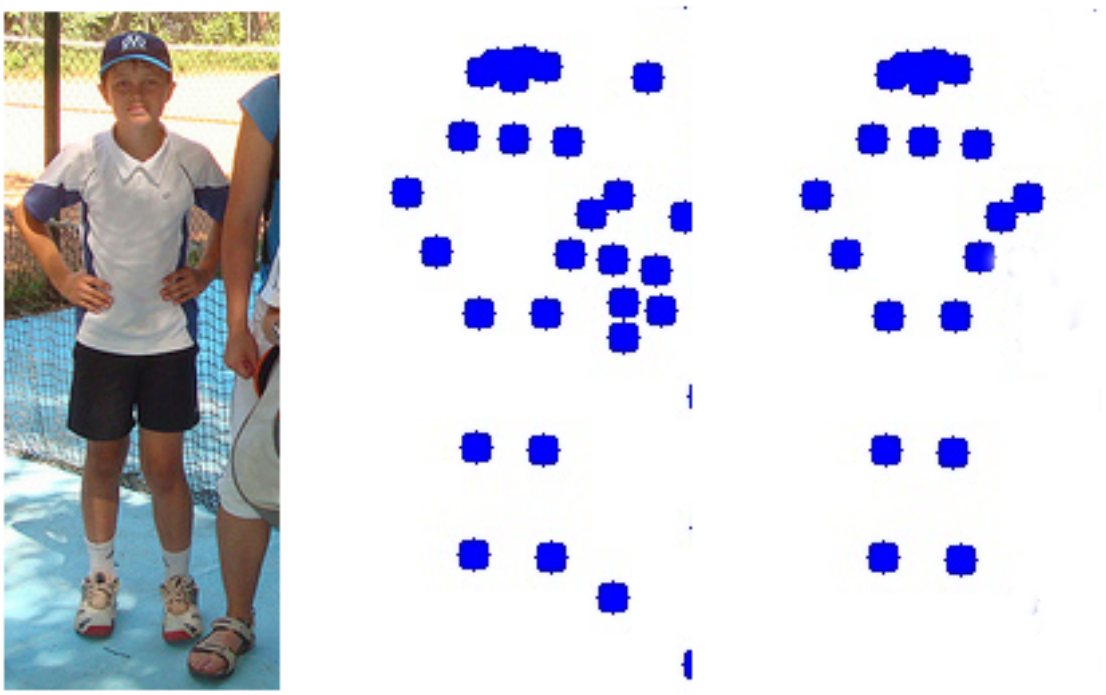


Figura 4.4 – Correção em pose realizada pela PRN.

Fonte: Igor Soares, 2019

Classificador de Atividade Baseado na Pose

Para o treinamento do classificador de atividades, extraiu-se a pose (coordenadas de articulações) de diversas pessoas realizando determinadas atividades. Para isso, foi utilizado os modelos já treinados até aqui. Essas coordenadas são a entrada do modelo supervisionado, onde a saída é a atividade. A posição 2D das articulações é normalizada entre 0 e 1,e encadeada de forma a obter um vetor $[0_x, 0_y, 1_x, 1_y..., j_x, j_y]$ de tamanho $2j$, onde j é o número total de articulações.

O treinamento também é feito com otimizador Adam e taxa de aprendizagem inicial de $1e^{-3}$ com fator de redução 0.1 e função de perda *categorical crossentropy*.

A pose de estimada de cada pessoa serve como entrada para o classificador de atividade, sendo esta pré-processada para o mesmo formato utilizado no treinamento. A saída é a classe com maior probabilidade na última camada da rede e é comparada a classe desejada para o cálculo do erro e execução do treinamento.

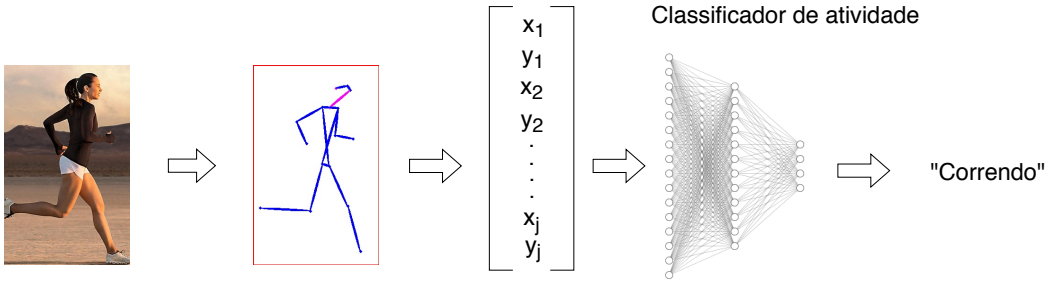


Figura 4.5 – Entrada e saída do classificador de atividade.

Fonte: Igor Soares, 2019

4.2 Datasets Utilizados

Datasets em aprendizagem de máquina são uma coleção de dados que podem conter anotações ou supervisões para o treinamento de um algoritmo para realização de determinada tarefa. Em visão computacional, os *datasets* são geralmente constituídos de grandes quantidades de imagens dos mais variados tipos, contendo anotações sobre classes de objetos, localização de objetos e outras informações presentes em cada imagem do conjunto. Para o desenvolvimento do estudo aqui realizado, foram utilizados três *datasets*: o MS COCO Dataset 2017 (LIN; ZITNICK; DOLL, 2014), o MPII Human Pose (ANDRILUKA *et al.*, 2014) e um terceiro *dataset* desenvolvido também neste trabalho para o treino do classificador de poses.

4.2.1 MS COCO Dataset 2017

COCO Dataset é um conjunto de dados de grande escala para as tarefas de detecção de objetos, segmentação de cenas, detecção de *keypoints* e reconhecimento em contexto. É patrocinado pela Microsoft e o Facebook, e anualmente realiza competições na execução das possíveis tarefas deste *dataset*. Possui mais de 330 mil imagens, sendo mais de 220 mil delas anotadas com as informações presentes. Dentre essas informações, existem aproximadamente 1.5 milhão de objetos em 80 classes diferentes e mais de 250 mil pessoas com as coordenadas das partes do seus corpos especificadas.

Dada a semântica do problema, apenas as imagens contendo pessoas são interessantes para as etapas de treinamento e de teste. Assim, utilizou-se um subconjunto do *dataset* contendo apenas imagens com pessoas e limitou-se as anotações de apenas uma única

classe (*person*). Esse subconjunto é divido em um conjunto de treinamento, contendo $64k$ imagens com mais de $240k$ instâncias de pessoas, e um conjunto de validação com 2693 imagens contendo pessoas.

As métricas utilizadas para cálculo da precisão dos modelos são as oficiais da competição COCO e as mesmas utilizadas em outros trabalhos, *average precision* (AP) de OKS (*object keypoint similarity*), para detecção de *keypoints*, e *average precision* de IoU (*intersection over union*) para detecção de pessoas. Todas as métricas são discutidas na seção 4.3.

4.2.2 MPII Human Pose

MPII Human Pose é um *dataset* para avaliação de modelos de estimação de pose por partes do corpo. Contém cerca de 25 mil imagens com mais de 40 mil pessoas com suas informações anotadas. É um *dataset* mais antigo que o MS COCO, porém possui uma informação relevante para o trabalho: as imagens foram coletadas de atividades específicas do cotidiano. Ao todo, são 410 atividades humanas. Esse *dataset* não foi utilizado no treinamento e nem nos testes, mas foi importante para o desenvolvimento do *dataset* de treinamento do classificador de atividades.

4.2.3 Dataset Próprio para Reconhecimento de Atividade por Pose

Para a tarefa de reconhecimento de atividade pela pose estimada foi preciso construir um *dataset* próprio contendo poses associadas à categorias de atividades. Para isso, extraiu-se a pose de pessoas presentes em cenas categorizadas por atividade com o modelo de estimação de pose desenvolvido e treinado neste trabalho.

O *MPII Human Pose dataset* (ANDRILUKA *et al.*, 2014) serviu como base para imagens de atividades, complementando com imagens obtidas na internet. Foi coletado um total de 1204 imagens com mais de $3k$ de pessoas realizando quatro atividades: andando, correndo, sentado e parado. Cada atividade tem pelo menos 500 exemplos de poses.

Para construção do *dataset* foi necessário separar manualmente as cenas de acordo com a atividade que as pessoas estavam executado. Em seguida, detectou-se cada pessoa presente na imagem, recortando-as de acordo com a *bounding-box*. Então, cada pessoa tem

Tabela 4.3 – Quantidade de poses por classe do dataset próprio

	Quantidade
Andando	921
Correndo	789
Sentado	502
Parado	1083

sua pose estimada e associada a atividade previamente categorizada. A pose serve como entrada para o treinamento do classificador e a atividade como a saída desejada.

A Figura 4.7 exemplifica imagens que são partes do *dataset* desenvolvido para treinamento da rede de reconhecimento de atividade por pose. Todas poses foram obtidas utilizando-se o estimador de pose construído aqui. Da esquerda para a direita, de cima pra baixo, tem-se as poses associadas as seguintes atividades: correndo, sentado, parado, andando e andando (diferente perspectiva)

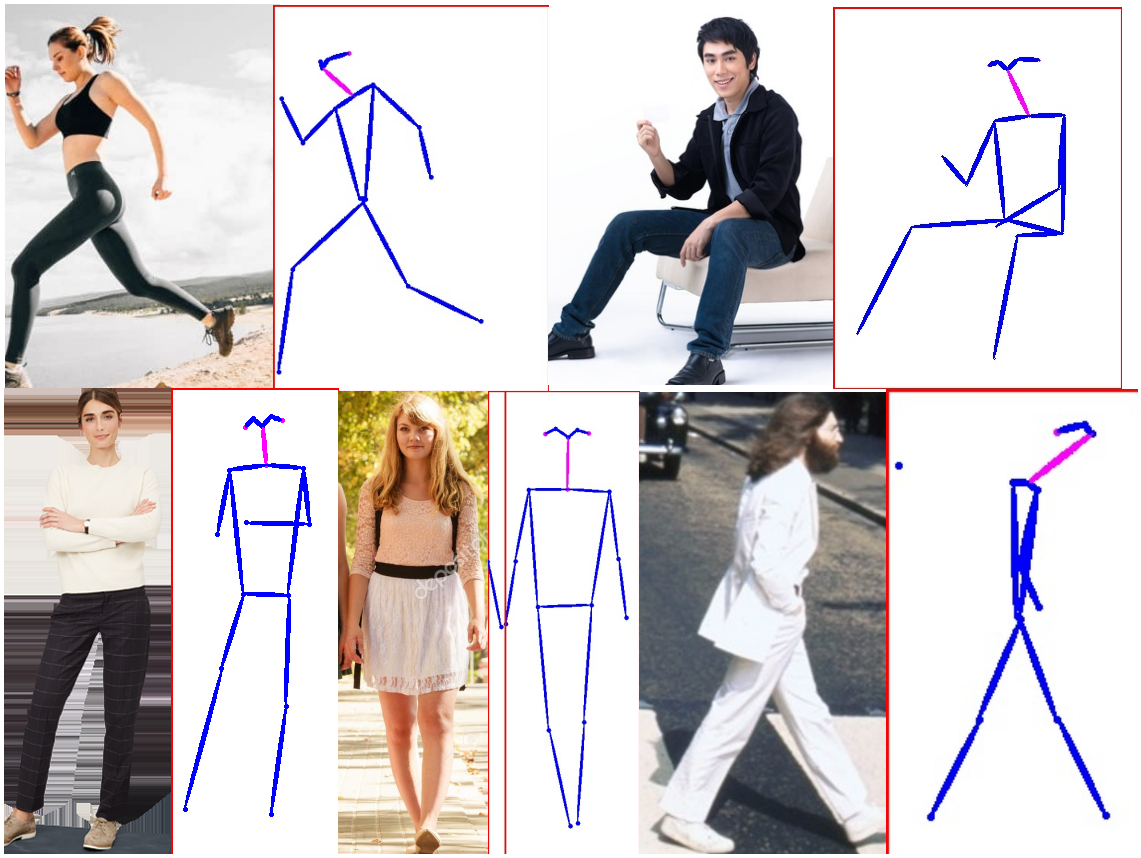


Figura 4.6 – Poses geradas para o *dataset*.

Fonte: Igor Soares, 2019

4.3 As métricas

Métricas são números utilizados como medidas de desempenho, permitindo a comparação de resultados em diferentes metodologias e/ou períodos de tempos. Para as tarefas de detecção de objetos, convencionou-se utilizar a precisão média dos resultados (*average precision*) sobre a área a de intersecção entre a saída e a resposta desejada. Já para a detecção de pontos ou partes do corpo humano, utiliza-se a precisão média da similaridade entre os pontos obtidos e os desejados. Todas as métricas utilizadas para mensurar os resultados são discutidas a seguir.

4.3.1 Precisão e Recall

Precisão e Recall são duas métricas comuns quando se quer julgar a performance de um modelo de classificação. Geralmente, elas são a base de outras métricas e por isso é importante entendê-las primeiro.

Precisão de uma determinada classe é a taxa de positivos verdadeiros em um conjunto de positivos na classe. Sua fórmula é dada por:

$$Prec = \frac{TP}{TP + FP} \tag{4.5}$$

onde TP é o numero de positivos verdadeiros e FP de falsos positivos para a classe.

De maneira semelhante, o Recall é a taxa de positivos verdadeiros de uma classe para todo o *ground truth*, ou seja, quanto foi predito corretamente em relação ao total que deveria ser predito.

$$Recall = \frac{TP}{TP + FN} \tag{4.6}$$

4.3.2 F1 Score

Essa métrica combina a precisão e o *recall* em único número, podendo ser entendida como uma média ponderada desses dois números. O F_1 score atinge valores entre 0 e 1, sendo 1 no melhor caso (precisão e *recall* perfeitos) e 0 no pior caso. Essa métrica é

utilizada nos testes do classificador de atividades pela pose para decidir qual a melhor arquitetura.

$$F_1 = 2\frac{Prec \times Recall}{Prec + Recall} \tag{4.7}$$

4.3.3 Precisão média (AP)

Ao se computar a precisão e o *recall* em várias posições de uma sequência, é possível plotar uma curva da relação precisão-*recall* $p(r)$ para todos os elementos. O *average precision* é a área abaixo dessa curva, sendo o valor médio de $p(r)$ no intervalo de 0 a 1.

$$AP = \int_0^1 p(r)dr \tag{4.8}$$

Na prática a integral pode ser calculada como a soma sobre todas as posições da sequência:

$$AP = \sum_{k=1}^{n} P(k)\Delta_r(k) \tag{4.9}$$

onde $P(k)$ é a precisão de na posição K e $\Delta_r(k)$ a variação do recall de $k-1$ para k.

Para detecção de pessoas, o *average precision* é calculado sobre o índice de Jaccard (*intersection over union*) para cada *bounding box* predita. Já para detecção de *keypoints* é calculado sobre o similaridade entre pontos de saída e desejado para cada ponto.

4.3.4 Intersecção Sobre a União (IoU)

A intersecção sobre a união, ou *intersection over union* em inglês, é medição da sobreposição de duas delimitações. É uma importante métrica para detecção de objetos, medindo-se quanto a caixa delimitadora (*bounding box*) predita é próxima da delimitação desejada (*ground truth*).

$$IoU = \frac{|A \cap B|}{|A \cup B|} \tag{4.10}$$

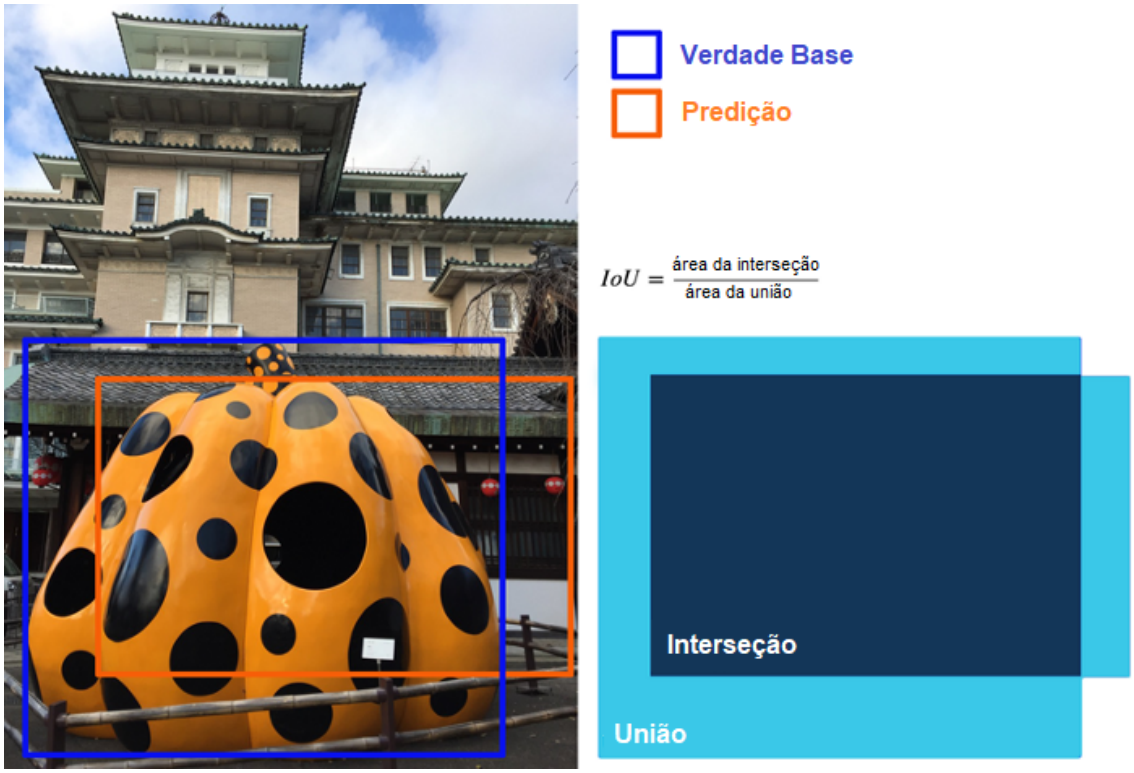


Figura 4.7 – Interseção sobre a união da região delimitada predita e da delimitação desejada.

Fonte: Repositório Digital[11]

Com o valor de IoU calculado, defini-se um valor limiar (*threshold*) para determinar se a predição é um positivo verdadeiro ou um falso positivo (geralmente esse valor é 0.5). Com os valores de positivo verdadeiro e falso positivo, calcula-se o *average precision* sobre todas IoU de todos objetos preditos na imagem. O mesmo processo é executado para todas as imagens de teste, obtendo-se uma média do *average precision* (mAP).

4.3.5 *Object Keypoint Similarity* (OKS)

Object Keypoint Similarity (similaridade dos keypoints de um objeto) é a métrica definida para avaliação dos modelos de detecção de *keypoints* no *COCO dataset*. A principal ideia dessa métrica é imitar as métricas da tarefa de detecção de objetos, *average precision* da medida de similaridade, neste caso a interseção sobre a união. Para isso, definiu-se uma medida de similaridade análoga à IoU, mas para *keypoints*.

Para cada objeto (pessoas neste caso), as coordenadas desejadas do pontos tem a forma $[x1, y1, v1, ..., xk, yk, vk]$, onde x, y são as localizações e v é uma variável de controle

[11] Disponível em: https://medium.com/@jonathan$_hui/map - mean - average - precision - for - object - detection - 45c121a31173$

indicando se este ponto é visível na imagem ou não. A saída do detector de *keypoints* deve conter a localização dos pontos e um valor de confiança daquilo que está sendo predito, possuindo o mesmo formato das coordenadas desejadas (*ground truth*).

A similaridade entre os pontos chaves de um objeto é definido como:

$$OKS = \frac{\sum_i e^{\frac{-d_i^2}{2s^2k_i^2}} \delta(v_i > 0)}{\sum_i \delta(v_i > 0)} \tag{4.11}$$

onde d_i é a distância euclidiana entre cada localização do *keypoint* predito e a correspondente localização desejada; v_i é a *flag* de controle da visibilidade do ponto, onde pontos não visíveis na imagem ($v_i = 0$) não entram no cálculo do OKS; e sk_i é o desvio padrão no qual o *keypoint* é passado em uma gaussiana não normalizada.

Análogo à intersecção sobre a união, o OKS tem valores definidos entre 0 e 1, sendo 1 o melhor valor (predição perfeita) e o 0 o pior valor (predição totalmente errada). Definindo-se um valor de *threshold* é possível então calcular o mAP dos testes.

A Figura 4.8 apresenta um exemplo de curva de precisão-*recall* em que o cálculo da precisão média de OKS é a área abaixo dessa curva.

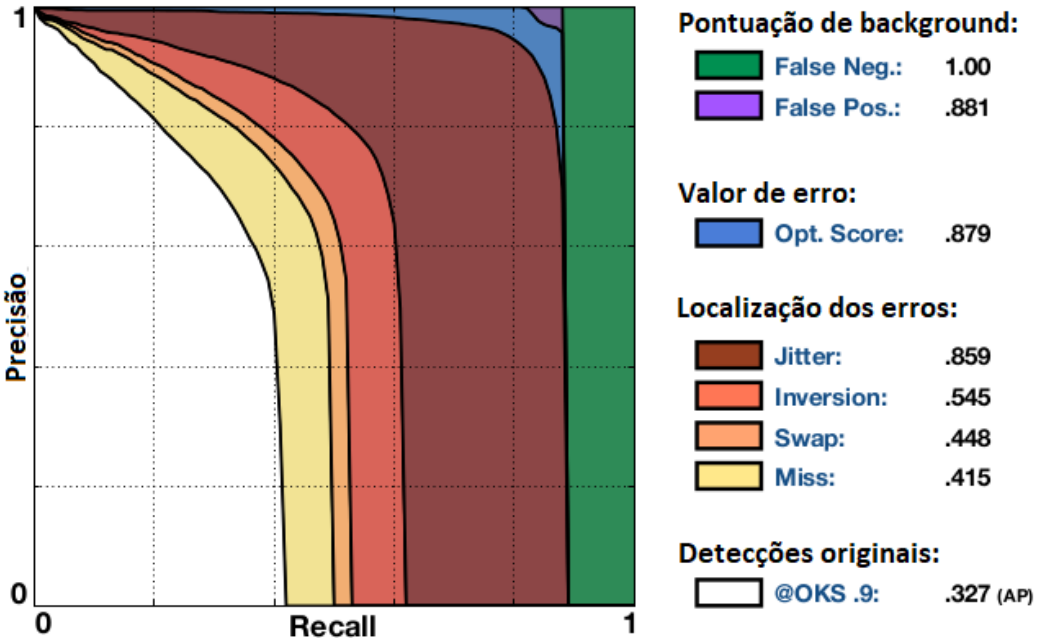


Figura 4.8 – Precisão média do OKS.

Fonte: (CAO *et al.*, 2017)

4.4 Experimentos, Resultados e Discussão

Ao longo do desenvolvimento do trabalho, vários experimentos foram realizados afim de se obter os melhores resultados possíveis. Não utilizou-se nenhum tipo de técnica de pesquisa de parâmetros, da qual consiste em realizar sucessivos testes automatizados alterando os valores dos hiperparâmetros das redes de forma aleatória em um intervalo de valores definidos. Apesar de ser uma técnica comum em aprendizagem de máquina, a mesma torna-se impraticável quando aplicada a aprendizagem profunda, devido ao tempo de treinamento dos algoritmos. Além disso, esse tipo de abordagem não leva em conta uma bagagem teórica, sendo preferida para processos automáticos, que não necessitam de grande conhecimento teórico e de menor complexidade de treinamento. Por isso, os experimentos foram realizados, em sua maioria, tomando como referência parâmetros já estudados e definidos na literatura como ótimos para as arquiteturas já existentes, sendo feitas algumas pequenas mudanças que serão comparadas adiante. Para o modelo final e a rede de classificação, foi testado, de forma consciente, uma variação de parâmetros para obtenção da melhor arquitetura.

4.4.1 Detecção de Pessoas e Estimação de Pose

Dados

Para o treinamento e avaliação dos modelos responsáveis pelas tarefas de detecção de pessoas e estimação de pose, utilizou-se o *MSCOCO Dataset 2017* discutido na seção 4.2. O próprio *dataset* possui uma divisão em um conjunto de treinamento, um conjunto de avaliação e um conjunto de teste, e essa divisão foi respeitada e seguida. Ambos conjuntos de treinamento e de avaliação possuem anotações de *ground truth*, diferenciando apenas pela quantidade de imagens. Já o conjunto de teste possui apenas as imagens propriamente ditas e é utilizado durante as competições onde apenas o comitê organizador possui as devidas respostas para o cálculo da pontuação final.

Ao avaliar um modelo de redes neurais, é necessário utilizar dados diferentes dos utilizados na fase de treinamento, possibilitando a verificação de *overfitting*, um problema causado quando a rede aprende características muito específicas e perde a sua capacidade de generalização. Por isso, o conjunto de avaliação deve possuir um número considerável de

imagens, todas diferentes das apresentadas à rede durante o treinamento, e com distribuição semelhante entre as classes do problema. Além disso, é importante que as imagens de validação também possuam anotações para o cálculo das métricas.

Sendo assim, por motivos de comparação com outros trabalhos, o *dataset* de treinamento usado foi o *MS COCO Dataset 2017 training set*, contendo mais de 64*mil* imagens, e para avaliação o *MS COCO Dataset 2017 validation set* com 2693 imagens. Não foi adicionado nenhum material exterior.

Diferentes Arquiteturas de ResNet

Utilizou-se arquitetura ResNet com o backbone compartilhado e com três diferentes tipos de profundidade: ResNet50, ResNet101 e ResNet152. Os números após "ResNet"indicam a quantidade de camadas presentes na rede, e como é de se esperar, quanto mais camadas, mais demorado o processo de treinamento, mas também um resultado melhor. Entretanto, Bianco *et al.* (2018) mostraram que a ResNet50 é a arquitetura que possui melhor custo-benefício quando comparado resultados por tempo de inferência, sendo essa a mais utilizada em pesquisas como base de outras arquiteturas e também o modelo padrão do código deste trabalho (é possível também selecionar como parâmetro outras profundidades para os testes).

Na Figura 4.9 é possível ver as diferenças entre os diversos tipos de ResNets, enquanto na Figura 4.10 é apresentado o resultado de Bianco *et al.* (2018) que realizaram testes com as principais arquiteturas de *deep learning* existentes, no *dataset* ImageNet, comparando a acurácia de cada uma pelo seu número de operações necessária para inferência (gigaflops)

Nome camada	Tamanho saída	18	34	50	101	152
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figura 4.9 – Diferenças entre as profundidades da ResNet.

Fonte: He, 2015

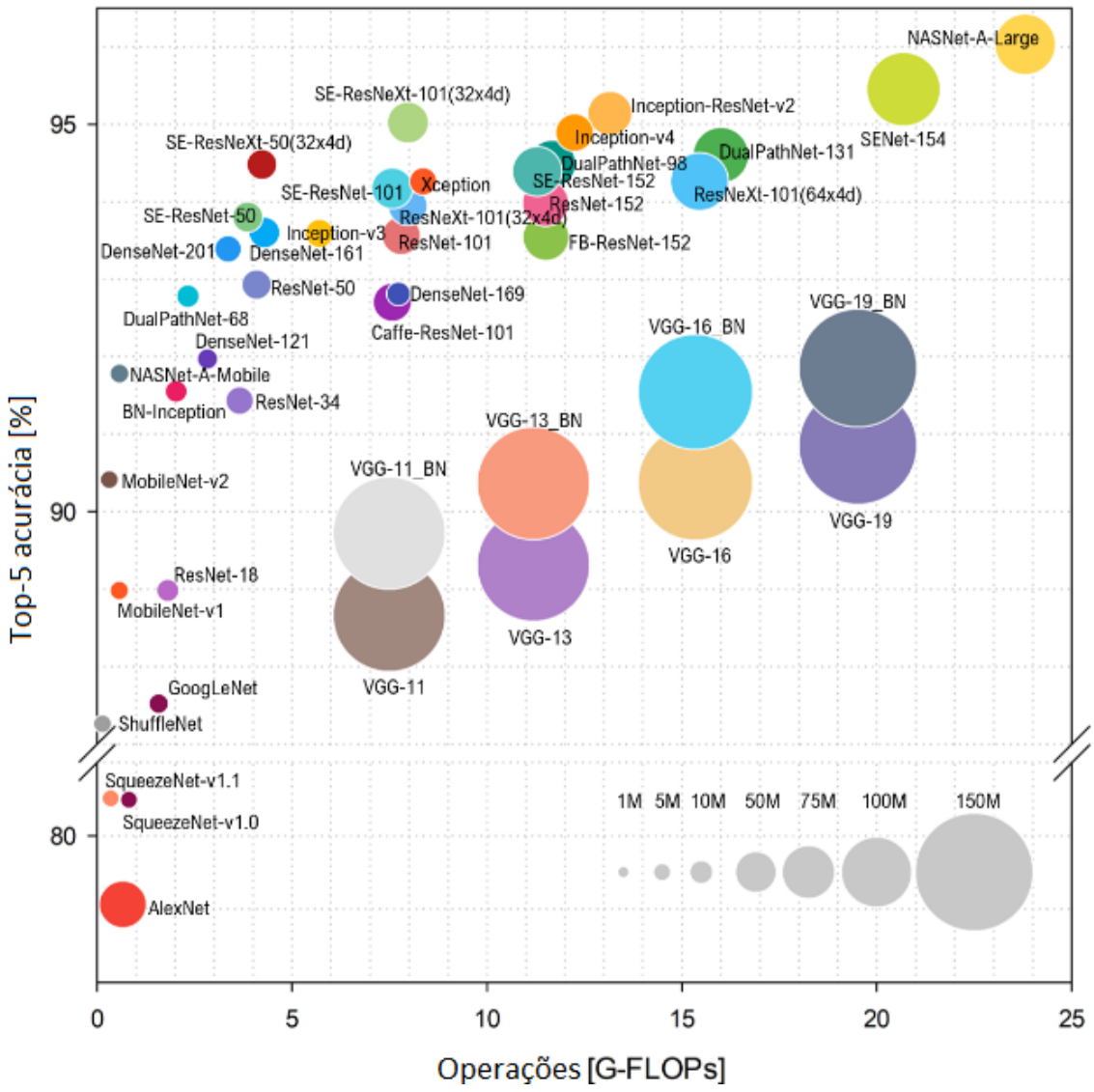


Figura 4.10 – Acurácia de arquiteturas de *deep learning* por número de operações.

Fonte: (BIANCO *et al.*, 2018)

Devido à diferença de complexidade entre as profundidades, é esperado que o treinamento demore mais a convergir, tanto pela quantidade de operações computacionais (mais camadas, mais operações), quanto pela dificuldade de otimização do *backpropagation* ao atualizar os pesos. Sendo a rede de detecção de *keypoints* (Keypoints Net) a primeira a ser treinada, é esperado que esta seja a que demande mais tempo de treinamento, uma vez que a próxima rede de detecção de pessoas (PersonDet Net), inicia o processo de treinamento com o peso do *backbone* compartilhado e congelado. Porém, é importante notar que ao usar os pesos de outra rede e não ser possível realizar a atualização, a rede demorou mais épocas (chama-se época o tempo de iteração total sobre todo *dataset*) para convergir.

A Tabela 4.4 detalha os tempos de treinamento utilizando uma GPU Nvidia V100 com as configurações descritas na seção 4.1.2 e o *dataset* de *MSCOCO Dataset 2017 training set*. Realizou-se um teste com a rede de detecção de pessoas, treinando o modelo completo, inclusive o *backbone*. Apesar de não ser útil para o trabalho aqui desenvolvido, mostra-se a diferença entre o tempo de treinamento de cada época e a quantidade de épocas necessárias em cada um. No fim, o tempo de convergência é próximo. Os números após o nome dos modelos indicam a profundidade da ResNet

Tabela 4.4 – Tempo de treinamento das redes para detecção de *keypoints* e detecção de pessoas.

	Tempo total)	**Tempo/época**	**Qtd. de épocas**
KeypointNet-50	41 horas	32 min	78
KeypointNet-101	68 horas	47 min	87
KeypointNet-152	91 horas	56 min	98
PersonDet Net-50 (backbone congelado)	15 horas	16 min	61
PersonDet Net-101 (backbone congelado)	25 horas	17 min	80
PersonDet Net-152 (backbone congelado)	26 horas	19 min	80
PersonDet Net-50 (backbone treinável)	17 horas	32 min	30
PersonDet Net-101 (backbone treinável)	24 horas	42 min	35
PersonDet Net-152 (backbone treinável)	33 horas	54 min	37

Os gŕaficos da função de custo (*loss*) por época de treinamento são apresentados a seguir para cada uma das subredes com os três tipos de *backbone*: ResNet50, Resnet101, Resnet152. Esse cálculo é realizado de acordo com as expressões 4.12 e 4.13, para a rede de detecção de *keypoints* e a rede de detecção de pessoas, respectivamente. O modelo com menor erro do primeiro treinamento (KeypointNet) é salvo e utilizado como inicialização dos pesos do *backbone* do próximo treinamento (PersonDet).

Os resultados do erro ou perda (*loss*) (4.12) no treinamento e validação são obtidos ao final de cada época quando é executado uma rotina que avalia o modelo com os pesos atuais no dataset de validação. Isso permite identificar o ponto em que o modelo apresenta melhor generalização e onde começa o problema de overfitting. Na Figura 4.11, o melhor modelo foi obtido após 78 épocas de treinamento, com a loss de validação aumentando após esse período enquanto a loss de treinamento continua decrescendo. As Figuras 4.12 e 4.13 seguem o mesmo princípio.

$$f = \mathbf{W} \cdot ||H_p - S_j^*||_2^2 + \sum_{i=2}^{5} \mathbf{W} \cdot ||Z_i - S_{j,k}^*||_2^2 \tag{4.12}$$

$$FL(p_t) = -\alpha_t (1 - p_t)^{\gamma} log(p_t) \tag{4.13}$$

Figura 4.11 – Gráfico de treinamento da subrede KeypointNet com backbone Resnet50.

Fonte: Igor Soares, 2019

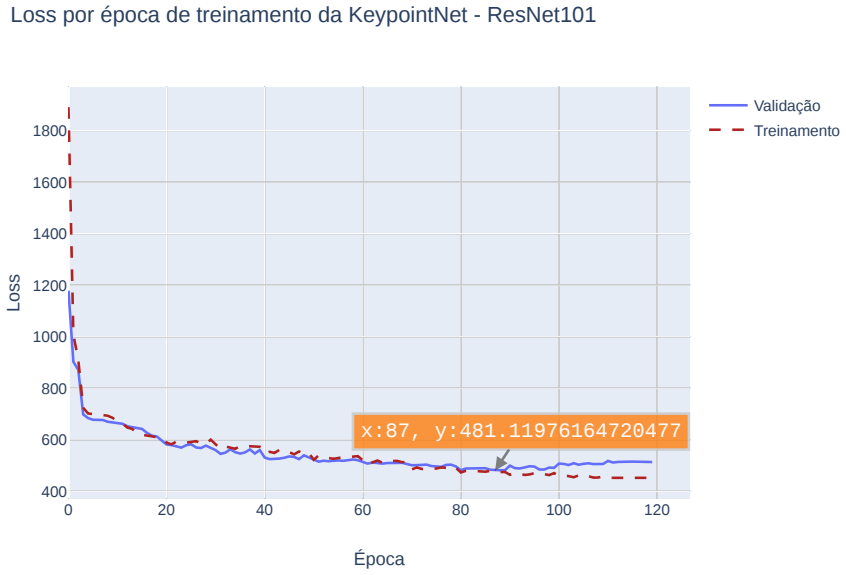


Figura 4.12 – Gráfico de treinamento da subrede KeypointNet com backbone Resnet101.

Fonte: Igor Soares, 2019

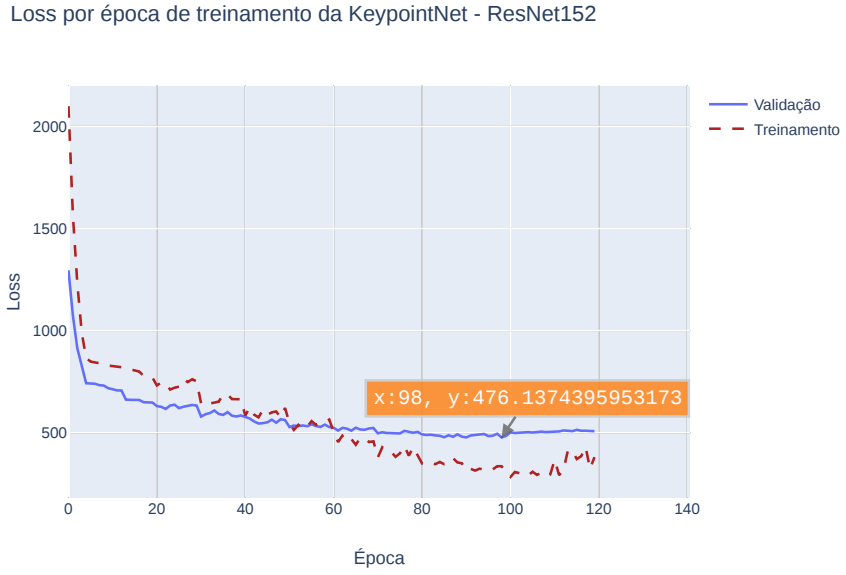


Figura 4.13 – Gráfico de treinamento da subrede KeypointNet com *backbone* Resnet152.

Fonte: Igor Soares, 2019

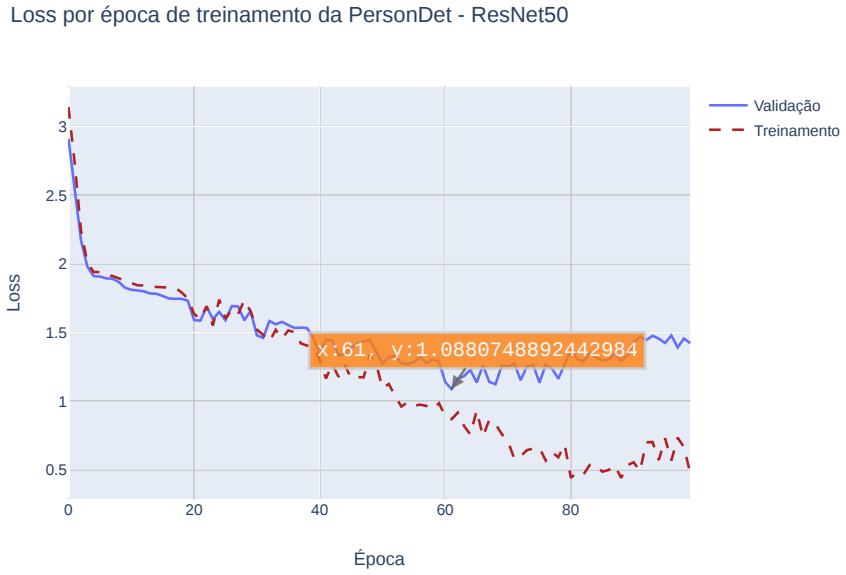


Figura 4.14 – Gráfico de treinamento da subrede PersonDet com inicialização da KeypointNet-ResNet50.

Fonte: Igor Soares, 2019

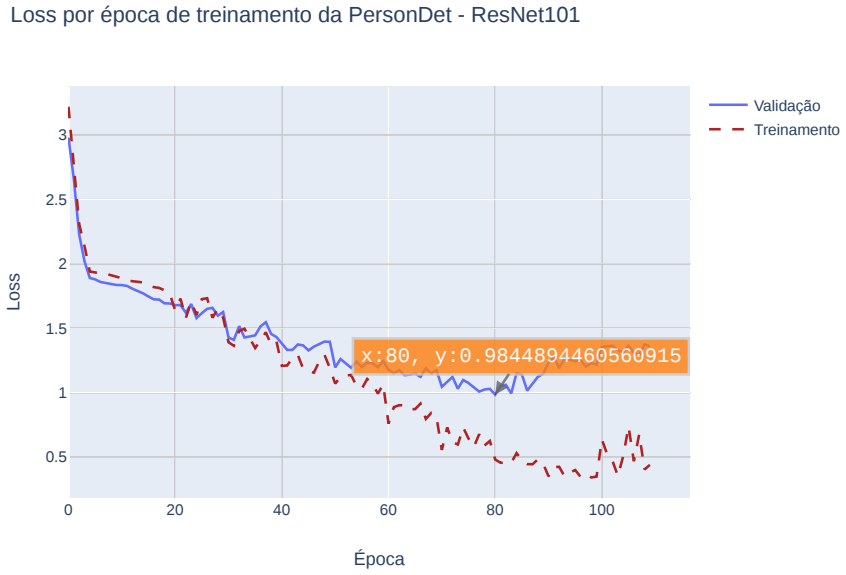


Figura 4.15 – Gráfico de treinamento da subrede PersonDet com inicialização da KeypointNet-ResNet101.

Fonte: Igor Soares, 2019

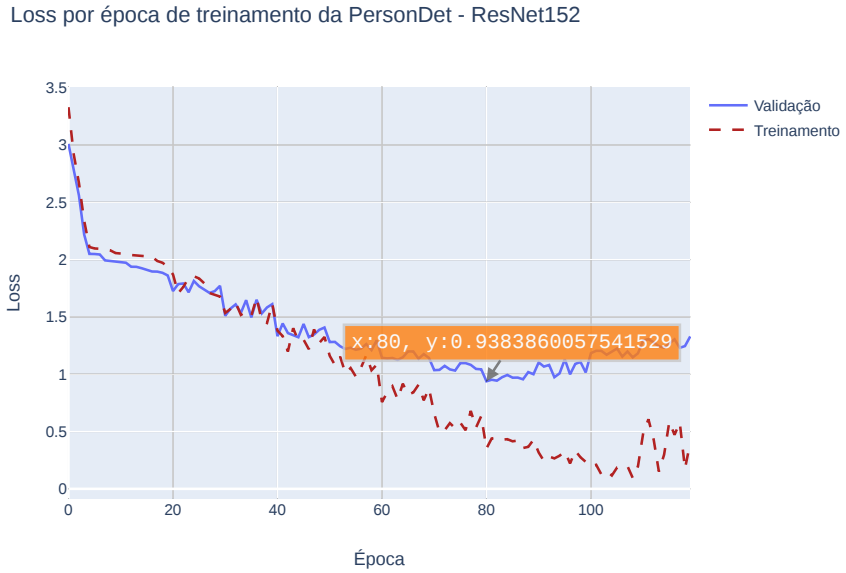


Figura 4.16 – Gráfico de treinamento da subrede PersonDet com inicialização da KeypointNet-ResNet152.

Fonte: Igor Soares, 2019

Ao final do treinamento, os modelos foram avaliados em suas respectivas tarefas com *MSCOCO Dataset validation set*. Para a tarefa de estimação de pose, utilizou-se a saída dos dois modelos para associar os *keypoints* detectados a cada pessoa na cena. Nota-se que os resultados de estimação de pose completa são alcançados ao se colocar a camada PRN (KOCABAS; KARAGOZ; AKBAS, 2018) e estes são apresentados mais pra frente. A rede de detecção de pessoas não sofre nenhuma alteração posterior e por isso seus resultados já são comparados com outros trabalhos na Tabela 4.5. Ambas as tarefas são avaliadas com o *mean Average Precision*(mAP) dos resultados, porém na de detecção de pessoas, é calculado o mAP sobre a interseção da união (IoU), e na tarefa de detecção de *keypoints* sobre a similaridade de pontos no objeto (OKS).

Os resultados da parte de detecção de pessoas deste trabalho são avaliados na tarefa de detecção de objetos do *dataset* para uma única classe (*person*) e são apresentados na Tabela 4.5, onde calcula-se o mAP sobre o IoU da *bounding box* predita e da desejada. Os resultados são comparados com outros trabalhos e também avaliou-se a diferença entre o *backbone* treinável (BT) e com os pesos congelados

Mesmo compartilhando pesos de outro treinamento no *backbone*, a rede para detecção de pessoas alcançou o estado da arte com resultados próximos a de outros trabalhos, inclusive a própria RetinaNet com treinamento completo. Isso demonstra a

Tabela 4.5 – Resultados de detecção de pessoas no *MSCOCO 2017 validation set*

	AP	**AP$_{50}$**	**AP$_{75}$**	**AP$_{S}$**	**AP$_{M}$**	**AP$_{L}$**
MultiPoseNet(101)(KOCABAS; KARAGOZ; AKBAS, 2018)	**52.5**	**81.5**	**55.3**	**35.2**	**59.0**	71.0
RetinaNet(LIN; AI; DOLL, 2018)	50.2	77.7	53.5	31.6	59.0	**71.5**
PersonDet SubNet (ResNet50)	49.9	77.1	53.0	31.5	58.1	70.4
PersonDet SubNet (ResNet101)	50.4	79.3	54.1	33.7	58.7	70.8
PersonDet SubNet (ResNet152)	51.1	80.4	54.8	34.9	58.9	71.0
PersonDet SubNet (ResNet50)(BT)	50.7	78.3	53.8	32.3	58.4	70.9
PersonDet SubNet (ResNet101) (BT)	51.5	80.0	54.3	34.3	58.7	71.2
PersonDet SubNet (ResNet152) (BT)	52.2	80.5	55.0	35.2	59.0	71.5

capacidade de utilizar *transfer learning* com congelamento de camadas em tarefas de semântica parecida sem grandes perdas no resultado final.

Os resultados da estimação de pose com a detecção de *keypoints* e detecção de pessoas foram obtidos utilizando-se a região delimitada pela *bounding box* predita de cada pessoa pela rede PersonDet para associar os *keypoints* detectados pela rede KeypointNet a cada pessoa na imagem. Os resultados presentes na Tabela 4.6 ainda não possuem nenhuma tarefa de correção. O mAP é calculado sobre o OKS de cada coordenada de um ponto predita com a coordenada desejada.

Tabela 4.6 – Resultados de estimação de pose no *MSCOCO 2017 validation set* com a combinação de detecção de *keypoints* mais detecção de pessoas.

	AP	**AP$_{50}$**	**AP$_{75}$**	**AP$_{M}$**	**AP$_{L}$**
Estimação de Pose (ResNet50)	47.4	72.2	51.5	38.8	55.9
Estimação de Pose (ResNet101)	48.2	74.8	52.6	39.3	56.8
Estimação de Pose (ResNet152)	48.9	77.1	53.0	39.8	57.3

Necessidade de Supervisão Intermediária

Ao longo do desenvolvimento do trabalho percebeu-se a necessidade de aplicar uma supervisão intermediária no treinamento da rede para detecção de *keypoints*. Tal supervisão consiste em adicionar no cálculo da função de perda a comparação entre a saída desejada e camadas intermediárias, e não apenas a tradicional comparação com a saída da rede. Acredita-se que esta técnica, utilizada principalmente em tarefas de redes totalmente convolucionais, melhora o aprendizado das camadas intermediárias, dando um *feedback* diretamente as mesmas, além de ajudar no problema de desaparecimento do gradiente (CAO *et al.*, 2017).

Inicialmente, a função de custo para o treinamento da rede de detecção de *keypoints* adotada neste trabalho era a descrita pela Expressão 4.14, sendo calculada com a saída desejada e cada *heatmap* predito. Porém, os primeiros resultados foram bem abaixo do esperado.

$$f = \mathbf{W} \cdot ||H_p - S_j^*||_2^2 \tag{4.14}$$

Pesquisando na literatura por possíveis melhorias, percebe-se, em trabalhos importantes no campo de estimação de pose como Cao *et al.* (2017) e Kocabas, Karagoz e Akbas (2018), a necessidade de se calcular o erro da rede de acordo a saída de camadas intermediárias e a saída desejada. Dessa forma, obteve-se a função custo ideal para treinamento (discutida em 4.1.2), ao relacionar a saída desejada com os mapas pirâmidais do *backbone*.

$$f = \mathbf{W} \cdot ||H_p - S_j^*||_2^2 + \sum_{i=2}^{5} \mathbf{W} \cdot ||Z_i - S_{j,k}^*||_2^2 \tag{4.15}$$

Dada a diferença entre as funções de custo, não é possível medir o desempenho dos modelos durante o treinamento, uma vez que o erro (*loss*) em cada época apresenta valores muito distintos. Assim, a Tabela 4.7 mostra a diferença entre os dois modelos, com e sem supervisão, avaliando os mesmos no *dataset* de validação do *MSCOCO Dataset 2017*. Os modelos treinados sem supervisão intermediária (marcados na Tabela com um S) apresentam resultados bem inferiores ao modelos com supervisão na função de custo.

Tabela 4.7 – Comparação de resultados de modelos com e sem supervisão intermediária.

	AP	**AP$_{50}$**	**AP$_{75}$**	**AP$_{M}$**	**AP$_{L}$**
Estimação de Pose (ResNet50)(S)	34.6	53.8	40.9	25.2	45.7
Estimação de Pose (ResNet101)(S)	35.1	55.3	42.9	26.9	46.2
Estimação de Pose (ResNet152)(S)	35.3	55.6	43.4	27.1	46.3
Estimação de Pose (ResNet50)	47.4	72.2	51.5	38.8	55.9
Estimação de Pose (ResNet101)	48.2	74.8	52.6	39.3	56.8
Estimação de Pose (ResNet152)	48.9	77.1	53.0	39.8	57.3

Influência da Camada de BatchNormalization do Keras

Outro fator importante para obtenção do resultado esperado foi a modificação da camada de *BatchNormalization* do Keras. O problema (melhor discutido no apêndice

A) aparece ao utilizar os pesos de um treinamento em outra tarefa e ainda congelá-los para que não sejam atualizados no novo treinamento, situação presente e necessária no treinamento da rede de detecção de pessoas. Ao gerar novas informações estatísticas do *dataset* no segundo treinamento e propagá-las na rede, sem a possibilidade de atualizar os pesos de grande parte das camadas, o treinamento demora mais a convergir e obtém resultados finais inferiores.

A solução é manter a média e variância obtida no primeiro treinamento, sem recálculos no *batch* de entrada do segundo treinamento, mantendo os pesos congelados condizentes com os últimos valores vistos por eles. O gráfico 4.17 e a Tabela 4.8 mostram a diferença no treinamento e avaliação da subrede de detecção de pessoas com a camada de *BatchNormalization* oficial do Keras e a modificada. O modelo treinado com a camada de BatchNormalization oficial apresenta muito mais dificuldade em convergir e resultado final bem inferior.

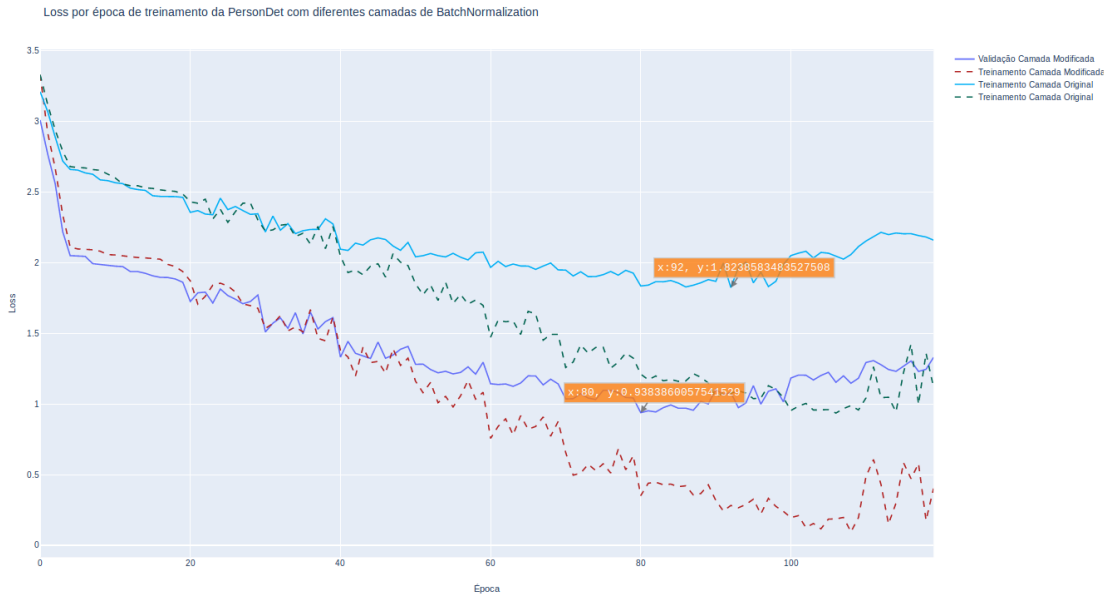


Figura 4.17 – Gráfico de comparação do treinamento e validação da PersonDet com a camada de BatchNormalization oficial e modificada.

Fonte: Igor Soares, 2019

Tabela 4.8 – Validação do modelo de detecção de pessoas com a camada BatchNormalization oficial do Keras (O) e a modificada (M).

	AP	**AP$_{50}$**	**AP$_{75}$**	**AP$_{S}$**	**AP$_{M}$**	**AP$_{L}$**
PersonDet SubNet (ResNet50) (M)	49.9	77.1	53.0	31.5	58.1	70.4
PersonDet SubNet (ResNet50) (O)	35.7	51.9	38.6	22.7	40.1	49.4

4.4.2 Rede Residual de Poses

O modelo PRN (*Pose Residual Network*) é um modelo simples e eficiente para melhorar a acurácia de arquiteturas para estimação de pose (KOCABAS; KARAGOZ; AKBAS, 2018). Tal rede consegue agrupar melhor os *keypoints* a cada instância de pessoa detectada, baseando-se em poses já conhecidas.

O treinamento é feito com os próprios dados de *ground truth* do *MSCOCO Dataset train set*, onde a entrada é todos os *keypoints* detectados em cena (situação convencional de um modelo *bottom-up* limitados pela *bounding box* de cada pessoa. Com isso, tem-se alguns pontos em regiões de outras pessoas (sobreposição). A saída é a correção desses pontos que não devem existir em tal região (ver Figura 4.4).

Durante o treinamento, o modelo é avaliado ao final de cada época no *MSCOCO Dataset Validation Set*, calculando-se a perda e também o mAP. O gráfico 4.18 mostra a diminuição do erro de validação causada pela PRN, onde nota-se que o modelo começa o treinamento já com uma perda baixa devido a entrada ser gerada com a própria ground truth do *dataset*. Mesmo assim, a rede consegue reduzir em 0.12 pontos a perda na validação. Já o gráfico 4.20 indica o ganho de mAP do modelo de aproximadamente 7 pontos com a PRN em comparação a entrada com apenas a *ground truth* do *dataset*

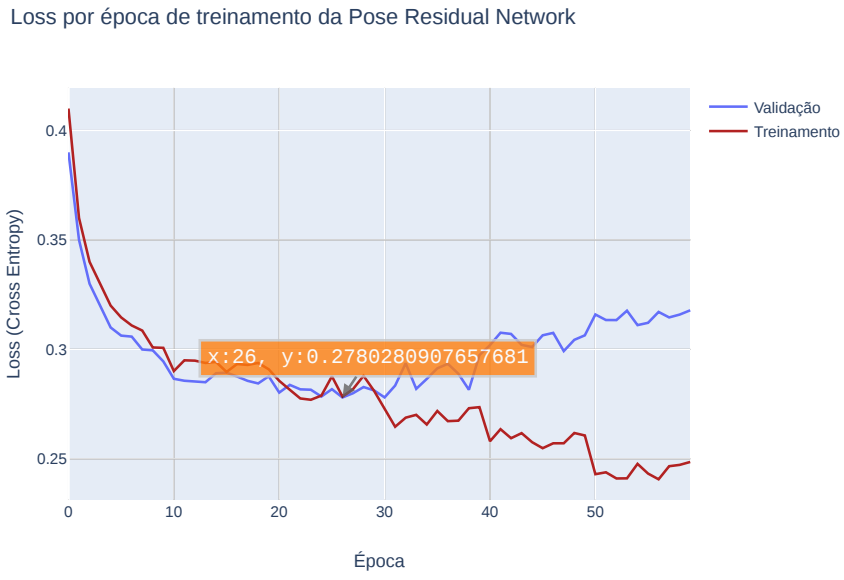


Figura 4.18 – Gráfico da perda (*loss*) por época de treinamento da PRN.

Fonte: Igor Soares, 2019

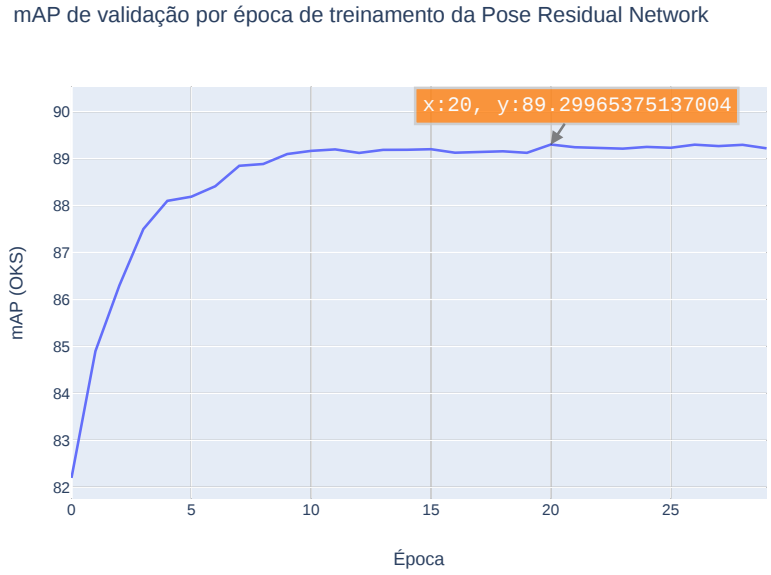


Figura 4.19 – Gráfico do mAP de validação da PRN.

Fonte: Igor Soares, 2019

Tabela 4.9 – Resultado da validação da Rede Residual de Pose.

	AP	**AP$_{50}$**	**AP$_{75}$**	**AP$_{M}$**	**AP$_{L}$**
GT Keypoints + GT BB	82.5	90.4	84.9	81.1	84.6
Rede Residual de Pose	89.3	97.8	92.0	88.3	91.5

A rede residual de poses foi acoplada a saída dos modelos de detecção de pessoas e detecção de *keypoints* deste trabalho para melhores resultados na tarefa de estimação de pose. Como esperado, houve um ganho de 2 pontos no mAP. A Tabela 4.11 mostra os resultados deste trabalho na tarefa de estimação de pose com e sem PRN. Também é feito a comparação com outros trabalhos da mesma metodologia *bottom-up*. Todos os testes foram realizados no *MSCOCO Dataset Validation Set*.

Semelhante aos trabalhos comparados, durante os testes, aplicou-se uma técnica chamada *Test Time Augmentation*, que consiste em aplicar transformações na imagem de teste e mesclar os resultados. Isso permite que a rede "veja a mesma entrada de diferentes maneiras", melhorando a sua capacidade de generalização. Porém, não realizou-se *emsembling*, outra técnica utilizada na literatura para obter melhores resultados, consistindo em juntar o resultados de diferentes arquiteturas, ou a mesma arquitetura com parâmetros diferentes. Como o foco do trabalho é o desenvolvimento de uma única arquitetura, optou-se por não realizar *emsembling*. Por isso, resultados pouco piores (geralmente a influência de emsembling é na casa dos decimais) são esperados.

Para *test time augmentation*, realizou-se apenas transformações de escala na imagem, apresentando a mesma imagem com sua altura redimensionada em $240, 480, 720$ e 960 pixels (esses valores correspondem respectivamente a 50%, 100%, 150% e 200% do tamanho das entradas utilizado no treinamento). A largura é redimensionada de maneira a manter proporção altura-largura original da imagem. A Tabela 4.10 mostra a influência do *Test Time Augmentation* no mAP ao, onde os resultados da Tabela 4.6 são consideravelmente melhorados ao realizar a mesma avaliação, mas com 4 escalas diferentes de imagens.

Tabela 4.10 – Ganho de mAP com *test time augmentation*.

	AP	**AP$_{50}$**	**AP$_{75}$**	**AP$_{M}$**	**AP$_{L}$**
Estimação de Pose (ResNet50)	47.4	72.2	51.5	38.8	55.9
Estimação de Pose (ResNet101)	48.2	74.8	52.6	39.3	56.8
Estimação de Pose (ResNet152)	48.9	77.1	53.0	39.8	57.3
Estimação de Pose + TTA (ResNet50)	56.8	80.3	64.9	52.5	64.3
Estimação de Pose + TTA (ResNet101)	57.9	81.2	67.7	54.1	66.4
Estimação de Pose + TTA (ResNet152)	59.2	82.0	69.9	54.7	67.1

Tabela 4.11 – Resultados de estimação de pose no MSCOCO validation set. Comparações com outros trabalhos.

	AP	**AP$_{50}$**	**AP$_{75}$**	**AP$_{M}$**	**AP$_{L}$**
MultiPoseNet(101)(KOCABAS; KARAGOZ; AKBAS, 2018)	63.9	87.1	**73.2**	58.1	72.2
CMU-Pose(CAO *et al.*, 2017)	58.4	81.5	62.6	54.4	65.1
SSD(LIU *et al.*, 2015) + CPM(WEI, 2016)	52.7	71.1	57.2	47.0	64.2
Li et. al(LI *et al.*, 2018)	**64.7**	**88.5**	69.6	**59.4**	**73.7**
Este trabalho (ResNet50)	56.8	80.3	64.9	52.5	64.3
Este trabalho (ResNet101)	57.9	81.2	67.7	54.1	66.4
Este trabalho(ResNet152)	59.2	82.0	69.9	54.7	67.1
Este trabalho + PRN (ResNet50)	59.7	82.9	67.4	54.8	66.3
Este trabalho + PRN (ResNet101)	60.8	83.3	70.5	55.9	68.8
Este trabalho + PRN (ResNet152)	61.3	84.4	72.1	56.3	70.6

4.4.3 Rede para Reconhecimento de Atividades pela Pose

A rede para reconhecimento de atividades pela pose, proposta neste trabalho, é uma maneira simples e eficiente de classificar atividade de uma pessoa através dos padrões entre as coordenadas das partes de seu corpo. Tal rede foi desenvolvida com intuito de ser acoplada em uma única arquitetura para as duas tarefas, estimação de pose e classificação de atividade, além de ser rápida ao ponto de não influenciar no tempo de inferência. Para

desenvolver um modelo com mais acurácia, várias configurações de hiperparâmetros foram testadas. A Tabela 4.12 mostra essas configurações e os resultados obtidos.

Tabela 4.12 – Resultados de diferentes modelos de classificação de atividade.
Os números após "Camada"indicam a quantidade de neurônios em cada camada e o "+D"indica a presença de *dropout* e seu fator.

NN Modelos	**Precisão**	**Recall**	**F1 Score**	**Acurácia**
1 Camada 128	0.6646	0.6247	0.6386	0.6171
1 Camada 528 + D0.3	0.6774	0.6373	0.6497	0.6285
2 Camadas 528-128 + D0.3	0.7067	0.6993	0.7015	0.6815
2 Camadas 1024-128 + D0.3	0.7171	0.7113	0.7110	0.6914
2 Camadas 1024 -512 + D0.3	0.6864	0.6790	0.6806	0.6571
2 Camadas 2048-128 + D0.3	0.7268	0.7089	0.7159	0.6857
2 Camadas 3072-128 + D0.3	**0.7371**	**0.7324**	**0.7344**	**0.7028**
2 Camadas 4096-128 + D0.3	0.6865	0.6726	0.6588	0.6514
2 Camadas 3072-128	0.6685	0.6751	0.6655	0.6457
2 Camadas 3072-128 + D0.1	0.6614	0.6672	0.6586	0.6342
2 Camadas 3072-128 + D0.2	0.7198	0.7021	0.7042	0.6857
2 Camadas 3072-128 + D0.4	0.7310	0.7046	0.7111	0.6914
2 Camadas 3072-128 + D0.5	0.7196	0.6852	0.6932	0.6742

Para a medir qual modelo se comportou melhor, utilizou-se quatro métricas: precisão, *recall*, *f1 score* e acurácia. Além disso, a validação foi realizada de maneira cruzada em cinco diferentes *subsets* de validação contendo 20% dos dados do *dataset* (desenvolvido neste trabalho). Essa técnica, conhecida como *K Fold Cross Validation*, evita que o treinamento e a validação sejam enviesados dentre as classes presentes, tornando os resultados mais próximos da realidade. Por se tratar de uma rede neural de múltiplas camadas, não existe a necessidade de aprendizagem profunda, e por isso o treinamento de cada modelo demora aproximadamente 10 minutos para convergir, tornando mais prático o teste com diferentes configurações.

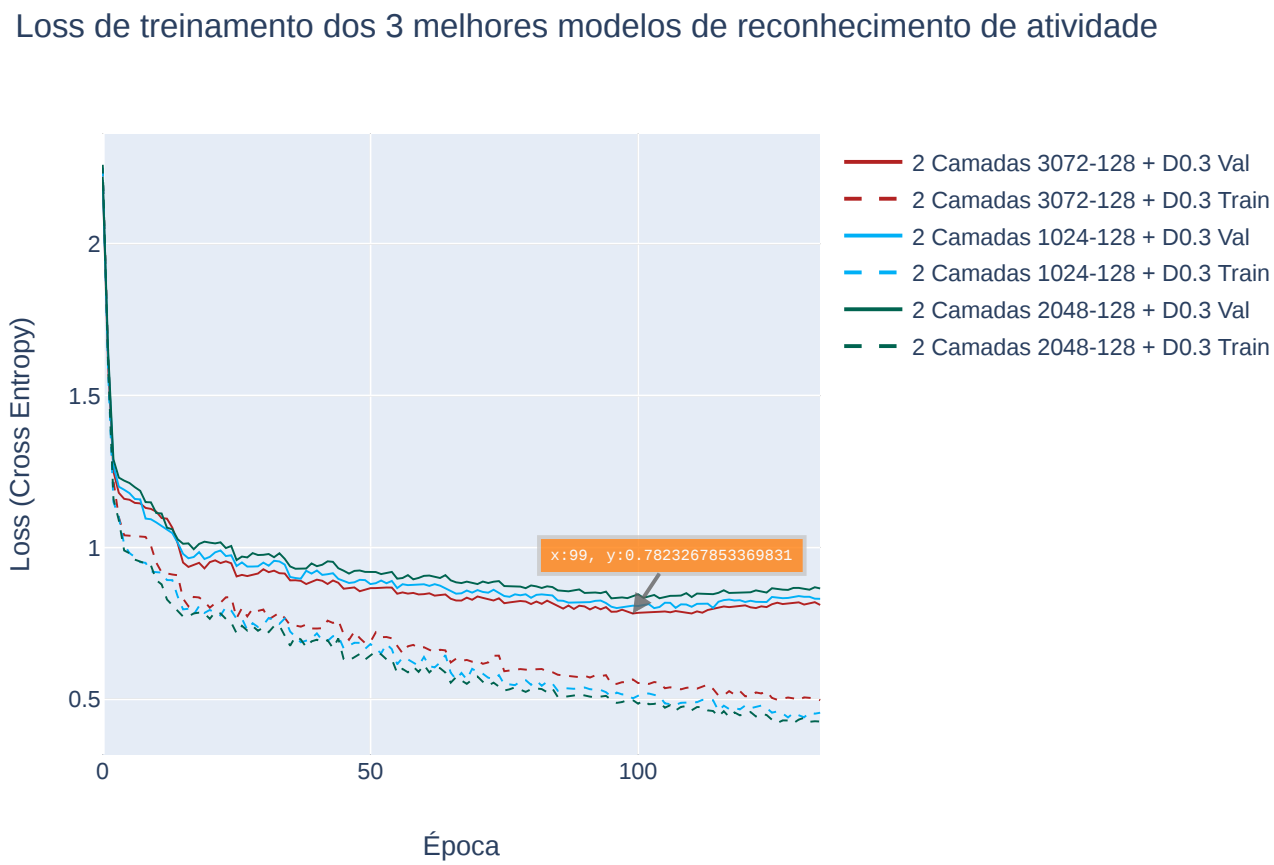


Figura 4.20 – Gráfico da perda (*loss*) de treinamento e validação dos 3 melhores modelos da tabela 4.12.

Fonte: Igor Soares,2019

Para o modelo com as melhores pontuações nas quatro métricas, calculou-se a sua matriz de confusão, uma espécie de tabela utilizada para medir a performance de um algoritmo de classificação. Esta matriz, quando normalizada, indica a porcentagem de classes classificadas corretamente e também qual a porcentagem de confusão com outras classes. A Figura 4.21 apresenta a matriz de confusão para o modelo com melhor resultado em que a classe "sentado"é a atividade mais fácil de ser reconhecida. Os resultados são a média da validação nos 5 *subsets* diferentes.

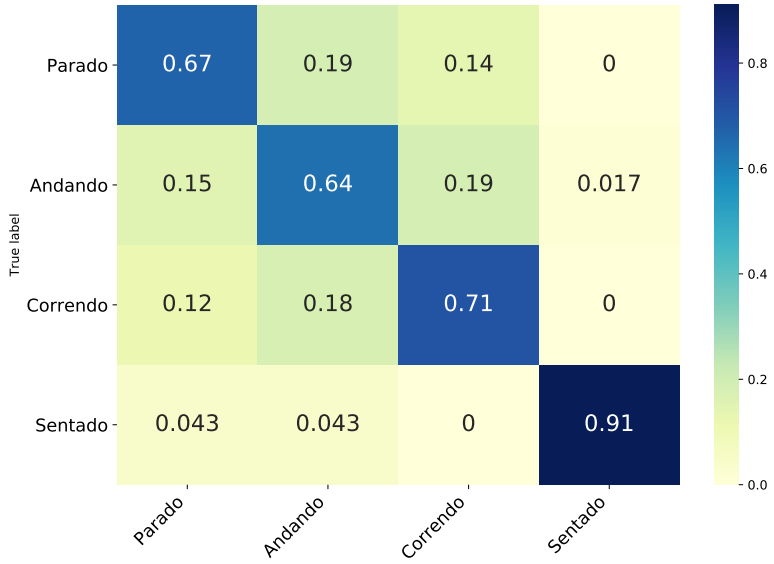


Figura 4.21 – Matriz de confusão do melhor modelo apresentado na Tabela 4.12.

Fonte: Igor Soares,2019

É importante observar que a classe da atividade "sentado"obteve 91% de classificações corretas, enquanto as outras classes, "parado", "andando"e "correndo", possuem respectivamente 67%, 64%, e 71%, indicando uma facilidade maior em reconhecer a atividade "sentado". Isso pode ser explicado pela proximidade da pose nas outras atividades, principalmente em imagens, que são capazes de captar um único momento.

4.4.4 Experimentos de Tempo de Execução

Um dos grandes contras de algoritmos de aprendizagem profunda é o tempo de execução. Alguns algoritmos, principalmente os multitarefas como o caso deste trabalho, não são viáveis por não conseguirem execução em tempo real. A escolha inicial da metodologia *bottom-up* visava a construção de uma arquitetura com um tempo de inferência aceitável para implantações futuras em outros projetos.

Após a finalização do desenvolvimento da metodologia proposta, realizou-se análises de tempo de execução. Para isso, utilizou-se duas placas de vídeo: a NVIDIA v100 utilizada durante todos os treinamentos, e uma NVIDDIA GTX 1080TI, placa mais convencional e utilizada também em outros trabalhos, possibilitando a comparação. Também mediu-se o

tempo de execução em uma CPU. As especificações do hardware utilizado nos testes são mostradas nas Tabelas 4.13[12] e 4.14[13].

Tabela 4.13 – Comparação entre a Nvidia Tesla V100 e a NVIDIA GTX 1080 TI

	Tesla V100	**GTX 1080 TI**
CUDA Cores	5120	3584
Tensor Cores	640	N/A
Double Precision	7 teraFLOPS	0.355 teraFLOPS
Single Precision	14 teraFLOPS	0.7 teraFLOPS
Deep Learning	112 teraFLOPS	11.3 teraFLOPS
Memory Bandwith	900GB/s	484GB/s
Memory Capacity	16GB	11GB

Tabela 4.14 – Especificaçãos da CPU utilizada nos testes

	Intel Core i5 7200U
Número Cores	2
Número Threads	4
Tamanho Cache	3 mb
Frequência	2.5 GHz
Frequência Turbo	3.1 GHz

Para o teste de *runtime*, utilizou-se as 2693 imagens presentes no *MSCOCO Dataset validation set*. Os tempos apresentados são a média do tempo de execução de cada uma dessas imagens. Para medir a performance, as imagens foram redimensionadas no tamanho de 384×576, o mesmo tamanho utilizado nos testes de outros trabalhos de metodologia *bottom-up*. Como o tamanho da entrada é um fator crucial no tempo de execução, esse tamanho é utilizado por motivo de comparação, mas a arquitetura final permite a utilização de tamanhos variados.

As Tabelas 4.15 e 4.16 fornecem informações da quantidade parâmetros e tempo de execução de cada parte da rede. Os tempos de execução medidos são apenas das operações de instruções da rede, desconsiderando outros tipos de processamento como carregamento de arquivos na memória e transformações na entrada. Os tempos mostrados das subredes KeypointNet e PersonDet são da inferência em suas camadas após o *backbone*.

Como esperado, a parte que consome mais processamento e consequentemente possui maior tempo de inferência é o *backbone*, responsável pela extração de características

[12] Informações disponibilizadas pelo fabricante no manual de cada produto. Disponível em: https://www.nvidia.com/

[13] Informações disponibilizadas pelo fabricante em: https://ark.intel.com/content/www/us/en/ark/products/95443/int core-i5-7200u-processor-3m-cache-up-to-3-10-ghz.html

Tabela 4.15 – Quantidade de parâmetros de cada subrede do modelo final.

	Qtd. Parâmetros
Backbone (Resnet50)	23 M
Backbone (Resnet101)	42M
Backbone (Resnet152)	60M
Keypoint Net	3M
Person Det	12 M
PRN	8M
Classificador de atividade	0.5 M

Tabela 4.16 – Tempo de execução de cada subrede do modelo final.

	GTX 1080 TI	**Tesla V100**	**i5 7200U**
Backbone (Resnet50)	29.12 ms	24.46 ms	2235.81 ms
Backbone (Resnet101)	39.74 ms	34.35 ms	4358.40 ms
Backbone (Resnet152)	54.72 ms	47.93 ms	6528.47 ms
Keypoint Net	6.84 ms	5.97 ms	345.87 ms
Person Det	15.21 ms	12.87 ms	1024.65 ms
PRN	3.24 ms	2.28 ms	258.75 ms
Classificador de atividade	1.53 ms	1.02 ms	98.15 ms

necessárias para identificação dos padrões. Assim, compartilhando o *backbone* entre as próximas duas subredes, é necessário apenas o tempo de uma inferência nessa parte, reduzindo bastante o tempo total do modelo completo. As redes PRN e de classificação de atividade possuem seu tempo de inferência proporcional ao número de pessoas na imagem, já que elas são executadas para cada instância, e os valores apresentados na Tabela 4.15 são uma média de diferentes imagens com número variado de pessoas. Porém, essas duas redes são rápidas quando comparadas as demais, principalmente a parte de classificação deste trabalho, com influência mínima no tempo de execução final.

Tabela 4.17 – Resultados dos experimentos de tempo de execução.
Comparação com outros trabalhos em números de *frames* por segundo (fps) e diferença de tempo de execução em 2 GPUs e 1 CPU.

	GTX 1080 TI	**Tesla V100**	**i5 7200U**
MultiPoseNet(50)(KOCABAS; KARAGOZ; AKBAS, 2018)	23 fps	–	–
CMU-Pose(CAO *et al.*, 2017)	10 fps	–	–
Li et. al(LI *et al.*, 2018)	12 fps	–	–
Modelo final (ResNet50)	20.3 fps	24.8 fps	0.27 fps
Modelo final (ResNet101)	16.7 fps	20.1 fps	0.17 fps
Modelo final (ResNet152)	13.9 fps	15.6 fps	0.13 fps

Tabela 4.18 – Comparação entre modelo único com backbone compartilhado e modelos distintos.
É possível obter os mesmos resultados executando cada modelo separadamente entretanto, sem o compartilhamento de *backbone*, a parte mais pesada da rede é executada duas vezes, além de ser sequencial.

	GTX 1080 TI	**Tesla V100**	**i5 7200U**
(ResNet50)KeypointNet + (ResNet50)PersonDet + PRN + Classificador	11.1 fps	13.4 fps	0.11 fps
(ResNet101)KeypointNet + (ResNet50)PersonDet + PRN + Classificador	7.8 fps	9.9 fps	0.08 fps
(ResNet152)KeypointNet + (ResNet50)PersonDet + PRN + Classificador	6.5 fps	8.1 fps	0.05 fps
Modelo final (ResNet50)	20.3 fps	24.8 fps	0.27 fps
Modelo final (ResNet101)	16.7 fps	20.1 fps	0.17 fps
Modelo final (ResNet152)	13.9 fps	15.6 fps	0.13 fps

5 Conclusão

O presente trabalho apresentou o desenvolvimento de uma arquitetura de rede neural convolucional multitarefas para reconhecimento de atividade de uma pessoa através da sua pose. O modelo final é capaz de detectar *keypoints*, detectar pessoas, estimar a pose destas e classificar a atividade que estão realizando a partir da pose. Toda metodologia foi implementada em Python, utilizando bibliotecas de aprendizagem profunda escritas em C++, o que resultou em praticidade de codificação e alta performance nos treinamentos do algoritmo.

Todos treinamentos foram realizados em uma GPU NVIDIA Tesla V100, placa de vídeo essencial para realização de vários testes devido ao tempo médio de treinamento de 93 horas. Quanto aos dados, utilizou-se o *dataset MSCOCO Dataset 2017*, além de um *dataset* próprio desenvolvido neste trabalho, necessário para treinamento de classificação de atividade a partir da pose. Este último, contou com quatro classes de atividade: andando, correndo, parado e sentado. Os dois *datasets* foram suficientes para o treinamento e validação de todas tarefas do modelo final.

Para medir os resultados, utilizou-se as métricas oficiais do *MS COCO Dataset* nas tarefas de detecção de pessoas e estimação de pose, sendo elas precisão da média da interseção sobre a união e precisão média da similaridade de pontos do objeto, respectivamente. As diferentes profundidades da ResNet mostraram a versatilidade em adaptar o modelo de acordo com o objetivo, sendo preferível utilizar profundidades menores quando a prioridade é tempo de treinamento e execução, mas sem perdas exacerbadas nos resultados.

Com os experimentos, ficou clara a necessidade de se adicionar supervisão intermediária na função de custo do treinamento da rede de detecção de *keypoints*, elevando consideravelmente os resultados finais. A arquitetura aqui apresentada, é capaz de estimar pose com 61.3 mAP, quando utilizado o *backbone* com ResNet152, o de maior profundidade. Esse resultado está no estado da arte quando comparado a outras metodologias *bottom-up* e próximo ao melhor resultado atual.

Quanto a rede de detecção de pessoas, mostrou-se que é possível o compartilhamento de camadas com outra rede já treinada. O treinamento, com pesos compartilhados e congelados de outra tarefa com semântica parecida, diminuiu minimamente o resultado quando comparado a metodologia convencional, porém com um tempo maior para conver-

gir. Também observou-se durante os experimentos que o Keras apresenta um problema na camada de BatchNormalization quando utilizada com essa técnica de transferência de aprendizagem, dificultando a convergência do treinamento e sendo necessário a reimplementação de tal camada. Tal rede alcançou 51.1 mAP em detecção de pessoas, resultado também no nível do estado da arte atual.

Também acrescentou-se à arquitetura a rede residual de pose, capaz de melhorar a tarefa de estimação de pose em até 2 pontos de mAP. Sua estrutura é simples e rápida, corrigindo as poses preditas de maneira a evitar combinação de pontos que não formem poses reais.

Por fim, desenvolveu-se o classificador de atividade, uma rede neural de múltiplas camadas que foi acoplada ao restante da estrutura de estimação de pose e detecção de pessoas. Seu treinamento foi realizado com *dataset* desenvolvido neste trabalho, contendo mais de 3000 poses em quatro diferentes atividades: correndo, andando, sentado e parado. Para avaliação dos diversos modelos experimentados, utilizou-se quatro métricas: precisão, *recall*, *f1 score* e acurácia. O melhor modelo alcançou 0.7371 de precisão, 0.7324 de *recall*, 0.7344 de *f1 score* e 0.7028 de acurácia, sendo estes valores calculados em 5 subgrupos de validação diferentes.

A análise da matriz de confusão mostrou que o modelo possui resultados consideravelmente bons para a atividade "sentado", com média de 91% de classificações corretas. As outras atividades apresentaram maiores níveis de confusão entre elas, indicando que há dificuldade em diferenciar padrões entre estas categorias. Isso indica a viabilidade do classificador de atividade por pose em imagens quando se tratando de poses bem distintas, e a necessidade de um outro fator para diferenciação quando há poses próximas.

Os experimentos de tempo de execução mostraram que o modelo final, isto é, a arquitetura completa capaz de detectar pessoas, estimar pose, corrigi-lás (PRN) e classificar a atividade, é possível de ser aplicada em tempo real, quando executada em uma GPU. A adição do classificador não influenciou no resultado de nenhuma outra tarefa e contribuiu irrisoriamente para o tempo de execução total. É importante notar que o compartilhamento do *backbone* entre as subredes de detecção de pessoas e detecção de *keypoints*, e a execução destas em paralelo, contribuíram significativamente para os 20.3 frames por segundo, em uma GTX 1080 TI, do modelo final (*backbone* ResNet50)

Dessa forma, conclui-se que é possível classificar a atividade de pessoas através de suas poses em imagens, e ainda utilizar uma única arquitetura capaz de realizar todo o

procedimento necessário para isso. Graças a metodologia *bottom-up*, é viável a aplicação do modelo completo quando realizada em uma GPU. Entretanto, é fundamental que as atividades de treinamento do modelo sejam descritas principalmente pela posição do corpo de uma pessoa, e que estas atividades apresentem poses mais distintas possíveis, evitando confusões de classificação.

Para trabalhos futuros propõe-se:

* reestruturar a arquitetura para lidar com poses em três dimensões;
* implementar diferentes modelos de *backbone* para melhorias de resultado ou performance;
* aumentar o *dataset* de treinamento e verificar a eficiência em poses mais complexas;
* adicionar modelos de redes neurais capazes de lidar com características temporais e verificar se há melhorias na classificação de poses parecidas;
* aplicar a metodologia desenvolvida neste trabalho em projetos com tomadas de decisões autônomas.

Referências[1]

ABADI, M.; AGARWAL, A.; BARHAM, P.; AL. et. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: ⟨http://tensorflow.org/⟩. Citado na página 52.

AGARAP, A. F. M. Deep Learning using Rectified Linear Units (ReLU). n. 1, p. 2–8, 2018. Citado na página 47.

ANDRILUKA, M.; PISHCHULIN, L.; GEHLER, P.; SCHIELE, B. 2D Human Pose Estimation : New Benchmark and State of the Art Analysis. In: *IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2014. Citado 2 vezes nas páginas 61 e 62.

BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, IEEE Press, Piscataway, NJ, USA, v. 5, n. 2, p. 157–166, mar. 1994. ISSN 1045-9227. Disponível em: ⟨https://doi.org/10.1109/72.279181⟩. Citado na página 43.

BIANCO, S.; CADENE, R.; CELONA, L.; NAPOLETANO, P. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, v. 6, p. 64270–64277, 2018. ISSN 2169-3536. Citado 2 vezes nas páginas 69 e 70.

CAO, Z.; SIMON, T.; WEI, S.-E.; SHEIKH, Y. Realtime multi-person 2d pose estimation using part affinity fields. In: *CVPR*. [S.l.: s.n.], 2017. Citado 8 vezes nas páginas 17, 21, 40, 67, 76, 77, 81 e 86.

CARREIRA, J.; ZISSERMAN, A. Quo Vadis , Action Recognition ? A New Model and the Kinetics Dataset. In: *IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2017. p. 4724–4733. Citado na página 18.

CARUANA, R.; LAWRENCE, S.; GILES, C. L. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In: . [S.l.: s.n.], 2000. v. 13, p. 402–408. Citado na página 33.

CHEN, Y.; WANG, Z.; PENG, Y.; ZHANG, Z.; YU, G.; SUN, J. Cascaded Pyramid Network for Multi-Person Pose Estimation. In: *IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2018. Citado na página 18.

CHOLLET, F. *Deep Learning with Python*. 1st. ed. Greenwich, CT, USA: Manning Publications Co., 2017. ISBN 1617294438, 9781617294433. Citado na página 52.

CHOLLET, F. *et al. Keras*. 2015. ⟨https://keras.io⟩. Citado na página 52.

CIREsAN, D. C.; MEIER, U.; MASCI, J.; GAMBARDELLA, L. M.; SCHMIDHUBER, J. Flexible, high performance convolutional neural networks for image classification. In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*. AAAI Press, 2011. (IJCAI'11), p. 1237–1242. ISBN 978-1-57735-514-4. Disponível em: ⟨http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-210⟩. Citado na página 33.

[1] De acordo com a Associação Brasileira de Normas Técnicas. NBR 6023.

DENG, J.; DONG, W.; SOCHER, R.; LI, L.-j.; LI, K.; FEI-FEI, L. ImageNet : A Large-Scale Hierarchical Image Database. In: *IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2009. p. 2–9. Citado 3 vezes nas páginas 15, 44 e 57.

EVANGELIDIS, G.; SINGH, G.; HORAUD, R. Skeletal quads: Human action recognition using joint quadruples. *Proceedings - International Conference on Pattern Recognition*, p. 4513–4518, 2014. ISSN 10514651. Citado na página 18.

EVERINGHAM, M.; GOOL, L. V.; WILLIAMS, C. K. I.; WINN, J. The P ASCAL Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, p. 303–338, 2010. Citado na página 15.

FANG, H.-s.; XIE, S.; TAI, Y.-w.; LU, C.; YOUTU, T. RMPE: Regional Multi-Person Pose Estimation. In: *IEEE International Conference on Computer Vision*. [S.l.: s.n.], 2017. Citado na página 17.

FERNEDA, E. Redes neurais e sua aplicação em sistemas de recuperação de informação. p. 25–30, 2006. Citado na página 23.

GAISER, H.; VRIES, M. de; LACATUSU, V.; AL. et. *fizyr/keras-retinanet: 0.5.0*. 2018. Disponível em: ⟨https://doi.org/10.5281/zenodo.1464720⟩. Citado 3 vezes nas páginas 52, 58 e 101.

GARCIA-GARCIA, A.; ORTS, S.; OPREA, S.; VILLENA-MARTINEZ, V.; RODRÍGUEZ, J. G. A review on deep learning techniques applied to semantic segmentation. *CoRR*, abs/1704.06857, 2017. Citado na página 15.

GEIGER, A.; LENZ, P.; STILLER, C.; URTASUN, R. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013. Citado na página 15.

GIRSHICK, R. Fast r-cnn. In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. Washington, DC, USA: IEEE Computer Society, 2015. (ICCV '15), p. 1440–1448. ISBN 978-1-4673-8391-2. Disponível em: ⟨http://dx.doi.org/10.1109/ICCV.2015.169⟩. Citado na página 38.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*. [S.l.: s.n.], 2010. Citado na página 43.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: The MIT Press, 2016. ISBN 0262035618, 9780262035613. Citado 2 vezes nas páginas 26 e 29.

HAYKIN, S. *Neural Networks: A Comprehensive Foundation*. 2nd. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998. ISBN 0132733501. Citado na página 26.

HAYKIN, S. *Redes Neurais - 2ed.* Bookman, 2001. ISBN 9788573077186. Disponível em: ⟨https://books.google.com.br/books?id=lBp0X5qfyjUC⟩. Citado na página 24.

HAYKIN, S. S. *Neural networks and learning machines*. Third. Upper Saddle River, NJ: Pearson Education, 2009. Citado na página 28.

HE, K.; GKIOXARI, G.; DOLL, P.; GIRSHICK, R. Mask R-CNN. In: *IEEE International Conference on Computer Vision*. [S.l.: s.n.], 2017. Citado 5 vezes nas páginas 17, 20, 38, 43 e 44.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, p. 770–778, 2015. Citado 3 vezes nas páginas 17, 43 e 44.

HINTON, G. Dropout : A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, v. 15, p. 1929–1958, 2014. Citado 3 vezes nas páginas 35, 36 e 51.

IOFFE, S.; SZEGEDY, C. Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2014. Citado 2 vezes nas páginas 34 e 51.

JI, S.; XU, W.; YANG, M.; YU, K. 3D Convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, v. 35, n. 1, p. 221–231, 2013. ISSN 01628828. Citado na página 18.

KINGMA, D. P.; BA, J. Adam: A Method for Stochastic Optimization. In: *International Conference on Learning Representations*. [s.n.], 2015. p. 1–15. ISBN 9781450300728. ISSN 09252312. Disponível em: ⟨http://arxiv.org/abs/1412.6980⟩. Citado na página 58.

KOCABAS, M.; KARAGOZ, S.; AKBAS, E. MultiPoseNet: Fast Multi-Person Pose Estimation Using Pose Residual Network. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 11215 LNCS, p. 437–453, 2018. ISSN 16113349. Citado 14 vezes nas páginas 17, 40, 41, 42, 48, 49, 50, 58, 75, 76, 77, 79, 81 e 86.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. USA: Curran Associates Inc., 2012. (NIPS'12), p. 1097–1105. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=2999134.2999257⟩. Citado na página 15.

KUHLMAN, D. *A Python Book: Beginning Python, Advanced Python, and Python Exercises*. PLATYPUS GLOBAL MEDIA, 2011. ISBN 9780984221233. Disponível em: ⟨https://books.google.es/books?id=1FL-ygAACAAJ⟩. Citado na página 52.

LECUN, Y.; BOSER, B.; HENDERSON, D. Backpropagation Applied t o Handwritten Zip Code Recognition. *Neural Computation*, v. 551, p. 541–551, 1989. Citado 2 vezes nas páginas 15 e 29.

LI, M.; ZHOU, Z.; LI, J.; LIU, X. Bottom-up Pose Estimation of Multiple Person with Bounding Box Constraint. In: *International Conference on Pattern Recognition*. [S.l.: s.n.], 2018. Citado 6 vezes nas páginas 7, 17, 40, 41, 81 e 86.

LIN, T.-y.; AI, F.; DOLL, P. Focal Loss for Dense Object Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018. Citado 9 vezes nas páginas 9, 20, 38, 39, 40, 47, 58, 76 e 101.

LIN, T.-y.; DOLL, P.; GIRSHICK, R.; HE, K.; HARIHARAN, B.; BELONGIE, S.; AI, F. Feature Pyramid Networks for Object Detection. *IEEE Conference on Computer Vision and Pattern Recognition*, 2017. Citado 2 vezes nas páginas 43 e 45.

LIN, T.-y.; ZITNICK, C. L.; DOLL, P. Microsoft COCO : Common Objects in Context. In: *European Conference on Computer Vision*. [S.l.: s.n.], 2014. p. 1–15. Citado 4 vezes nas páginas 15, 17, 39 e 61.

LIU, W.; ANGUELOV, D.; ERHAN, D.; SZEGEDY, C.; REED, S.; FU, C.-y.; BERG, A. C. SSD : Single Shot MultiBox Detector. 2015. Citado 3 vezes nas páginas 38, 39 e 81.

LUVIZON, D. C.; PICARD, D.; TABIA, H. 2D / 3D Pose Estimation and Action Recognition using Multitask Deep Learning. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2018. Citado na página 18.

NICKOLLS, J.; BUCK, I.; GARLAND, M.; SKADRON, K. Scalable parallel programming with cuda. *Queue*, ACM, New York, NY, USA, v. 6, n. 2, p. 40–53, mar. 2008. ISSN 1542-7730. Disponível em: ⟨http://doi.acm.org/10.1145/1365490.1365500⟩. Citado na página 53.

POPPE, R. A survey on vision-based human action recognition. *Image and Vision Computing*, Elsevier B.V., v. 28, n. 6, p. 976–990, 2010. ISSN 02628856. Disponível em: ⟨http://dx.doi.org/10.1016/j.imavis.2009.11.014⟩. Citado 2 vezes nas páginas 16 e 18.

QIU, Z.; YAO, T.; MEI, T. Learning Spatio-Temporal Representation with Pseudo-3D Residual Networks. *Proceedings of the IEEE International Conference on Computer Vision*, v. 2017-October, p. 5534–5542, 2017. ISSN 15505499. Citado na página 18.

REDMON, J.; DIVVALA, S. K.; GIRSHICK, R. B.; FARHADI, A. You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, p. 779–788, 2015. Citado 2 vezes nas páginas 20 e 38.

REN, S.; HE, K.; GIRSHICK, R.; SUN, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. Cambridge, MA, USA: MIT Press, 2015. (NIPS'15), p. 91–99. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=2969239.2969250⟩. Citado na página 38.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. In: RUMELHART, D. E.; MCCLELLAND, J. L.; GROUP, C. P. R. (Ed.). Cambridge, MA, USA: MIT Press, 1986. cap. Learning Internal Representations by Error Propagation, p. 318–362. ISBN 0-262-68053-X. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=104279.104293⟩. Citado na página 26.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. ISBN 0136042597, 9780136042594. Citado 2 vezes nas páginas 15 e 25.

SANTURKAR, S.; TSIPRAS, D.; ILYAS, A.; MADRY, A. How does batch normalization help optimization? In: BENGIO, S.; WALLACH, H.; LAROCHELLE, H.; GRAUMAN, K.; CESA-BIANCHI, N.; GARNETT, R. (Ed.). *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 2018. p. 2483–2493. Disponível em: ⟨http://papers.nips.cc/paper/7515-how-does-batch-normalization-help-optimization.pdf⟩. Citado na página 34.

SCHERER, D.; MüLLER, A.; BEHNKE, S. Evaluation of pooling operations in convolutional architectures for object recognition. In: *Proceedings of the 20th International Conference on Artificial Neural Networks: Part III*. Berlin, Heidelberg: Springer-Verlag, 2010. (ICANN'10), p. 92–101. ISBN 3-642-15824-2, 978-3-642-15824-7. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=1886436.1886447⟩. Citado na página 33.

SHELHAMER, E.; LONG, J.; DARRELL, T. Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, IEEE Computer Society, Washington, DC, USA, v. 39, n. 4, p. 640–651, abr. 2017. ISSN 0162-8828. Disponível em: ⟨https://doi.org/10.1109/TPAMI.2016.2572683⟩. Citado na página 36.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. Citado na página 38.

SONG, S.; LAN, C.; XING, J.; ZENG, W.; LIU, J. An End-to-End Spatio-Temporal Attention Model for Human Action Recognition from Skeleton Data. In: *AAAI Conference on Artificial Intelligence*. [s.n.], 2017. p. 4263–4270. ISBN 9780124160088. ISSN 0191-9601. Disponível em: ⟨http://arxiv.org/abs/1611.06067⟩. Citado na página 18.

TOSHEV, A. DeepPose: Human Pose Estimation via Deep Neural Networks. In: *IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2014. Citado na página 16.

TRAN, D.; WANG, H.; TORRESANI, L.; RAY, J.; LECUN, Y.; PALURI, M. A Closer Look at Spatiotemporal Convolutions for Action Recognition. In: *IEEE Conference on Computer Vision and Pattern Recognition*. [s.n.], 2018. ISBN 978-1-5386-6420-9. Disponível em: ⟨http://arxiv.org/abs/1711.11248⟩. Citado na página 18.

WANG, C.; WANG, Y.; YUILLE, A. L. An approach to pose-based action recognition. In: *IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2013. Citado na página 18.

WEI, S.-e. Convolutional Pose Machines. In: *IEEE Conference on Computer Vision and Pattern Recognition Shih-En*. [S.l.: s.n.], 2016. Citado 2 vezes nas páginas 57 e 81.

XIAO, B.; WU, H.; WEI, Y. Simple baselines for human pose estimation and tracking. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 11210 LNCS, p. 472–487, 2018. ISSN 16113349. Citado na página 17.

YANG, W.; WANG, Y.; MORI, G. Recognizing Human Actions from Still Images with Latent Poses. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2010. Citado na página 18.

YOSINSKI, J.; CLUNE, J.; BENGIO, Y.; LIPSON, H. How transferable are features in deep neural networks? In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. Cambridge, MA, USA: MIT Press, 2014. (NIPS'14), p. 3320–3328. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=2969033.2969197⟩. Citado na página 36.

ZHAO, Z.-q.; ZHENG, P. Object Detection with Deep Learning : A Review. *IEEE Transactions on Neural Networks and Learning Systems*, 2019. Citado na página 15.

ØRSTAVIK, M.; MIDTBø, T. En ny aera for uthenting av informasjon fra satellittbilder ved hjelp av maskinlaering mathilde Ørstavik og terje midtbø mathilde Ørstavik and terje midtbø, a new era for feature extraction in remotely sensed images by the use of machine learning. *Kart og Plan*, v. 77, p. 93–107, 01 2017. Citado na página 30.

Apêndice A – O Problema da Camada de BatchNormalization no Keras

Realizar transferência de aprendizado em *deep learning* é uma tarefa crucial, sendo capaz de reduzir o tempo de treinamento e a necessidade de grande quantidade de dados, reaproveitando um treinamento prévio. Tal técnica consiste em congelar boa parte da arquitetura de uma rede neural, mantendo os pesos do treinamento anterior e apenas atualizando suas últimas camadas. Porém, com camadas que se comportam diferentemente nas etapas de treinamento e de inferência, a transferência de aprendizado pode levar a resultados pobres. Esse é o caso da camada BatchNormalization do Keras.

A camada de *batch normalization* tem como função, de maneira simplificada, reescalonar o *batch* de treinamento através da substração do mesmo pela sua média e divisão pelo seu desvio padrão. Os pesos de toda rede são ajustados de acordo com a média dessa normalização de todos os *batchs* durante o treinamento. Uma vez finalizado o treino, essa média é mantida para a predição, reescalonando os dados futuros de acordo com o que foi aprendido durante o treino.

O Keras mantém o controle desse comportamento através de um mecanismo chamado *learning phase*, o qual determina se a rede está em treinamento (*learning phase true*) ou em inferência (*learning phase false*). Com isso, o Keras consegue determinar de que forma utilizar a camada de BatchNormalization (a camada de Dropout também segue a mesma lógica). Porém, ao realizar transferência de aprendizado congelando os pesos da camada, esse mecanismo falha.

Ao congelar os pesos da camada de *batch normalization*, seu comportamento deveria também ser considerado como de inferência, porém o mecanismo *learning phase* é atribuído a toda rede, fazendo com que esta camada entre no modo de treinamento. Apesar de seus pesos não serem atualizados (problema corrigido na versão 2.1.3), esta camada continua a extrair a média e a variância dos *batchs* dos novos dados, porém treinando de acordo com estes apenas as últimas camadas, mantendo os pesos das outras de acordo com as estatísticas do *dataset* original. Além disso, durante a inferência, os dados serão reescalonados de acordo com a média e a variância do primeiro treinamento, apresentando dados em padrões diferentes para as últimas camadas treinadas durante a transferência de aprendizado.

Este problema foi descoberto por Vasilis Vryniotis que submeteu uma requisição[1] para a *staff* do Keras analisando os efeitos desse problema e sugerindo uma possível solução. Infelizmente, o problema ainda está em análise por parte da equipe do Keras e devido ao nível de profundidade, muitos usuários desconhecem a razão dos resultados ruins ao aplicar essa técnica.

Entretanto, a solução pode ser obtida ao adicionar um novo mecanismo para verificar, se mesmo em treinamento, qual deve ser o comportamento da camada de *batch normalization*. Neste caso, ao treinar com os pesos congelados, esta camada deve se comportar como se estivesse em inferência, utilizando as estatísticas de *batchs* do treinamento original e evitando inconsistências em inferências futuras. Tal solução foi desenvolvida neste trabalho e os resultados podem ser vistos na seção 4.4.1.

[1] https://github.com/keras-team/keras/pull/9965

Apêndice B – Esboços de implementação

O *backbone* descrito em 3.2.1 é constituído de uma ResNet mais uma FPN. Apesar de haver o modelo pronto da ResNet no Keras, existe um problema no *framework*, mais especificamente na camada de *batch normalization*, ao congelar os parâmetros e realizar *transfer learning*. Infelizmente, esse caso ocorre na metodologia de treinamento, e por isso a ResNet é implementada utilizando-se uma versão não oficial do Keras em que há a correção dessa camada. O problema é melhor discutido no apêndice A.

Algoritmo B.1 Parte do código da ResNet.
As camadas de batch normalization são implementadas utilizando o pacote não oficial keras resnet.

```
x = keras.layers.Conv2D(64, (7, 7), strides=(2, 2),
use_bias=False, name="conv1", padding="same")(inputs)
x = keras_resnet.layers.BatchNormalization(axis=axis,
epsilon=1e-5, freeze=freeze_bn, name="bn_conv1")(x)
x = keras.layers.Activation("relu", name="conv1_relu")(x)
x = keras.layers.MaxPooling2D((3, 3), strides=(2, 2),
padding="same", name="pool1")(x)

features = 64

outputs = []

for stage_id, iterations in enumerate(blocks):
    for block_id in range(iterations):
        x = block(
            features,
            stage_id,
            block_id,
            numerical_name=(block_id > 0 and numerical_names[stage_id]),
            freeze_bn=freeze_bn
        )(x)

    features *= 2

    outputs.append(x)

....
```

A saída do *backbone* são as quatro características piramidais P_2, P_3, P_4 e P_5, compartilhadas entre a rede detecção de *keypoints* e a rede de detecção de pessoas.

Algoritmo B.2 Parte do código da FPN utilizando Keras

```
pyramid_5 = keras.layers.Conv2D(
                filters=256,
                kernel_size=1,
                strides=1,
                padding="same",
                name="c5_reduced"
            )(c5)
upsampled_p5 = keras.layers.UpSampling2D(
    interpolation="bilinear",
    name="p5_upsampled",
    size=(2, 2)
)(pyramid_5)

pyramid_4 = keras.layers.Conv2D(
    filters=256,
    kernel_size=1,
    strides=1,
    padding="same",
    name="c4_reduced"
)(c4)

pyramid_4 = keras.layers.Add(
    name="p4_merged"
)([upsampled_p5, pyramid_4])

upsampled_p4 = keras.layers.UpSampling2D(
    interpolation="bilinear",
    name="p4_upsampled",
    size=(2, 2)
)(pyramid_4)

....
```

Algoritmo B.3 Rede para detecção de keypoints (3.2.3) acoplada ao backbone

```
self.D2 = KL.Conv2D(128, (3, 3), name="d2_1", padding="same")(pyramid_2)
self.D2 = KL.Conv2D(128, (3, 3), name="d2_1_2", padding="same")(self.D2)
self.D3 = KL.Conv2D(128, (3, 3), name="d3_1", padding="same")(pyramid_3)
self.D3 = KL.Conv2D(128, (3, 3), name="d3_1_2", padding="same")(self.D3)
self.D4 = KL.Conv2D(128, (3, 3), name="d4_1", padding="same")(pyramid_4)
self.D4 = KL.Conv2D(128, (3, 3), name="d4_1_2", padding="same")(self.D4)
self.D5 = KL.Conv2D(128, (3, 3), name="d5_1", padding="same")(pyramid_5)
self.D5 = KL.Conv2D(128, (3, 3), name="d5_1_2", padding="same")(self.D5)

self.D3 = KL.UpSampling2D((2, 2), name='d3_up')(self.D3)
self.D4 = KL.UpSampling2D((4, 4), name='d4_up')(self.D4)
self.D5 = KL.UpSampling2D((8, 8), name='d5_up')(self.D5)

self.concat = KL.concatenate([self.D2, self.D3, self.D4, self.D5]
                             , axis=-1)
self.D = KL.Conv2D(512, (3, 3), activation="relu", padding="SAME",
                   name="Dfinal_1")(self.concat)
self.keypoint_output = KL.Conv2D(self.nb_keypoints, (1, 1),
                       padding="SAME", name="Dfinal_2")(self.D)

....
```

As mesmas saídas do *backbone* P_2, P_3, P_4 e P_5 são as entradas da rede RetinaNet. Nesta parte, o código é uma modificação do disponibilizado por (GAISER *et al.*, 2018). É reaproveitado a estrutura desenvolvida por eles sem o *backbone*, sendo esta acoplada na saída da FPN. A camada final também é modificada para uma camada com um único neurônio, já que nosso objetivo é classificar apenas um tipo de objeto: pessoas.

Algoritmo B.4 Novas características piramidais propostas por (LIN; AI; DOLL, 2018) para RetinaNet

```
# "P6     obtida via a 3x3 stride-2 conv em C5"
P6 = keras.layers.Conv2D(feature_size, kernel_size=3, strides=2,
padding='same',name='P6')(C5)

# "P7     computada aplicando ReLU seguida de uma conv 3x3
#com stride-2 em P6"
P7 = keras.layers.Activation('relu', name='C6_relu')(P6)
P7 = keras.layers.Conv2D(feature_size, kernel_size=3, strides=2,
padding='same', name='P7')(P7)
```

Até aqui foi implementado *backbone*, a subrede para detecção de *keypoints* e a subrede para detecção de pessoas, sendo que a saída dessas duas últimas deve ser

Algoritmo B.5 Regressão do bounding boxes com as features P_2, P_3, P_4, P_5, P_6 *e* P_7

```
if keras.backend.image_data_format() == 'channels_first':
    inputs  = keras.layers.Input(shape=(pyramid_feature_size,
                                        None, None))
else:
    inputs  = keras.layers.Input(shape=(None, None,
                                        pyramid_feature_size))
outputs = inputs
for i in range(4):
    outputs = keras.layers.Conv2D(
        filters=regression_feature_size,
        activation='relu',
        name='pyramid_regression_{}'.format(i),
        **options
    )(outputs)

outputs = keras.layers.Conv2D(num_anchors * 4,
name='pyramid_regression', **options)(outputs)
if keras.backend.image_data_format() == 'channels_first':
    outputs = keras.layers.Permute((2, 3, 1),
    name='pyramid_regression_permute')(outputs)
outputs = keras.layers.Reshape((-1, 4),
name='pyramid_regression_reshape')(outputs)

return keras.models.Model(inputs=inputs, outputs=outputs, name=name)

....
```

aplicada à entrada da rede residual de poses. Para isso, elas devem ser preprocessadas para sincronizarem suas dimensões e aplicar o algoritmo 3.1. O código abaixo é a implementação de uma camada neural que realiza tal pré-processamento em Tensorflow.

O pré-processamento gera entradas nas dimensões $N \times 56 \times 36 \times j$, onde N é o número de *bounding boxes* e j o número de *keypoints* detectados.

Cada pose, após estimada e associada a cada pessoa, é classificada em alguma das atividades em que o classificador foi treinado. O modelo final possui como saída a composição da saída de todas as etapas, ou seja, as coordenadas das pessoas detectadas, as poses estimadas e a classificação da atividade para cada pose de cada pesso. O sumário da rede neural contendo todas as camadas está disponível no apêndice C.

Algoritmo B.6 Classificação do objeto

```
if keras.backend.image_data_format() == 'channels_first':
    inputs  = keras.layers.Input(shape=(pyramid_feature_size,
                                        None, None))
else:
    inputs  = keras.layers.Input(shape=(None, None,
                                        pyramid_feature_size))
outputs = inputs
for i in range(4):
    outputs = keras.layers.Conv2D(
        filters=classification_feature_size,
        activation='relu',
        name='pyramid_classification_{}'.format(i),
        kernel_initializer=keras.initializers
        .normal(mean=0.0, stddev=0.01, seed=None),
        bias_initializer='zeros',
        **options
    )(outputs)

outputs = keras.layers.Conv2D(
    filters=num_classes * num_anchors,
    kernel_initializer=keras.initializers
    .normal(mean=0.0, stddev=0.01, seed=None),
    bias_initializer=initializers
    .PriorProbability(probability=prior_probability),
    name='pyramid_classification',
    **options
)(outputs)

# reshape output and apply sigmoid
if keras.backend.image_data_format() == 'channels_first':
    outputs = keras.layers.Permute((2, 3, 1),
    name='pyramid_classification_permute')(outputs)
outputs = keras.layers.Reshape((-1, num_classes),
name='pyramid_classification_reshape')(outputs)
outputs = keras.layers.Activation('sigmoid',
name='pyramid_classification_sigmoid')(outputs)

return keras.models.Model(inputs=inputs, outputs=outputs, name=name)
....
```

Algoritmo B.7 Camada de pré-processamento para a PRN
Camada intermediária desenvolvida com o tensorflow para ajustes das entradas na PRN3.2.4

```
heatmap = inputs[0]
bbox = inputs[1]
scores = inputs[2]

heatmap = K.tf.Print(heatmap, [heatmap], "heatmap: ")
indices = tf.where(K.greater(scores[0], 0.5))
filtered_boxes = K.tf.gather_nd(bbox[0], indices)
filtered_boxes = filtered_boxes / 4
x1,y1,x2,y2 = filtered_boxes[:,0], filtered_boxes[:,1],
filtered_boxes[:,2], filtered_boxes[:,3]
filtered_boxes = tf.stack([y1,x1,y2,x2], axis=1)
#tf.image.crop_and_resize bbox order

filtered_boxes = filtered_boxes / 119

filtered_boxes = K.tf.Print(filtered_boxes,
[filtered_boxes], "filtered_boxes: ")
shape_boxes = K.shape(filtered_boxes)
heatmap_c = tf.image.crop_and_resize(heatmap, filtered_boxes,
box_ind=tf.zeros(shape_boxes[0], tf.int32), crop_size=(56,36))

heatmap_c = K.tf.Print(heatmap_c, [heatmap_c], "heatmap_cropped: ")
# heatmap_c = K.tf.expand_dims(heatmap_c, axis=0)

return heatmap_c
```

Algoritmo B.8 Rede Residual de Poses

```
input = Input(shape=(56, 36, 18))
y = Flatten(name='prn_flatten')(input)
x = Dense(1024, activation='relu', name="prn_dense_1")(y)
x = Dropout(0.5, name='do_1')(x)
x = Dense(1024, activation='relu', name="prn_dense_2")(x)
x = Dropout(0.5, name='do_2')(x)
x = Dense(36 * 56 * 18, activation='relu', name='prn_dense_3')(x)
x = keras.layers.Add(name="prn_dense1_add_dense2")([x, y])
x = keras.layers.Activation('softmax', name='prn_activation')(x)
x = Reshape((56, 36, 18), name='prn_reshape')(x)
model = Model(inputs=input, outputs=x)
# print(model.summary())
return model
```

Algoritmo B.9 Classificador das poses estimadas

```
x = Dense(3072, activation='relu',
          name="prn_class_dense_1")(self.prn_output)
x = keras.layers.BatchNormalization()(x)
x = Dropout(0.5, name='do_class_1')(x)
x = Dense(128, activation='relu', name="prn_class_dense_2")(x)
x = keras.layers.BatchNormalization()(x)
x = Dropout(0.5, name='do_class_2')(x)

# x = Dense(36, activation='relu', name="prn_dense_3")(x)
# x = BatchNormalization()(x)
# x = Dropout(0.3, name='do_3')(x)
# x = keras.layers.Add(name="prn_dense1_add_dense2")([x, y])
self.class_output = Dense(4, activation='softmax', name="activ")(x)
output.append(self.class_output)
```

Apêndice C – Sumário da Arquitetura Completa

O sumário de uma rede neural é a descrição textual de toda sua arquitetura. O Keras fornece um método para sumarizar seus modelos, contendo descrições de:

- todas as camadas com nome e tipo;
- as respectivas ordens de cada camada e suas conexões;
- as dimensões dos vetores esperados na entrada e saída de cada camada;
- o número de parâmetros (pesos) em cada camada;
- o total de parâmetros do modelo assim como a quantidade de quais são e não são treináveis.

O sumário da arquitetura final deste trabalho é apresentado a seguir.

Layer (type)	Output Shape	Param #	Connected to
inputs (InputLayer)	(None, None, None, 3	0	
conv1 (Conv2D)	(None, None, None, 6	9408	inputs[0][0]
bn_conv1 (BatchNormalization)	(None, None, None, 6	256	conv1[0][0]
conv1_relu (Activation)	(None, None, None, 6	0	bn_conv1[0][0]
pool1 (MaxPooling2D)	(None, None, None, 6	0	conv1_relu[0][0]
res2a_branch2a (Conv2D)	(None, None, None, 6	4096	pool1[0][0]
bn2a_branch2a (BatchNormalizati	(None, None, None, 6	256	res2a_branch2a[0][0]
res2a_branch2a_relu (Activation	(None, None, None, 6	0	bn2a_branch2a[0][0]
padding2a_branch2b (ZeroPadding	(None, None, None, 6	0	res2a_branch2a_relu[0][0]
res2a_branch2b (Conv2D)	(None, None, None, 6	36864	padding2a_branch2b[0][0]
bn2a_branch2b (BatchNormalizati	(None, None, None, 6	256	res2a_branch2b[0][0]
res2a_branch2b_relu (Activation	(None, None, None, 6	0	bn2a_branch2b[0][0]
res2a_branch2c (Conv2D)	(None, None, None, 2	16384	res2a_branch2b_relu[0][0]
res2a_branch1 (Conv2D)	(None, None, None, 2	16384	pool1[0][0]
bn2a_branch2c (BatchNormalizati	(None, None, None, 2	1024	res2a_branch2c[0][0]
bn2a_branch1 (BatchNormalizatio	(None, None, None, 2	1024	res2a_branch1[0][0]

res2a (Add)	(None, None, None, 2	0	bn2a_branch2c[0][0] bn2a_branch1[0][0]
res2a_relu (Activation)	(None, None, None, 2	0	res2a[0][0]
res2b_branch2a (Conv2D)	(None, None, None, 6	16384	res2a_relu[0][0]
bn2b_branch2a (BatchNormalizati	(None, None, None, 6	256	res2b_branch2a[0][0]
res2b_branch2a_relu (Activation	(None, None, None, 6	0	bn2b_branch2a[0][0]
padding2b_branch2b (ZeroPadding	(None, None, None, 6	0	res2b_branch2a_relu[0][0]
res2b_branch2b (Conv2D)	(None, None, None, 6	36864	padding2b_branch2b[0][0]
bn2b_branch2b (BatchNormalizati	(None, None, None, 6	256	res2b_branch2b[0][0]
res2b_branch2b_relu (Activation	(None, None, None, 6	0	bn2b_branch2b[0][0]
res2b_branch2c (Conv2D)	(None, None, None, 2	16384	res2b_branch2b_relu[0][0]
bn2b_branch2c (BatchNormalizati	(None, None, None, 2	1024	res2b_branch2c[0][0]
res2b (Add)	(None, None, None, 2	0	bn2b_branch2c[0][0] res2a_relu[0][0]
res2b_relu (Activation)	(None, None, None, 2	0	res2b[0][0]
res2c_branch2a (Conv2D)	(None, None, None, 6	16384	res2b_relu[0][0]
bn2c_branch2a (BatchNormalizati	(None, None, None, 6	256	res2c_branch2a[0][0]
res2c_branch2a_relu (Activation	(None, None, None, 6	0	bn2c_branch2a[0][0]

padding2c_branch2b (ZeroPadding (None, None, None, 6 0 res2c_branch2a_relu[0][0]

_____

res2c_branch2b (Conv2D) (None, None, None, 6 36864 padding2c_branch2b[0][0]

_____

bn2c_branch2b (BatchNormalizati (None, None, None, 6 256 res2c_branch2b[0][0]

_____

res2c_branch2b_relu (Activation (None, None, None, 6 0 bn2c_branch2b[0][0]

_____

res2c_branch2c (Conv2D) (None, None, None, 2 16384 res2c_branch2b_relu[0][0]

_____

bn2c_branch2c (BatchNormalizati (None, None, None, 2 1024 res2c_branch2c[0][0]

_____

res2c (Add) (None, None, None, 2 0 bn2c_branch2c[0][0]
res2b_relu[0][0]

_____

res2c_relu (Activation) (None, None, None, 2 0 res2c[0][0]

_____

res3a_branch2a (Conv2D) (None, None, None, 1 32768 res2c_relu[0][0]

_____

bn3a_branch2a (BatchNormalizati (None, None, None, 1 512 res3a_branch2a[0][0]

_____

res3a_branch2a_relu (Activation (None, None, None, 1 0 bn3a_branch2a[0][0]

_____

padding3a_branch2b (ZeroPadding (None, None, None, 1 0 res3a_branch2a_relu[0][0]

_____

res3a_branch2b (Conv2D) (None, None, None, 1 147456 padding3a_branch2b[0][0]

_____

bn3a_branch2b (BatchNormalizati (None, None, None, 1 512 res3a_branch2b[0][0]

_____

res3a_branch2b_relu (Activation (None, None, None, 1 0 bn3a_branch2b[0][0]

_____

res3a_branch2c (Conv2D) (None, None, None, 5 65536 res3a_branch2b_relu[0][0]

_____

res3a_branch1 (Conv2D) (None, None, None, 5 131072 res2c_relu[0][0]

_____

bn3a_branch2c (BatchNormalizati (None, None, None, 5 2048 res3a_branch2c[0][0]

bn3a_branch1 (BatchNormalizatio (None, None, None, 5 2048 res3a_branch1[0][0]

res3a (Add) (None, None, None, 5 0 bn3a_branch2c[0][0]
bn3a_branch1[0][0]

res3a_relu (Activation) (None, None, None, 5 0 res3a[0][0]

res3b_branch2a (Conv2D) (None, None, None, 1 65536 res3a_relu[0][0]

bn3b_branch2a (BatchNormalizati (None, None, None, 1 512 res3b_branch2a[0][0]

res3b_branch2a_relu (Activation (None, None, None, 1 0 bn3b_branch2a[0][0]

padding3b_branch2b (ZeroPadding (None, None, None, 1 0 res3b_branch2a_relu[0][0]

res3b_branch2b (Conv2D) (None, None, None, 1 147456 padding3b_branch2b[0][0]

bn3b_branch2b (BatchNormalizati (None, None, None, 1 512 res3b_branch2b[0][0]

res3b_branch2b_relu (Activation (None, None, None, 1 0 bn3b_branch2b[0][0]

res3b_branch2c (Conv2D) (None, None, None, 5 65536 res3b_branch2b_relu[0][0]

bn3b_branch2c (BatchNormalizati (None, None, None, 5 2048 res3b_branch2c[0][0]

res3b (Add) (None, None, None, 5 0 bn3b_branch2c[0][0]
res3a_relu[0][0]

res3b_relu (Activation) (None, None, None, 5 0 res3b[0][0]

res3c_branch2a (Conv2D) (None, None, None, 1 65536 res3b_relu[0][0]

bn3c_branch2a (BatchNormalizati (None, None, None, 1 512 res3c_branch2a[0][0]

res3c_branch2a_relu (Activation	(None, None, None, 1	0	bn3c_branch2a[0][0]
padding3c_branch2b (ZeroPadding	(None, None, None, 1	0	res3c_branch2a_relu[0][0]
res3c_branch2b (Conv2D)	(None, None, None, 1	147456	padding3c_branch2b[0][0]
bn3c_branch2b (BatchNormalizati	(None, None, None, 1	512	res3c_branch2b[0][0]
res3c_branch2b_relu (Activation	(None, None, None, 1	0	bn3c_branch2b[0][0]
res3c_branch2c (Conv2D)	(None, None, None, 5	65536	res3c_branch2b_relu[0][0]
bn3c_branch2c (BatchNormalizati	(None, None, None, 5	2048	res3c_branch2c[0][0]
res3c (Add)	(None, None, None, 5	0	bn3c_branch2c[0][0] res3b_relu[0][0]
res3c_relu (Activation)	(None, None, None, 5	0	res3c[0][0]
res3d_branch2a (Conv2D)	(None, None, None, 1	65536	res3c_relu[0][0]
bn3d_branch2a (BatchNormalizati	(None, None, None, 1	512	res3d_branch2a[0][0]
res3d_branch2a_relu (Activation	(None, None, None, 1	0	bn3d_branch2a[0][0]
padding3d_branch2b (ZeroPadding	(None, None, None, 1	0	res3d_branch2a_relu[0][0]
res3d_branch2b (Conv2D)	(None, None, None, 1	147456	padding3d_branch2b[0][0]
bn3d_branch2b (BatchNormalizati	(None, None, None, 1	512	res3d_branch2b[0][0]
res3d_branch2b_relu (Activation	(None, None, None, 1	0	bn3d_branch2b[0][0]
res3d_branch2c (Conv2D)	(None, None, None, 5	65536	res3d_branch2b_relu[0][0]

bn3d_branch2c (BatchNormalizati	(None, None, None, 5	2048	res3d_branch2c[0][0]
res3d (Add)	(None, None, None, 5	0	bn3d_branch2c[0][0] res3c_relu[0][0]
res3d_relu (Activation)	(None, None, None, 5	0	res3d[0][0]
res4a_branch2a (Conv2D)	(None, None, None, 2	131072	res3d_relu[0][0]
bn4a_branch2a (BatchNormalizati	(None, None, None, 2	1024	res4a_branch2a[0][0]
res4a_branch2a_relu (Activation	(None, None, None, 2	0	bn4a_branch2a[0][0]
padding4a_branch2b (ZeroPadding	(None, None, None, 2	0	res4a_branch2a_relu[0][0]
res4a_branch2b (Conv2D)	(None, None, None, 2	589824	padding4a_branch2b[0][0]
bn4a_branch2b (BatchNormalizati	(None, None, None, 2	1024	res4a_branch2b[0][0]
res4a_branch2b_relu (Activation	(None, None, None, 2	0	bn4a_branch2b[0][0]
res4a_branch2c (Conv2D)	(None, None, None, 1	262144	res4a_branch2b_relu[0][0]
res4a_branch1 (Conv2D)	(None, None, None, 1	524288	res3d_relu[0][0]
bn4a_branch2c (BatchNormalizati	(None, None, None, 1	4096	res4a_branch2c[0][0]
bn4a_branch1 (BatchNormalizatio	(None, None, None, 1	4096	res4a_branch1[0][0]
res4a (Add)	(None, None, None, 1	0	bn4a_branch2c[0][0] bn4a_branch1[0][0]
res4a_relu (Activation)	(None, None, None, 1	0	res4a[0][0]

res4b_branch2a (Conv2D) (None, None, None, 2 262144 res4a_relu[0][0]

bn4b_branch2a (BatchNormalizati (None, None, None, 2 1024 res4b_branch2a[0][0]

res4b_branch2a_relu (Activation (None, None, None, 2 0 bn4b_branch2a[0][0]

padding4b_branch2b (ZeroPadding (None, None, None, 2 0 res4b_branch2a_relu[0][0]

res4b_branch2b (Conv2D) (None, None, None, 2 589824 padding4b_branch2b[0][0]

bn4b_branch2b (BatchNormalizati (None, None, None, 2 1024 res4b_branch2b[0][0]

res4b_branch2b_relu (Activation (None, None, None, 2 0 bn4b_branch2b[0][0]

res4b_branch2c (Conv2D) (None, None, None, 1 262144 res4b_branch2b_relu[0][0]

bn4b_branch2c (BatchNormalizati (None, None, None, 1 4096 res4b_branch2c[0][0]

res4b (Add) (None, None, None, 1 0 bn4b_branch2c[0][0]
res4a_relu[0][0]

res4b_relu (Activation) (None, None, None, 1 0 res4b[0][0]

res4c_branch2a (Conv2D) (None, None, None, 2 262144 res4b_relu[0][0]

bn4c_branch2a (BatchNormalizati (None, None, None, 2 1024 res4c_branch2a[0][0]

res4c_branch2a_relu (Activation (None, None, None, 2 0 bn4c_branch2a[0][0]

padding4c_branch2b (ZeroPadding (None, None, None, 2 0 res4c_branch2a_relu[0][0]

res4c_branch2b (Conv2D) (None, None, None, 2 589824 padding4c_branch2b[0][0]

bn4c_branch2b (BatchNormalizati (None, None, None, 2 1024 res4c_branch2b[0][0]

res4c_branch2b_relu (Activation (None, None, None, 2 0 bn4c_branch2b[0][0]

_____

res4c_branch2c (Conv2D) (None, None, None, 1 262144 res4c_branch2b_relu[0][0]

_____

bn4c_branch2c (BatchNormalizati (None, None, None, 1 4096 res4c_branch2c[0][0]

_____

res4c (Add) (None, None, None, 1 0 bn4c_branch2c[0][0]
res4b_relu[0][0]

_____

res4c_relu (Activation) (None, None, None, 1 0 res4c[0][0]

_____

res4d_branch2a (Conv2D) (None, None, None, 2 262144 res4c_relu[0][0]

_____

bn4d_branch2a (BatchNormalizati (None, None, None, 2 1024 res4d_branch2a[0][0]

_____

res4d_branch2a_relu (Activation (None, None, None, 2 0 bn4d_branch2a[0][0]

_____

padding4d_branch2b (ZeroPadding (None, None, None, 2 0 res4d_branch2a_relu[0][0]

_____

res4d_branch2b (Conv2D) (None, None, None, 2 589824 padding4d_branch2b[0][0]

_____

bn4d_branch2b (BatchNormalizati (None, None, None, 2 1024 res4d_branch2b[0][0]

_____

res4d_branch2b_relu (Activation (None, None, None, 2 0 bn4d_branch2b[0][0]

_____

res4d_branch2c (Conv2D) (None, None, None, 1 262144 res4d_branch2b_relu[0][0]

_____

bn4d_branch2c (BatchNormalizati (None, None, None, 1 4096 res4d_branch2c[0][0]

_____

res4d (Add) (None, None, None, 1 0 bn4d_branch2c[0][0]
res4c_relu[0][0]

_____

res4d_relu (Activation) (None, None, None, 1 0 res4d[0][0]

_____

res4e_branch2a (Conv2D) (None, None, None, 2 262144 res4d_relu[0][0]

bn4e_branch2a (BatchNormalizati	(None, None, None, 2	1024	res4e_branch2a[0][0]
res4e_branch2a_relu (Activation	(None, None, None, 2	0	bn4e_branch2a[0][0]
padding4e_branch2b (ZeroPadding	(None, None, None, 2	0	res4e_branch2a_relu[0][0]
res4e_branch2b (Conv2D)	(None, None, None, 2	589824	padding4e_branch2b[0][0]
bn4e_branch2b (BatchNormalizati	(None, None, None, 2	1024	res4e_branch2b[0][0]
res4e_branch2b_relu (Activation	(None, None, None, 2	0	bn4e_branch2b[0][0]
res4e_branch2c (Conv2D)	(None, None, None, 1	262144	res4e_branch2b_relu[0][0]
bn4e_branch2c (BatchNormalizati	(None, None, None, 1	4096	res4e_branch2c[0][0]
res4e (Add)	(None, None, None, 1	0	bn4e_branch2c[0][0] res4d_relu[0][0]
res4e_relu (Activation)	(None, None, None, 1	0	res4e[0][0]
res4f_branch2a (Conv2D)	(None, None, None, 2	262144	res4e_relu[0][0]
bn4f_branch2a (BatchNormalizati	(None, None, None, 2	1024	res4f_branch2a[0][0]
res4f_branch2a_relu (Activation	(None, None, None, 2	0	bn4f_branch2a[0][0]
padding4f_branch2b (ZeroPadding	(None, None, None, 2	0	res4f_branch2a_relu[0][0]
res4f_branch2b (Conv2D)	(None, None, None, 2	589824	padding4f_branch2b[0][0]
bn4f_branch2b (BatchNormalizati	(None, None, None, 2	1024	res4f_branch2b[0][0]
res4f_branch2b_relu (Activation	(None, None, None, 2	0	bn4f_branch2b[0][0]

```
_____
res4f_branch2c (Conv2D)        (None, None, None, 1 262144      res4f_branch2b_relu[0][0]
_____
bn4f_branch2c (BatchNormalizati (None, None, None, 1 4096        res4f_branch2c[0][0]
_____
res4f (Add)                    (None, None, None, 1 0           bn4f_branch2c[0][0]
                                                                res4e_relu[0][0]
_____
res4f_relu (Activation)        (None, None, None, 1 0           res4f[0][0]
_____
res5a_branch2a (Conv2D)        (None, None, None, 5 524288      res4f_relu[0][0]
_____
bn5a_branch2a (BatchNormalizati (None, None, None, 5 2048        res5a_branch2a[0][0]
_____
res5a_branch2a_relu (Activation (None, None, None, 5 0           bn5a_branch2a[0][0]
_____
padding5a_branch2b (ZeroPadding (None, None, None, 5 0           res5a_branch2a_relu[0][0]
_____
res5a_branch2b (Conv2D)        (None, None, None, 5 2359296     padding5a_branch2b[0][0]
_____
bn5a_branch2b (BatchNormalizati (None, None, None, 5 2048        res5a_branch2b[0][0]
_____
res5a_branch2b_relu (Activation (None, None, None, 5 0           bn5a_branch2b[0][0]
_____
res5a_branch2c (Conv2D)        (None, None, None, 2 1048576     res5a_branch2b_relu[0][0]
_____
res5a_branch1 (Conv2D)         (None, None, None, 2 2097152     res4f_relu[0][0]
_____
bn5a_branch2c (BatchNormalizati (None, None, None, 2 8192        res5a_branch2c[0][0]
_____
bn5a_branch1 (BatchNormalizatio (None, None, None, 2 8192        res5a_branch1[0][0]
_____
res5a (Add)                    (None, None, None, 2 0           bn5a_branch2c[0][0]
                                                                bn5a_branch1[0][0]
_____
_____
```

res5a_relu (Activation) (None, None, None, 2 0 res5a[0][0]

---

res5b_branch2a (Conv2D) (None, None, None, 5 1048576 res5a_relu[0][0]

---

bn5b_branch2a (BatchNormalizati (None, None, None, 5 2048 res5b_branch2a[0][0]

---

res5b_branch2a_relu (Activation (None, None, None, 5 0 bn5b_branch2a[0][0]

---

padding5b_branch2b (ZeroPadding (None, None, None, 5 0 res5b_branch2a_relu[0][0]

---

res5b_branch2b (Conv2D) (None, None, None, 5 2359296 padding5b_branch2b[0][0]

---

bn5b_branch2b (BatchNormalizati (None, None, None, 5 2048 res5b_branch2b[0][0]

---

res5b_branch2b_relu (Activation (None, None, None, 5 0 bn5b_branch2b[0][0]

---

res5b_branch2c (Conv2D) (None, None, None, 2 1048576 res5b_branch2b_relu[0][0]

---

bn5b_branch2c (BatchNormalizati (None, None, None, 2 8192 res5b_branch2c[0][0]

---

res5b (Add) (None, None, None, 2 0 bn5b_branch2c[0][0]
res5a_relu[0][0]

---

res5b_relu (Activation) (None, None, None, 2 0 res5b[0][0]

---

res5c_branch2a (Conv2D) (None, None, None, 5 1048576 res5b_relu[0][0]

---

bn5c_branch2a (BatchNormalizati (None, None, None, 5 2048 res5c_branch2a[0][0]

---

res5c_branch2a_relu (Activation (None, None, None, 5 0 bn5c_branch2a[0][0]

---

padding5c_branch2b (ZeroPadding (None, None, None, 5 0 res5c_branch2a_relu[0][0]

---

res5c_branch2b (Conv2D) (None, None, None, 5 2359296 padding5c_branch2b[0][0]

---

bn5c_branch2b (BatchNormalizati (None, None, None, 5 2048 res5c_branch2b[0][0]

---

res5c_branch2b_relu (Activation (None, None, None, 5 0 bn5c_branch2b[0][0]

---

res5c_branch2c (Conv2D) (None, None, None, 2 1048576 res5c_branch2b_relu[0][0]

---

bn5c_branch2c (BatchNormalizati (None, None, None, 2 8192 res5c_branch2c[0][0]

---

res5c (Add) (None, None, None, 2 0 bn5c_branch2c[0][0]
res5b_relu[0][0]

---

res5c_relu (Activation) (None, None, None, 2 0 res5c[0][0]

---

c5_reduced (Conv2D) (None, None, None, 2 524544 res5c_relu[0][0]

---

p5_upsampled (UpSampling2D) (None, None, None, 2 0 c5_reduced[0][0]

---

c4_reduced (Conv2D) (None, None, None, 2 262400 res4f_relu[0][0]

---

p4_merged (Add) (None, None, None, 2 0 p5_upsampled[0][0]
c4_reduced[0][0]

C5_reduced (Conv2D) (None, None, None, 2 524544 res5c_relu[0][0]

---

p4_upsampled (UpSampling2D) (None, None, None, 2 0 p4_merged[0][0]

---

c3_reduced (Conv2D) (None, None, None, 2 131328 res3d_relu[0][0]

P5_upsampled (UpsampleLike) (None, None, None, 2 0 C5_reduced[0][0]
res4f_relu[0][0]

C4_reduced (Conv2D) (None, None, None, 2 262400 res4f_relu[0][0]

---

p3_merged (Add) (None, None, None, 2 0 p4_upsampled[0][0]
c3_reduced[0][0]

P4_merged (Add)	(None, None, None, 2	0	P5_upsampled[0][0] C4_reduced[0][0]
p3_upsampled (UpSampling2D)	(None, None, None, 2	0	p3_merged[0][0]
c2_reduced (Conv2D)	(None, None, None, 2	65792	res2c_relu[0][0]
P4_upsampled (UpsampleLike)	(None, None, None, 2	0	P4_merged[0][0] res3d_relu[0][0]
C3_reduced (Conv2D)	(None, None, None, 2	131328	res3d_relu[0][0]
P6 (Conv2D)	(None, None, None, 2	4718848	res5c_relu[0][0]
p2_merged (Add)	(None, None, None, 2	0	p3_upsampled[0][0] c2_reduced[0][0]
p3 (Conv2D)	(None, None, None, 2	590080	p3_merged[0][0]
p4 (Conv2D)	(None, None, None, 2	590080	p4_merged[0][0]
P3_merged (Add)	(None, None, None, 2	0	P4_upsampled[0][0] C3_reduced[0][0]
C6_relu (Activation)	(None, None, None, 2	0	P6[0][0]
p2 (Conv2D)	(None, None, None, 2	590080	p2_merged[0][0]
d3_1 (Conv2D)	(None, None, None, 1	295040	p3[0][0]
d4_1 (Conv2D)	(None, None, None, 1	295040	p4[0][0]
d5_1 (Conv2D)	(None, None, None, 1	295040	c5_reduced[0][0]
P3 (Conv2D)	(None, None, None, 2	590080	P3_merged[0][0]

P4 (Conv2D)	(None, None, None, 2	590080	P4_merged[0][0]
P5 (Conv2D)	(None, None, None, 2	590080	C5_reduced[0][0]
P7 (Conv2D)	(None, None, None, 2	590080	C6_relu[0][0]
d2_1 (Conv2D)	(None, None, None, 1	295040	p2[0][0]
d3_1_2 (Conv2D)	(None, None, None, 1	147584	d3_1[0][0]
d4_1_2 (Conv2D)	(None, None, None, 1	147584	d4_1[0][0]
d5_1_2 (Conv2D)	(None, None, None, 1	147584	d5_1[0][0]
anchors_0 (Anchors)	(None, None, 4)	0	P3[0][0]
anchors_1 (Anchors)	(None, None, 4)	0	P4[0][0]
anchors_2 (Anchors)	(None, None, 4)	0	P5[0][0]
anchors_3 (Anchors)	(None, None, 4)	0	P6[0][0]
anchors_4 (Anchors)	(None, None, 4)	0	P7[0][0]
regression_submodel (Model)	(None, None, 4)	2443300	P3[0][0] P4[0][0] P5[0][0] P6[0][0] P7[0][0]
d2_1_2 (Conv2D)	(None, None, None, 1	147584	d2_1[0][0]
d3_up (UpSampling2D)	(None, None, None, 1	0	d3_1_2[0][0]
d4_up (UpSampling2D)	(None, None, None, 1	0	d4_1_2[0][0]

d5_up (UpSampling2D) (None, None, None, 1 0 d5_1_2[0][0]

---

anchors (Concatenate) (None, None, 4) 0 anchors_0[0][0]
anchors_1[0][0]
anchors_2[0][0]
anchors_3[0][0]
anchors_4[0][0]

---

regression (Concatenate) (None, None, 4) 0 regression_submodel[1][0]
regression_submodel[2][0]
regression_submodel[3][0]
regression_submodel[4][0]
regression_submodel[5][0]

---

concatenate_1 (Concatenate) (None, None, None, 5 0 d2_1_2[0][0]
d3_up[0][0]
d4_up[0][0]
d5_up[0][0]

---

boxes (RegressBoxes) (None, None, 4) 0 anchors[0][0]
regression[0][0]

---

classification_submodel (Model) (None, None, 1) 2381065 P3[0][0]
P4[0][0]
P5[0][0]
P6[0][0]
P7[0][0]

Dfinal_1 (Conv2D) (None, None, None, 5 2359808 concatenate_1[0][0]

---

clipped_boxes (ClipBoxes) (None, None, 4) 0 inputs[0][0]
boxes[0][0]

---

classification (Concatenate) (None, None, 1) 0 classification_submodel[1][0]
classification_submodel[2][0]
classification_submodel[3][0]
classification_submodel[4][0]
classification_submodel[5][0]

Dfinal_2 (Conv2D) (None, None, None, 1 9747 Dfinal_1[0][0]

---

filtered_detections (FilterDete [(None, 300, 4), (No 0 clipped_boxes[0][0]

			classification[0][0]
lambda_1 (Lambda)	(None, 56, 36, 18)	0	Dfinal_2[0][0] filtered_detections[0][0] filtered_detections[0][1] filtered_detections[0][2]
prn_lambda_input (Lambda)	(None, 56, 36, 18)	0	lambda_1[0][0]
prn_flatten (Flatten)	(None, 36288)	0	prn_lambda_input[0][0]
prn_dense_1 (Dense)	(None, 1024)	37159936	prn_flatten[0][0]
do_1 (Dropout)	(None, 1024)	0	prn_dense_1[0][0]
prn_dense_2 (Dense)	(None, 36288)	37195200	do_1[0][0]
prn_dense1_add_dense2 (Add)	(None, 36288)	0	prn_dense_2[0][0] prn_flatten[0][0]
prn_activation (Activation)	(None, 36288)	0	prn_dense1_add_dense2[0][0]
prn_reshape (Reshape)	(None, 56, 36, 18)	0	prn_activation[0][0]
lambda_2 (Lambda)	(1, None, 56, 36, 18	0	prn_reshape[0][0]
prn_class_dense_1 (Dense)	(1, None, 56, 36, 30	58368	lambda_2[0][0]
batch_normalization_1 (BatchNor	(1, None, 56, 36, 30	12288	prn_class_dense_1[0][0]
do_class_1 (Dropout)	(1, None, 56, 36, 30	0	batch_normalization_1[0][0]
prn_class_dense_2 (Dense)	(1, None, 56, 36, 12	393344	do_class_1[0][0]
batch_normalization_2 (BatchNor	(1, None, 56, 36, 12	512	prn_class_dense_2[0][0]

```
do_class_2 (Dropout)         (1, None, 56, 36, 12 0          batch_normalization_2[0][0]
_____________________
activ (Dense)                (1, None, 56, 36, 4) 516        do_class_2[0][0]
==========================================================================
============================
Total params: 118,097,476
Trainable params: 117,984,836
Non-trainable params: 112,640
_____________________
```

Anexo A – Publicações Aceitas deste Trabalho

Journal of Image and Graphics

Acceptance Letter-Author

2019 2nd International Conference on Artificial Intelligence and Pattern Recognition-AIPR

August 16-18, 2019 | Beijing, China

Role: Author

Paper ID: T2044

Presentation title: An end-to-end approach for recognizing human activity in images using pose estimation

Dear Igor Muniz Soares, Cássio Vinhal and Gelson da Cruz Jr.,

Congratulations on the acceptance of your oral presentation listed above for **2019 AIPR** Annual International Conference!

You are invited to participate and make an oral presentation in **2019 2nd International Conference on Artificial Intelligence and Pattern Recognition (AIPR 2019)** that will be held in **Beijing, China** on **August 16-18, 2019**. It is hosted by North China University of Technology (NCUT), supported by School of Information of NCUT.

Your paper is accepted to be published into the journal as below:

Journal of Image and Graphics (JOIG; ISSN: 2301-3699)

Abstracting/Indexing: Ulrich's Periodicals Directory, Google Scholar, Crossref, IndexCopernicus, etc.

Attached in the letter are the reviewer's comments, plus notification of acceptance letter. To complete registration and include your paper in journal, you need to submit all required documents before the deadline. If any questions for documents submission please click Guideline for Registration in the 2nd page of attached notification file.

Checking List

- ☐ **Formatted Paper (word & pdf)**
- ☐ **scanned signed copyright form (jpg or pdf)**
- ☐ **Filled registration form(word)**
- ☐ **Payment Proof (jpg or pdf)**

Mail the above documents to aipr@iact.net BEFORE August 1, 2019.

We're looking forward to seeing you in **Beijing**!

Sincerely,

AIPR 2019 Conference Committees

E-mail: aipr@iact.net

Web: www.aipr.net

Acceptance Letter-Author

2019 2nd International Conference on Artificial Intelligence and Pattern Recognition-AIPR

August 16-18, 2019 | Beijing, China

Role: Author

Paper ID: T2044

Presentation title: An end-to-end approach for recognizing human activity in images using pose estimation

Dear Igor Muniz Soares, Cássio Vinhal and Gelson da Cruz Jr.,

Congratulations on the acceptance of your oral presentation listed above for **2019 AIPR** Annual International Conference!

You are invited to participate and make an oral presentation in **2019 2nd International Conference on Artificial Intelligence and Pattern Recognition (AIPR 2019)** that will be held in **Beijing, China** on **August 16-18, 2019**. It is hosted by North China University of Technology (NCUT), supported by School of Information of NCUT.

All registered accepted papers will be published in the International Conference Proceedings, which will be indexed by **Ei Compendex** and **Scopus** and submitted to be reviewed by Thomson Reuters Conference Proceedings Citation Index (ISI Web of Science).

Attached in the letter are the reviewer's comments, plus instructions for producing the camera-ready copy, and all other necessary documents. To complete registration and include your paper in proceedings, you need to submit all required documents before the deadline. If any questions for documents submission please click Guideline for Registration in the 2nd page of attached notification file.

Checking List

- ☐ **Formatted Paper (word & pdf)**
- ☐ **Filled registration form(word)**
- ☐ **CCS Concept and xml code**
- ☐ **Payment Proof (jpg or pdf)**

Mail the above documents to aipr@iact.net BEFORE July 25, 2019.

We're looking forward to seeing you in **Beijing**!

Sincerely,

AIPR 2019 Conference Committees

E-mail: aipr@iact.net

Web: www.aipr.net