GABRIEL ROSA E SILVA

DETECÇÃO DE OBJETOS EM IMAGENS UTILIZANDO TÉCNICAS DE APRENDIZAGEM PROFUNDA

Trabalho de Conclusão de Curso submetido ao Departamento de Engenharia Elétrica e Eletrônica da Universidade Federal de Santa Catarina para a obtenção do título de bacharel em Engenharia Elétrica.
Orientador: Eduardo Luiz Ortiz Batista.
Co-Orientador: Walter Antonio Gontijo

FLORIANÓPOLIS
2018

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Silva, Gabriel Rosa e
Detecção de objetos em imagens utilizando técnicas de aprendizagem profunda / Gabriel Rosa e Silva ; orientador, Eduardo Luiz Ortiz Batista, coorientador, Walter Antonio Gontijo, 2018.
72 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Elétrica, Florianópolis, 2018.

Inclui referências.

1. Engenharia Elétrica. 2. Aprendizagem profunda. 3. Detecção de objetos. 4. Redes neurais convolucionais. I. Batista, Eduardo Luiz Ortiz. II. Gontijo, Walter Antonio. III. Universidade Federal de Santa Catarina. Graduação em Engenharia Elétrica. IV. Título.

Gabriel Rosa e Silva

DETECÇÃO DE OBJETOS EM IMAGENS UTILIZANDO TÉCNICAS DE APRENDIZAGEM PROFUNDA

Este Trabalho foi julgado adequado para a obtenção do Título de Bacharel em Engenharia Elétrica e aprovado em sua forma final pela Banca Examinadora.

Florianópolis, 27 de dezembro de 2018.

Prof. Jean Vianei Leite, Dr.
Coordenador do Curso

Banca examinadora:

Prof. Eduardo Luiz Ortiz Batista, Dr.
Universidade Federal de Santa Catarina

Prof. Walter Antonio Gontijo
Universidade do Vale do Itajaí

Prof. Richard Demo Souza, Dr.
Universidade Federal de Santa Catarina

Eng. Fernando Ghedin
Audaces Automação e Informática Industrial Ltda.

Agradecimentos

Desejo expressar meu reconhecimento a todos que, de uma maneira ou outra, colaboraram na realização deste trabalho, em especial

> Ao professor Eduardo Luiz Ortiz Batista, pela orientação na elaboração deste trabalho.
> Ao Laboratório de Circuitos e Processamento de Sinais (LINSE), em particular ao Walter Antonio Gontijo, pelo auxílio e co-orientação na realização deste trabalho.
> Aos colegas do LINSE, pelas diversas discussões sobre os assuntos abordados no trabalho.

RESUMO

Este trabalho apresenta métodos de detecção de objetos em imagens utilizando técnicas de aprendizagem profunda. Em particular, duas redes neurais profundas utilizando camadas convolucionais são apresentadas. Tais redes, de diferentes complexidades e capacidades de generalização, são avaliadas em duas tarefas distintas: a detecção de pessoas em ambientes internos e a detecção de veículos de uma vista superior. A performance das redes nas duas aplicações é comparada para cenários com conjuntos de exemplos de diferentes tamanhos e qualidade, e também para modelos pré-treinados.

Palavras-chave: Aprendizagem profunda. Redes neurais convolucionais. Detecção de objetos. Detecção de pessoas. Detecção de veículos.

ABSTRACT

This work presents different methods for object detection in images using deep learning. In particular, two deep neural networks with convolutional layers are presented. Such networks, differing in both in complexity and generalization capacity, are applied to two distinct tasks: the detection of humans in an indoor space and vehicle detection from a top-down view. The performance of both networks is evaluated using datasets of varying size and quality, as well as pre-trained models.

Keywords: Deep learning. Convolutional neural networks. Object detection. Human detection. Vehicle detection.

Lista de Figuras

Lista de Tabelas

Sumário

CAPÍTULO 1

Introdução

A visão humana é uma ferramenta essencial para a interação com o meio. A sua importância é evidenciada pela alta complexidade do olho humano, que converte ondas luminosas em impulsos nervosos. No cérebro, a percepção da imagem a partir de tais impulsos ocorre de forma automática e involuntária.

Na Engenharia, diversos problemas requerem sistemas automatizados que possuam capacidade de extrair informações a partir de imagens digitais. Denominada visão computacional, essa tecnologia é parte essencial em diversas aplicações, como por exemplo rastreamento de objetos, reconhecimento facial, veículos autônomos, dentre outras. Uma etapa importante nessas aplicações é a detecção de objetos pertencentes a uma classe, como detecção de faces [1], detecção de veículos [2] e detecção de pedestres [3].

Nesse contexto, o foco deste trabalho é desenvolver um sistema de detecção de objetos utilizando técnicas de aprendizagem profunda. A função desse tipo de sistema é identificar em uma imagem digital se existem objetos pertencentes a uma determinada classe e, em caso positivo, identificar sua posição na imagem. A utilização de técnicas de aprendizagem profunda com tal objetivo se justifica pelos avanços re-

centes obtidos em aplicações como reconhecimento de documentos [4], classificação de objetos [5] e descrição textual de imagens [6].

1.1 Estado da arte

Detecção de objetos é um problema clássico na área de visão computacional. Métodos para resolver esse tipo de problema são propostos desde os anos 70, como por exemplo o método de detecção por seguimento de bordas de Yakimovsky [7]. Porém, a ampla utilização desses métodos só vem se tornando possível nas últimas duas décadas, o que se deve tanto ao desenvolvimento de técnicas mais eficientes, quanto à disponibilidade de processadores com maior capacidade. Nesse contexto, o sistema proposto por Viola e Jones [8] em 2001 foi um dos primeiros a proporcionar a detecção facial em tempo real utilizando-se uma versão modificada do classificador *AdaBoost*. Além disso, outro grande avanço na área foi proposto em 2005 por Dalal e Triggs [9], com um sistema de detecção de pedestres que utiliza descritores do tipo *Histogram of Oriented Gradients* para obter maior precisão comparado a sistemas anteriores.

Uma revisão da literatura em detecção de objetos é apresentada em [10]. Nesse trabalho, alguns descritores de atributos que podem ser extraídos de regiões de uma imagem, como *Scale-invariant Feature Transform (SIFT)* [11], *Bag of Features (BoF)* [12], *Histogram of Oriented Gradients (HOG)* [9] e *GIST* [13] são apresentados. Também são revisados métodos de classificação de descritores, como *K-Nearest Neighbor (KNN)* [14], *AdaBoost* [8] e *Support Vector Machines (SVMs)* [9]. São discutidas finalmente algumas métricas de avaliação de modelos, o que inclui precisão de acerto do tipo de objeto detectado e razão entre a interseção e união de *bounding boxes*.

Recentemente, o foco da comunidade acadêmica se voltou à utilização de redes neurais convolucionais para o problema de detecção de objetos, com destaque para métodos como *R-CNN* [15], *R-FCN* [16], *RetinaNet* [17] e *YOLO* [18]. De maneira geral, esses métodos têm levado a resultados que superam com boa margem os resultados obtidos com os métodos clássicos.

1.2 Objetivos

O objetivo central deste trabalho é avaliar as diferentes técnicas de aprendizagem profunda que têm sido aplicadas com sucesso à detecção de objetos.

Como objetivos específicos, tem-se:

- Avaliar técnicas de diferentes complexidades computacionais;
- Avaliar as técnicas nos seguintes cenários de aplicação: detecção de veículos, e detecção de pessoas;
- Comparar o desempenho das técnicas em função da qualidade e tamanho do conjunto de dados;
- Comparar diferentes estratégias de treinamento.

1.3 Organização do trabalho

O presente trabalho está organizado como descrito a seguir. No Capítulo 2, uma visão geral da fundamentação teórica deste trabalho é apresentada. No Capítulo 3, são detalhadas as técnicas e cenários avaliados. No Capítulo 4, os experimentos realizados e os resultados obtidos são apresentados. No Capítulo 5, as considerações finais são apresentadas.

CAPÍTULO 2

Fundamentação Teórica

Este capítulo é dedicado à apresentação da fundamentação teórica para o desenvolvimento do presente trabalho. Assim, noções gerais de aprendizagem de máquina são apresentadas, seguidas pela teoria de redes neurais e aprendizagem profunda e, por fim, definições sobre a tarefa de detecção de objetos em imagens.

2.1 Aprendizagem de máquina

Hoje em dia, graças aos avanços monumentais na área de inteligência artificial e sistemas computacionais, os computadores têm sido capazes de automatizar tarefas até recentemente realizadas exclusivamente por humanos. Os primeiros desenvolvimentos em tal área dependiam da habilidade do pesquisador em codificar o conhecimento requerido no próprio algoritmo. Uma metodologia mais efetiva foi então desenvolvida, envolvendo o fornecimento de exemplos a partir dos quais o computador pode extrair esse conhecimento. Tal metodologia é chamada de aprendizagem de máquina [19].

De maneira geral, os métodos de aprendizagem de máquina empregam algoritmos para analisar um **conjunto de exemplos**, refinando

parâmetros de um modelo adaptativo para realizar uma determinada **tarefa** com melhor **performance**, de acordo com uma determinada métrica. A tal processo de refinamento, dá-se o nome de treinamento [20].

2.1.1 Conjunto de exemplos

Um exemplo, também chamado de amostra, é um conjunto de características obtidas de forma quantitativa a partir de algum objeto ou evento que o sistema de aprendizagem de máquina deve processar. A um exemplo pode ser associada a saída esperada do sistema, chamada de alvo ou anotação [19]. A existência ou não de anotações em um conjunto de exemplos determina o método de aprendizagem de máquina empregado.

2.1.2 Métodos de aprendizagem

Os métodos podem ser divididos entre supervisionados (ou preditivos), empregados quando o conjunto possui anotações, e não supervisionados (ou descritivos), empregados quando o conjunto não possui anotações [21].

Modelos treinados com métodos não supervisionados são capazes de extrair padrões que relacionam exemplos de um determinado conjunto. Um exemplo clássico de aprendizagem não supervisionada são os algoritmos de *clustering*, que dividem um conjunto em diferentes grupos de exemplos de acordo com alguma métrica de similaridade [20].

Modelos treinados com métodos supervisionados são capazes de realizar inferência sobre novos exemplos. Se o resultado do processo de inferência for um dado contínuo, como por exemplo um valor de probabilidade, o problema recebe o nome de regressão. No caso onde são inferidos valores discretos, o problema é chamado de classificação. Na prática a classificação pode ser realizada inferindo um valor contínuo ao qual aplica-se uma regra de decisão, resultando em um valor discreto [21].

2.1.3 Performance

A performance de um sistema de aprendizagem de máquina é uma medida quantitativa do quão bem o sistema realiza a tarefa para o qual

foi desenvolvido. Geralmente deseja-se saber o quão bem o sistema se sai ao processar novos exemplos (exemplos nunca processados anteriormente). A capacidade de processar corretamente exemplos novos é chamada de generalização. Para medir a capacidade de generalização de um sistema, o conjunto de exemplos disponível é dividido em um conjunto de treinamento e um conjunto de teste. Durante o treinamento, o sistema não tem acesso ao conjunto de teste. Os parâmetros do modelo, portanto, são obtidos visando melhorar a performance do conjunto de treinamento. Após o treinamento, os parâmetros do modelo são fixados e a capacidade de generalização do sistema é avaliada a partir da performance obtida para o conjunto de teste [19].

O equilíbrio entre performance do conjunto de treinamento e performance do conjunto de teste é o desafio central no treinamento de modelos de aprendizagem de máquina. Se o sistema apresenta uma performance ruim no conjunto de treinamento, diz-se que ocorreu *underfitting*, ou seja, o modelo não se adequou aos exemplos de treinamento. Se o sistema apresenta boa performance no conjunto de treinamento e performance ruim no conjunto de testes, diz-se que ocorreu *overfitting*, ou seja, o modelo se adequou demais aos exemplos de treinamento, prejudicando sua capacidade de generalização [19].

2.1.4 A tarefa de detecção de objetos

Detecção de objetos em imagens é uma entre muitas tarefas que podem ser realizadas por sistemas de aprendizagem de máquina. A detecção pode ser formulada como um problema de regressão ou de classificação. Na prática, métodos que encaram a detecção como um problema de classificação tendem a apresentar melhores resultados [22] e [15].

Nessa formulação, a imagem na qual deseja-se detectar objetos é dividida em diferentes regiões. Cada região é classificada separadamente para indicar se ela representa ou não um objeto de determinada classe. Em caso positivo, informações de posição, tamanho e contorno da região são utilizadas para indicar a localização do objeto [23]. A localização precisa de um objeto só é possível se ele for englobado por uma das regiões consideradas. O desafio dessa metodologia consiste em encontrar um conjunto de regiões grande o suficiente para englobar as localizações onde existem objetos, mas não tão grande de forma que o tempo de processamento se torne inviável [15].

2.1.5 Classificação binária

Em aplicações cujo objetivo é detectar apenas uma classe de objetos, basta classificar se uma região pertence à classe ou não. Esse tipo de problema é chamado de classificação binária.

Um classificador binário indica se um exemplo $\boldsymbol{x}$ pertence ou não a uma determinada classe. Tomando como exemplo um sistema de aprendizagem de máquina que indica a probabilidade $p_c \in [0, 1]$ da entrada $\boldsymbol{x}$ pertencer à classe y, tem-se

$$p_c = p(y|\boldsymbol{x}, \boldsymbol{w}) \tag{2.1}$$

onde $\boldsymbol{w}$ é o modelo de classificação, o classificador binário pode ser definido como

$$y = f(p(y = 1|\boldsymbol{x}, \boldsymbol{w})) = f(p_c) = \begin{cases} 1, & \text{se } p_c \geq p_{th} \\ 0, & \text{se } p_c < p_{th} \end{cases} \tag{2.2}$$

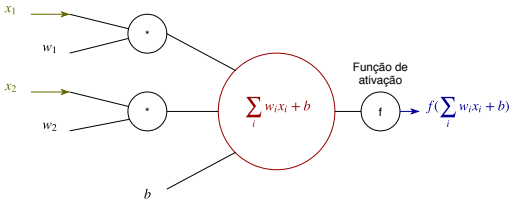
onde f é a regra de decisão que separa as classes usando um *threshold* de probabilidade p_{th} e y é um número inteiro que indica se x pertence $(y = 1)$ ou não $(y = 0)$ à classe [21].

2.2 Redes neurais

O conceito moderno de redes neurais é resultado de mais de meio século de pesquisa. Em 1958, o *perceptron* foi proposto pelo psicólogo Frank Rosenblatt como um modelo matemático simplificado de um neurônio [24]. A organização de vários *perceptrons* conectados em camadas distintas ficou conhecida por MLP (*multi-layer perceptron*). Se um MLP tem todos os *perceptrons* de uma camada conectados às camadas adjacentes, ele também pode ser chamado de rede neural *fully connected*, ou rede densa [25].

Hoje se sabe que, apesar do nome, as redes neurais não são uma representação fiel do funcionamento de um conjunto de neurônios em um cérebro [26]. Porém, essas ideias introduzidas há mais de meio século forneceram a base para grandes desenvolvimentos na área de inteligência artificial.

Figura 2.1: Diagrama gráfico de um *perceptron*



Fonte: Autor

2.2.1 Perceptron

O *perceptron* é um modelo de neurônio a partir do qual redes neurais podem ser compostas [25]. Ele pode ser definido matematicamente como a combinação de um classificador linear com uma função não-linear, chamada de função de ativação. Na Figura 2.1 é apresentado um diagrama gráfico de um *perceptron*.

Dado um vetor de entrada $\boldsymbol{x}$, um vetor de coeficientes $\boldsymbol{w}$ (também chamado de modelo) e um *bias* b, a relação de entrada e saída do *perceptron* pode ser definida como

$$y = f(\boldsymbol{x}^T \boldsymbol{w} + b) \tag{2.3}$$

onde f é a função de ativação, $\boldsymbol{x}^T$ é o vetor de entrada transposto e y é a saída do *perceptron*.

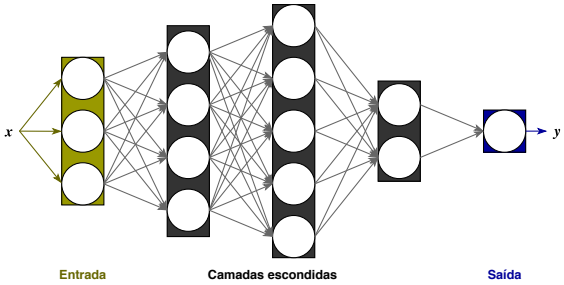
A saída do *perceptron* pode ter diversas interpretações, dependendo do modelo e da função de ativação utilizados. Por exemplo, se for utilizado um modelo treinado via regressão logística em combinação com a função de ativação $f(x) = \frac{e^x}{1+e^x}$ (função logística), a saída do *perceptron* representará a probabilidade de ocorrência de um determinado evento [27].

2.2.2 Redes densas

Redes densas consistem em um ou mais *perceptrons* organizados em camadas densamente conectadas, como ilustrado na Figura 2.2. Cada *perceptron* da rede possui o seu próprio vetor de coeficientes $\boldsymbol{w}$ e *bias* b. Os *perceptrons* podem ter diferentes funções de ativação.

A primeira camada da rede é chamada de camada de entrada. O va-

Figura 2.2: Diagrama gráfico de uma rede densa



Fonte: Autor

lor de entrada da rede é processado por cada *perceptron* dessa camada, resultando em valores escalares chamados de ativações. As ativações são concatenadas em um vetor que será processado pela camada seguinte.

A rede pode ter uma ou mais camadas intermediárias chamadas de camadas escondidas (*hidden layers*), cada uma com um determinado número de *perceptrons*. Cada camada escondida processa um vetor de ativações de entrada para produzir um novo vetor de ativações de saída.

A camada final, chamada de camada de saída, possui um *perceptron* associado a cada saída do sistema. Um classificador binário, por exemplo, apresenta apenas um *perceptron* em sua camada de saída. A entrada dessa última camada é um vetor de ativações e suas saídas são os dados de saída da rede [25].

Dado um exemplo $\boldsymbol{x}$ a ser processado por uma rede com $m > 2$ camadas ($m-2$ camadas escondidas) com n_i *perceptrons* cada, para $i = 1, ..., m$, cada *perceptron* possui o seu vetor de pesos $\boldsymbol{w_{i,j}}$, *bias* $b_{i,j}$ e função de ativação $f_{i,j}$, para $j = 1, ..., n_i$. As ativações $a_{i,j}$ são dadas por

$$a_{i,j} = f_{i,j}(\tilde{\mathbf{a}}_{\boldsymbol{i-1}} \cdot \boldsymbol{w_{i-1,j}} + b_{i-1,j}) \tag{2.4}$$

onde $\tilde{\mathbf{a}}_{\boldsymbol{i}} = \{a_{i,1}, ..., a_{i,n_i}\}$ é um vetor de ativação obtido pela concatenação de ativações individuais, $\tilde{\mathbf{a}}_{\mathbf{0}} = \boldsymbol{x}$ é o exemplo de entrada e $\boldsymbol{y} = \tilde{\mathbf{a}}_{\boldsymbol{m}}$ é a saída da rede. Se a camada de saída possuir apenas um *perceptron*, a saída pode ser simplificada como $y = a_{m,1}$.

2.2.3 Funções de ativação

Na literatura de aprendizagem de máquina, são apresentadas diferentes funções de ativação. Não existe, porém, muitos princípios teóricos para guiar a escolha dessas funções. Antes da introdução das *rectified linear units*, ou ReLUs, na literatura eram utilizadas predominantemente funções do tipo sigmoide logística e tangente hiperbólica. Essas funções ainda são muito utilizadas em camadas de saída, por permitirem que o valor de saída represente uma probabilidade no intervalo $[0, 1]$ [19]. Para as demais camadas, hoje em dia ReLUs são a escolha padrão para a maioria das aplicações, em função dos bons resultados obtidos. Mais recentemente, funções do tipo *leaky* ReLU têm se mostrado uma boa alternativa às ReLUs tradicionais. A seguir, as principais funções de ativação são descritas em detalhes.

2.2.3.1 Sigmoide e tangente hiperbólica

Funções do tipo sigmoide são funções cujo gráfico tem formato que se assemelha à letra *s*, como ilustrado na Figura 2.3. A função logística é uma função do tipo sigmoide definida por

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

onde $x \in \mathbb{R}$ e $\sigma(x) \in [0, 1]$. Tal função apresenta uma não linearidade do tipo saturação, e assim, quando o módulo da entrada x é muito grande, variações no seu valor produzem pouca variação na saída.

A função tangente hiperbólica tem comportamento semelhante, podendo ser definida tem termos da função sigmoide logística, fazendo

$$tanh(z) = 2\sigma(2z) - 1 \tag{2.6}$$

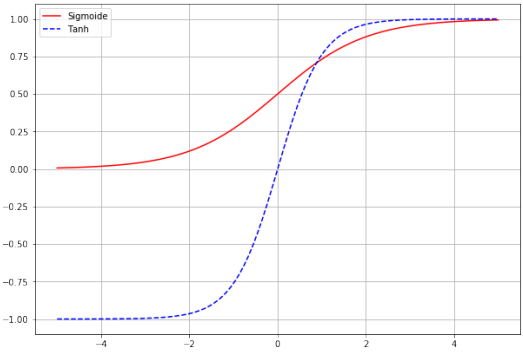
onde $z \in \mathbb{R}$ e $tanh(z) \in [-1, 1]$ [19].

2.2.3.2 ReLU e *leaky* ReLU

As ReLUs (*rectified linear units*) utilizam a função de ativação definida por

$$g(z) = max\{0, z\} = \begin{cases} z, & \text{se } z > 0 \\ 0, & \text{se } z \leq 0 \end{cases} \tag{2.7}$$

Figura 2.3: Comparação entre as funções sigmoide logística e tanh



Fonte: Autor

onde $z \in \mathbb{R}$. A função é linear se a entrada for positiva, mas satura em 0 caso contrário, como ilustrado na Figura 2.4. Em métodos de treinamento baseados no gradiente, a linearidade da função para entradas positivas garante que o gradiente com relação à função de ativação não sofre compressão, facilitando o processo de aprendizagem.

Uma desvantagem da ReLU é que, em métodos de treinamento baseados no gradiente, não é possível atualizar os parâmetros do modelo quando $z < 0$, pois nesses casos a derivada por partes da função g é igual a 0. As *leaky* ReLUs contornam o problema usando a função de ativação definida por

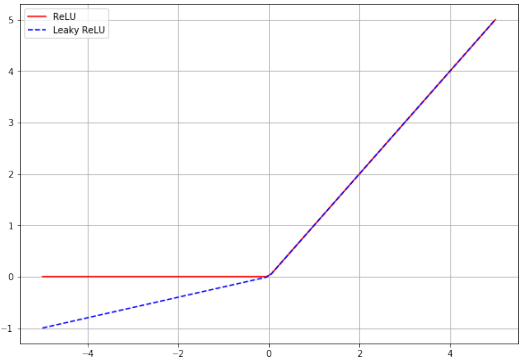
$$g(z) = max\{0, z\} + \alpha_i min\{0, z\} = \begin{cases} z, & \text{se } z > 0 \\ 0, & \text{se } z = 0 \\ \alpha_i z, & \text{se } z < 0 \end{cases} \tag{2.8}$$

onde $z \in \mathbb{R}$ e α_i é uma constante, tipicamente 0.01. Com a adição do termo $\alpha_i min\{0, z\}$, a derivada por partes da função h para $z < 0$ é igual a α_i [19].

2.2.4 Treinamento

De forma geral, algoritmos de aprendizagem de máquina podem ser definidos em termos de um conjunto de dados, uma função custo, um

Figura 2.4: Comparação entre as funções ReLU e Leaky ReLU



Fonte: Autor

procedimento de otimização e um modelo [19]. O conjunto de dados varia conforme a aplicação. O modelo, no caso das redes neurais, é definido pelas diferentes camadas com seus respectivos *perceptrons*. A função custo, também chamada de *loss function*, é uma função que quantifica a proximidade entre um valor inferido pela rede e um valor de referência de acordo com alguma métrica. O procedimento de otimização visa minimizar a função custo, aproximando os valores inferidos às referências.

As não linearidades inerentes às redes neurais fazem com que a maioria das funções custo se tornem não convexas. Por esse motivo, no treinamento de redes neurais não se usam otimizadores com garantia de convergência global, mas sim procedimentos de otimização iterativos e baseados no gradiente (*gradient descent*), que buscam reduzir o custo a um valor baixo. Cada iteração da otimização pode ser dividida em dois passos, chamados de *forward step* e *backward step*.

No *forward step* um exemplo passa pela rede, resultando em uma inferência sobre a qual é aplicada a função custo. No *backward step*, o gradiente da função custo em relação aos parâmetros da rede é calculado utilizando uma técnica chamada *backpropagation*. A direção oposta àquela apontada pelo gradiente indica a direção de mudança dos parâmetros que diminui mais acentuadamente a função custo. Por fim, os parâmetros da rede são atualizados na direção oposta à do gradiente, ou seja,

$$w = w - \alpha \frac{\delta f(\boldsymbol{x})}{\delta w} \tag{2.9}$$

onde w é um coeficiente qualquer da rede, $\frac{\delta f(\boldsymbol{x})}{\delta w}$ é a derivada da função custo, aplicada ao exemplo $\boldsymbol{x}$, em relação ao parâmetro w [28]. O parâmetro α é o tamanho do passo de atualização, chamado de *learning rate*. A *learning rate* não é um parâmetro da rede neural, mas sim do procedimento de otimização. Tais parâmetros são conhecidos como hiperparâmetros [19].

2.2.4.1 Treinamento em *batches*

Para acelerar o processo de treinamento e inferência, é possível combinar diversos exemplos em um *batch*. No *forward step* todos exemplos de um *batch* são processados em paralelo, produzindo um conjunto de saídas sobre as quais são aplicadas o *backward step*. Os gradientes produzidos no *backward step* são então somados e utilizados na atualização dos parâmetros da rede. Essa técnica recebe o nome de *stochastic gradient descent*, pois entre o processamento de dois exemplos de um mesmo *batch* os parâmetros da rede permanecem fixados, sendo atualizados apenas após todo o *batch* ser processado.

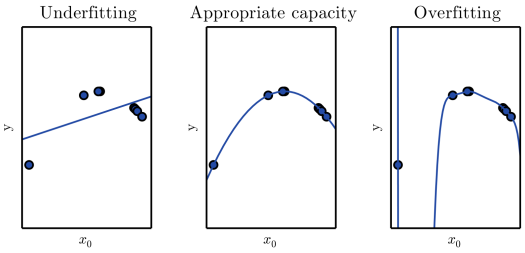
2.2.4.2 Regularização

Independente da *loss function* utilizada, comumente é adicionado à mesma um termo de penalização a valores muito grandes nos parâmetros do modelo, em uma técnica conhecida como regularização. O propósito da regularização é ajustar o equilíbrio entre *underfitting* e *overfitting*. Ao penalizar parâmetros com valor alto, o processo de otimização impede que poucos parâmetros que se encaixam bem a um determinado exemplo se sobressaiam aos demais, pois esses mesmos parâmetros podem não produzir bons resultados nos demais exemplos. O efeito da regularização é controlado por um hiperparâmetro aqui representado por λ.

2.2.4.3 Hiperparâmetros

A escolha de hiperparâmetros afeta diretamente o processo de treinamento e o modelo treinado. Com uma *learning rate* muito baixa, o

Figura 2.5: Efeito da *learning rate* no modelo produzido



Fonte: Goodfellow et al [19]

processo de treinamento pode levar um tempo excessivo até produzir um modelo com boa performance. Uma *learning rate* muito alta pode fazer o treinamento divergir.

O valor de λ controla o equilíbrio entre *underfitting* e *overfitting*. Com um λ muito baixo, a tendência é que ocorra *overfitting*, pois os parâmetros vão tentar se moldar perfeitamente ao conjunto de treinamento, em detrimento a novos exemplos. Já um valor muito alto de λ tende a causar *underfitting*, pois a forte penalização a qualquer variação nos parâmetros impedirá o treinamento do modelo [25].

Na prática, deseja-se encontrar hiperparâmetros que permitam que o processo de treinamento resulte em um modelo satisfatório. Para garantir que o treinamento não tenha qualquer influência do conjunto de teste, o conjunto de treinamento é dividido em dois subconjuntos: o conjunto de treinamento propriamente dito, e um conjunto de validação onde o treinamento pode ser avaliado para diferentes valores de hiperparâmetros. Essa estratégia permite encontrar bons hiperparâmetros e até mesmo adaptá-los durante o treinamento, sem que se introduza tendenciosidade à avaliação do conjunto de teste [19].

2.3 Aprendizagem profunda

Desde o início do século XXI, redes neurais têm se tornado ferramentas cada vez mais poderosas. Isso se deve, em grande medida, ao aumento da capacidade computacional dos computadores modernos e ao acesso a conjuntos de dados com muitos exemplos. Com tais avanços, modelos com cada vez mais camadas e melhor performance puderam ser obtidos

[19]. Essas redes com muitas camadas representam um avanço tão significativo que a sua área de pesquisa recebe uma denominação à parte, chamada de aprendizagem profunda.

2.3.1 Camadas

Redes profundas são organizadas em camadas que podem assumir diferentes formas dependendo da rede e da aplicação. Para a tarefa de detecção de objetos, redes modernas fazem uso de camadas convolucionais, que exploram propriedades espaciais inerentes a imagens, além de camadas de *max pooling*, conexões residuais, *feature pyramids* e *batch normalization*.

2.3.1.1 Camada convolucional

Camadas convolucionais são um tipo de camada desenvolvido especificamente para reconhecer padrões bidimensionais com um alto grau de invariância à translação, ao escalonamento, entre outras distorções [25]. A aplicação natural dessas camadas é na extração de padrões a partir de tipos de dados organizados em duas dimensões, como uma imagem por exemplo.

A operação de convolução é definida no caso unidimensional contínuo como

$$x(t) * w(t) = \int x(a)w(t-a)da \tag{2.10}$$

onde $x(t)$ e $w(t)$ são funções contínuas. A função $x(t)$ comumente representa um sinal de entrada e $w(t)$ a resposta ao impulso de um filtro linear que será aplicado a esse sinal.

Discretizando a equação da convolução, obtém-se

$$x[t] * w[t] = \sum_{n=-\infty}^{\infty} x[n]w[t-n] \tag{2.11}$$

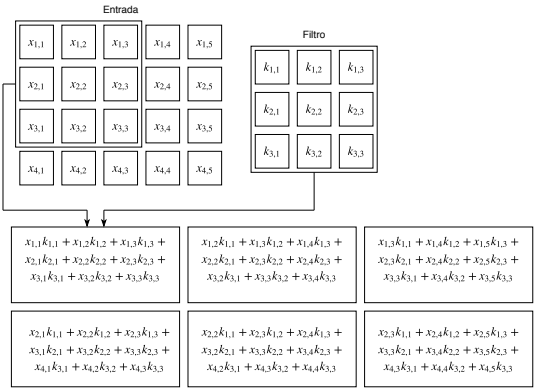
onde $x[t]$ e $w[t]$ são funções discretizadas no tempo.

Em duas dimensões, a convolução discretizada é dada por

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.12}$$

onde I é uma matriz que representa uma imagem digitalizada, por exemplo, e K é uma matriz que representa um filtro, também chamado de *kernel*.

Figura 2.6: Exemplo da operação de convolução em duas dimensões



Fonte: Autor

O diagrama na Figura 2.6 demonstra de forma visual como funciona a convolução discreta em duas dimensões. No contexto de redes profundas, as convoluções geralmente ocorrem entre imagens de dimensões muito maior que os filtros. Nestes casos, uma janela de mesmas dimensões que o filtro é posicionada sobre a imagem e os coeficientes dentro da janela são multiplicados par a par com o filtro e então somados. Essa mesma operação é repetida deslocando-se a janela de uma posição, varrendo toda a imagem e produzindo uma imagem filtrada como saída.

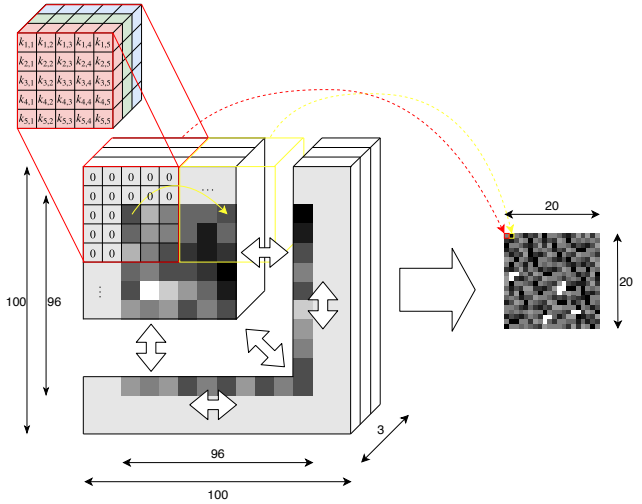
Nota-se que, caso os limites da imagem sejam extrapolados pela janela, o resultado é descartado. Dessa forma, a resolução da imagem filtrada será menor que a da imagem original. Para compensar tal diminuição, é possível utilizar uma técnica conhecida por *zero-padding*, na qual é adicionada uma borda com coeficientes 0 ao redor da imagem, aumentando sua resolução antes da filtragem. É possível ainda deslocar a janela de mais de uma posição sobre a imagem. O número de posições deslocadas é chamado de *stride*. Um valor de *stride* acima de 1 é utilizado quando deseja-se que a imagem filtrada tenha menor resolução.

Na convolução de imagens coloridas, digitalizadas em canais de cor vermelho (R, de *red*), verde (G, de *green*) e azul (B, de *blue*) são utilizados tensores ao invés de matrizes, resultando em

$$S(i,j) = \sum_m \sum_n \sum_{c=\{R,G,B\}} I(m,n,c)K(i-m, j-n, c) \tag{2.13}$$

onde $I \in \mathbb{R}^{w_i \times h_i \times 3}$ é o tensor tridimensional que representa a imagem colorida, de largura w_i e altura h_i, e $K \in \mathbb{R}^{w_k \times h_k \times 3}$ é o filtro utilizado com 3 canais, largura w_k e altura h_k.

Figura 2.7: Diagrama ilustrando as operações realizadas na convolução entre tensores. O tensor do filtro no exemplo possui dimensões $5 \times 5 \times 3$, a imagem de entrada possui dimensões $96 \times 96 \times 3$, o tensor de entrada com *zero padding* possui dimensões $100 \times 100 \times 3$ e a imagem produzida na saída possui dimensões 20×20

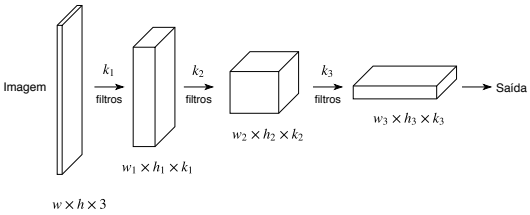


Fonte: Autor

Na Figura 2.7 é apresentado um diagrama da convolução entre uma imagem RGB de resolução 96×96 com um filtro de tamanho $5 \times 5 \times 3$, utilizando *zero-padding* $p = 2$ e *stride* $s = 5$. O resultado é uma imagem filtrada de resolução 20×20. Essa imagem filtrada é comumente chamada de *feature map*.

Cada camada convolucional de uma rede profunda é composta por múltiplos filtros de mesmas dimensões mas com coeficientes diferentes, conforme observa-se na Figura 2.8. A entrada de uma camada é um tensor de dimensões $w_i \times h_i \times K$. Os N filtros da camada têm dimensões $w_k \times h_k \times K$. A convolução de todos os filtros com a imagem resulta em N *feature maps* de dimensões $w_o \times h_o$. A cada elemento dos *feature maps* é aplicada uma função de ativação. Após a ativação, os *feature maps* são concatenados em N camadas de um tensor de saída com dimensões $w_o \times h_o \times N$, que serve de entrada para a camada seguinte.

Figura 2.8: Diagrama ilustrando a progressão das dimensões dos tensores em uma rede convolucional



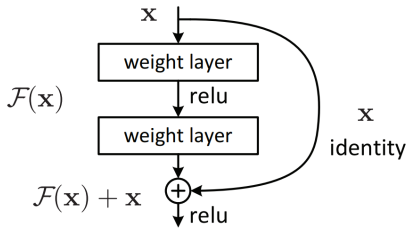
Fonte: Autor

2.3.1.2 *Max pooling*

A camada de *max pooling* é comumente utilizada após uma camada convolucional para sintetizar os *feature maps* produzidos. Isso é feito deslocando-se uma janela sobre o *feature map* em questão e extraíndo apenas o maior valor contido dentro dela. Para um tensor contendo múltiplas camadas, a operação é realizada separadamente em cada uma delas.

A operação resulta em uma representação aproximadamente invariante a pequenas translações na entrada. A propriedade de invariância à translação é interessante quando a prioridade é inferir se a entrada apresenta ou não uma determinada característica, sem dar muita importância à sua localização.

Figura 2.9: Diagrama ilustrando uma conexão residual



Fonte: He et al [29]

2.3.1.3 Conexões residuais

Na tarefa de detecção e reconhecimento de objetos em imagens, observa-se que a performance do sistema tende a aumentar de acordo com a profundidade do modelo. Porém, existe um ponto a partir do qual esse aumento satura e a performance passa a decair acentuadamente conforme se adicionam mais camadas. O uso de conexões residuais visa minimizar esse problema, permitindo o treinamento de redes muito profundas com performance superior às de menor número de camadas.

Uma conexão residual consiste em uma conexão que pula camadas de uma rede, conforme observa-se na Figura 2.9. O tensor do ponto de origem é somado diretamente ao tensor do ponto de destino. Com essa conexão, o trecho da rede passa a ter dois caminhos para o fluxo de informação, um representando uma função não-linear e outro a função identidade. O processo treinamento tem dificuldade em aprender a função identidade a partir de camadas não-lineares. Com a adição de uma conexão residual esse problema é eliminado, pois para aprender a função identidade basta que os pesos das camadas não-lineares sejam zerados [29].

2.3.1.4 *Feature pyramids*

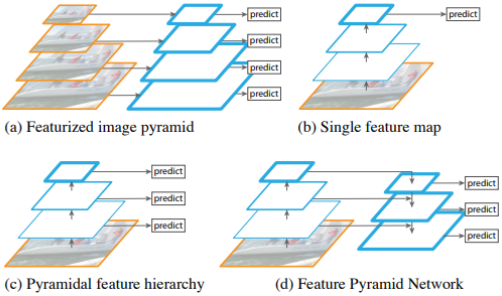
Reconhecer objetos em diferentes escalas é um desafio para sistemas de detecção de objetos. Para contornar o problema, técnicas foram desenvolvidas utilizando o conceito de pirâmide para indicar que uma mesma imagem (ou *feature map*) é representada em diferentes resoluções.

Quando a entrada da rede é representada em diferentes resoluções,

a técnica é chamada de *image pyramid*. A utilização de *image pyramids* fornece uma aproximação à invariância à escala de entrada. Processando em uma mesma rede várias resoluções da mesma imagem, espera-se que em ao menos em uma das escalas a rede será capaz de detectar o objeto.

Uma variação das *image pyramids* define uma rede para cada escala. Essa variação é chamada de *featurized image pyramids* e tem a vantagem de ter redes especificamente treinadas para cada escala. A desvantagem do método é a maior necessidade de espaço de memória e poder de processamento ocasionada pelas múltiplas redes.

Figura 2.10: Os diferentes tipos de pirâmides



Fonte: Lin et al [30]

A técnica de *feature pyramids* explora a natureza piramidal de redes com camadas convolucionais, onde *feature maps* tendem a decrescer em resolução de acordo com a profundidade da rede. Ao invés de definir inteiras redes para cada escala da entrada, a técnica consiste em conectar uma rede piramidal a *feature maps* de diferentes escalas de uma rede convolucional, conforme observa-se na Figura 2.10. As representações na rede convolucional tendem a diminuir em resolução e, na rede piramidal, a aumentar em resolução. Diferentes combinações de *feature maps* com diferentes resoluções são utilizados na tomada de decisão, levando em consideração representações da entrada em diferentes escalas. Dessa forma, a rede apresenta invariância à escala sem grande aumento no custo computacional [30].

2.3.1.5 *Batch normalization*

Redes profundas envolvem a composição de um grande número de funções ou camadas. Durante o treinamento, a atualização de todas as camadas é realizada ao mesmo tempo. A mudança simultânea de diversas funções em uma composição pode levar a resultados inesperados. A camada de *batch normalization* ataca o problema reparametrizando as entradas de cada *batch* de exemplos.

Se $\boldsymbol{H}$ representa um *batch* de tensores que entram em uma camada de *batch normalization*, a saída da camada é dada por

$$\boldsymbol{H'} = \frac{\boldsymbol{H} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \tag{2.14}$$

onde $\boldsymbol{\mu}$ e $\boldsymbol{\sigma}$ são vetores contendo a média e variância associada a cada exemplo e $\boldsymbol{H'}$ é o *batch* de tensores normalizados.

Na prática, a utilização de camadas de *batch normalization* em uma rede profunda facilita o treinamento, permitindo que *learning rates* maiores sejam utilizadas e relaxando os requisitos de inicialização dos parâmetros da rede [31].

2.3.2 Regularização

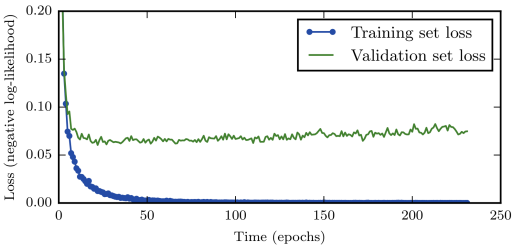
O treinamento de uma rede profunda é um processo demorado e requer um grande conjunto de exemplos para produzir um modelo com boa capacidade de generalização. O desafio de treinar modelos robustos levou ao desenvolvimento de estratégias de regularização para redes profundas. Duas dessas estratégias são *early stopping* e *data augmentation*.

2.3.2.1 *Early Stopping*

Em um treinamento típico de uma rede profunda, analisando a evolução da *loss function* ao longo do tempo nos conjuntos de treinamento (*training loss*) e validação (*validation loss*), observa-se que inicialmente ambas tendem a decair. Porém, a partir de um instante de tempo a *validation loss* passa a crescer, enquanto a *training loss* continua decaindo. Esse fenômeno é evidência gráfica do problema de *overfitting*.

A estratégia de *early stopping* reduz o problema de *overfitting* de uma forma simples e eficaz. Durante o treinamento, a *validation loss* é continuamente monitorada. Quando o seu valor tende a crescer em

Figura 2.11: Exemplo ilustrando como a *validation loss* pode crescer enquanto a *training loss* decai



Fonte: Goodfellow et al [19]

um determinado intervalo, considera-se que ele não voltará a decrescer. Nesse caso o treinamento é parado e o modelo é retornado para o estado com menor *validation loss* [19].

2.3.2.2 *Data Augmentation*

A forma mais fácil de aumentar a capacidade de generalização de um modelo é treiná-lo com mais exemplos. Na prática, porém, a quantidade de dados disponível é limitada. Uma forma de contornar esse problema consiste na criação de exemplos sintéticos a partir de exemplos reais, estratégia chamada de *data augmentation*.

Figura 2.12: Exemplo de operações de *data augmentation* realizadas

Fonte: Autor, adaptado de Almeida et al [32]

A estratégia é particularmente bem sucedida quando aplicada ao

problema de reconhecimento ou detecção de objetos. Isso se deve ao fato dos exemplos usados nesses problemas serem imagens que podem sofrer diversas transformações sem interferir na classe ou localização dos objetos. Operações como a translação das imagens por alguns pixels podem resultar em modelos consideravelmente mais robustos. Outras operações comumente utilizadas são a rotação da imagem de um pequeno ângulo, o recorte de regiões da imagem, a adição de ruído e a introdução de pequenos desvios nas cores da imagem [19].

2.4 Detecção de objetos

Detecção de objetos em imagens é uma tarefa de visão computacional que visa localizar instâncias de objetos de uma determinada classe em uma imagem digitalizada. Um sistema de detecção de objetos de uma classe específica pode ser definido como uma função f que mapeia uma imagem digital I no conjunto de localizações B, dado o modelo W. Assim, tem-se

$$B = f(I, W) \tag{2.15}$$

A localização de um objeto pode ser definida em termos de uma *bounding box*, um retângulo alinhado aos eixos x e y da imagem que descreve o objeto. Uma *bounding box* é definida por um vetor $\boldsymbol{b} \in \mathbb{R}^4$ que indica as coordenadas esquerda, direita, superior e inferior do retângulo, conforme observa-se na Figura 2.13.

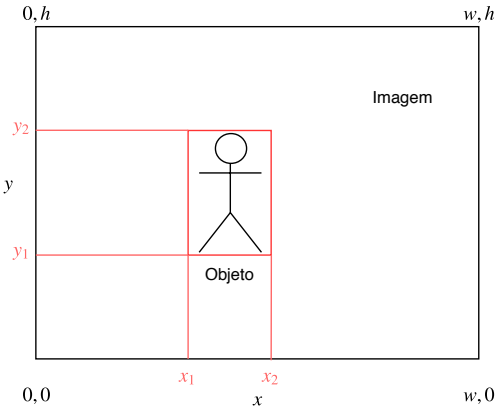
A cada localização detectada pelo sistema, é associado um valor $c \in [0, 1]$ que indica a confiabilidade do sistema de que a *bounding box* descreve corretamente um objeto da classe que se deseja detectar.

2.4.1 Critérios de detecção

Durante o treinamento de um sistema de detecção de objetos, um critério deve ser definido para identificar se uma localização detectada pelo sistema corresponde a um objeto na imagem processada. Se o conjunto de exemplos for anotado com as *bounding boxes* dos objetos contidos em suas imagens, as localizações detectadas são comparadas a cada uma das anotações de acordo com uma métrica de similaridade.

Se uma detecção for comparada às localizações de referência e não

Figura 2.13: Exemplo de uma *bounding box* descrevendo um objeto em uma imagem



Fonte: Autor

obter um valor mínimo de similaridade, considera-se que a localização não corresponde a um objeto da classe detectada, ou seja, ocorreu falsa detecção. Se mais de uma detecção obter uma similaridade acima do valor mínimo em relação a uma mesma referência, considera-se que ocorreram detecções repetidas. E se nenhuma detecção obter similaridade mínima em relação a uma referência específica, diz-se que ocorreu falta na detecção.

2.4.1.1 *Intersection over union* (IoU)

Uma forma de comparar duas localizações é através do método *intersection over union* (IoU), que computa a razão entre interseção e união das áreas de duas *bounding boxes*. A métrica foi popularizada no contexto de visão computacional a partir do desafio Pascal VOC [33].

O valor de IoU é dado por

$$\text{IoU} = \frac{area(\boldsymbol{b_1} \cap \boldsymbol{b_2})}{area(\boldsymbol{b_1} \cup \boldsymbol{b_2})} \tag{2.16}$$

onde $\boldsymbol{b_1}$ e $\boldsymbol{b_2}$ representam duas diferentes *bounding boxes*. Esse valor é então comparado a um valor mínimo IoU_{min} e, se $\text{IoU} \geq \text{IoU}_{min}$, considera-se que as duas *bounding boxes* correspondem ao mesmo ob-

jeto.

2.4.1.2 Non-maximum suppression

O caso de detecções repetidas pode ser aliviado utilizando uma técnica chamada *non-maximum suppression*. Dado um conjunto de localizações $B = \{\boldsymbol{b_1}, ..., \boldsymbol{b_n}\}$ que descrevem uma mesma ocorrência de um objeto, com valores de confiabilidade $c_1, ..., c_n$ associados a cada localização, apenas a *bounding box* b_k com a maior confiabilidade $c_k \geq c_i, \forall i \neq k$ é mantida.

Dessa forma um objeto terá no máximo uma detecção associada a ele e essa detecção será aquela que descreve o objeto com maior confiabilidade.

2.4.2 Avaliação usando *mean average precision*

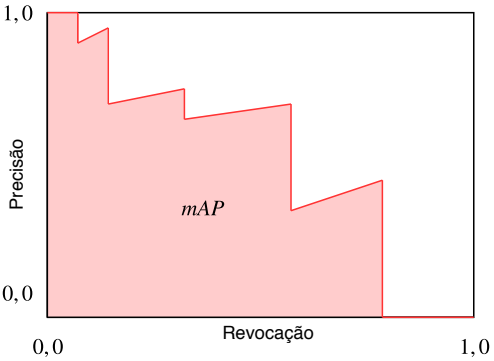
A avaliação de um sistema de detecção de objetos depende de uma métrica que leve em consideração detecções corretas, falsas detecções e faltas na detecção em todo o conjunto de teste. A *mean average precision* (mAP) é uma métrica desse tipo que foi popularizada nesse contexto a partir do desafio Pascal VOC [33]. O seu cálculo é realizado através das medidas de precisão e revocação, obtidas ao aplicar um sistema de detecção em imagens de teste que possuem seus objetos anotados.

Para uma determinada imagem, o sistema considerado pode detectar corretamente um objeto (detecção correta), detectar algo que não é objeto (detecção falsa), ou não detectar algum objeto presente na imagem (falha na detecção). A soma da quantidade de detecções corretas e detecções falsas resulta no número total de detecções. A soma da quantidade de detecções corretas e falhas na detecção resulta no número de objetos presentes na imagem, tanto aqueles que foram ou não detectados.

A partir da quantidade de detecções de cada tipo observadas em uma imagem, a medida de precisão é calculada como a relação entre a quantidade de detecções corretas e o número total de detecções. Já a medida de revocação é calculada como a razão entre detecções corretas e o número de objetos na imagem. O cálculo da precisão não leva em consideração falhas na detecção e o cálculo da revocação não leva em

consideração detecções falsas.

Figura 2.14: Exemplo fictício de curva de precisão x revocação



Fonte: Autor

A métrica mAP combina as medidas complementares de precisão e revocação em um só valor de performance que leva em consideração tanto falhas na detecção quanto detecções falsas.

Ordenando-se todas as detecções por valor decrescente de confiabilidade (falhas na detecção são consideradas detecções de confiabilidade 0), é possível percorrer a lista de detecções assinalando valores crescentes de revocação no intervalo $[0, 1]$ para detecções corretas, assinalando às detecções falsas o mesmo valor de revocação da última detecção correta. Para cada detecção, valores parciais de precisão podem ser calculados dividindo-se a quantidade de detecções corretas vistas até então, pela ordem da detecção. Dessa forma, um par de valores (p, r) representando precisão e revocação, respectivamente, pode ser associado a cada detecção.

Para um conjunto de detecções, os pares (p, r) podem ser visualizados de forma gráfica, conforme ilustrado na Figura 2.14. A curva que conecta os pares é chamada de curva de precisão-revocação. A área abaixo da curva corresponde ao valor de mAP.

CAPÍTULO 3

Desenvolvimento

Este capítulo é dedicado à apresentação das redes e conjuntos de dados utilizados no desenvolvimento deste trabalho. Assim, as informações sobre as redes MMOD e Retinanet são apresentadas, seguidas pelas aplicações consideradas em sua avaliação e os respectivos conjuntos de exemplos.

3.1 Redes avaliadas

Conforme mencionado anteriormente, a detecção de objetos em imagens é uma tarefa com inúmeras aplicações e diversos métodos para realizá-las. Dentre os métodos existentes, redes neurais profundas utilizando camadas convolucionais representam o estado da arte. Porém, encontrar uma configuração de rede que obtenha boa performance para uma determinada aplicação é um processo inexato, usualmente baseado em tentativas e erros.

Até um certo ponto, a adição de mais camadas a uma rede resulta em uma melhor capacidade de generalização. Essa melhoria na generalização, porém, é acompanhada de um aumento na complexidade computacional da rede. Isso tem impacto direto no tempo de execu-

ção do processo de inferência, bem como na duração do processo de treinamento. Assim, em geral é preciso buscar um compromisso entre complexidade e performance.

Visando comparar a diferença na performance de uma rede com menos camadas e uma rede profunda que representa o estado da arte em detecção, serão abordadas, neste trabalho, dois tipos diferentes de redes convolucionais: a rede MMOD, menos profunda e mais especializada, e a Retinanet, muito profunda e com grande capacidade de generalização.

Tabela 3.1: Relação de camadas da rede MMOD

Camada	**Tipo**	**# filtros**	**Tamanho**	***Stride***
0	Entrada			
1	Convolucional	16	5×5	2
2	*Batch normalization*			
3	ReLU			
4	Convolucional	32	5×5	2
5	*Batch normalization*			
6	ReLU			
7	Convolucional	32	5×5	2
8	*Batch normalization*			
9	ReLU			
10	Convolucional	55	5×5	1
11	*Batch normalization*			
12	ReLU			
13	Convolucional	55	5×5	1
14	*Batch normalization*			
15	ReLU			
16	Convolucional	55	5×5	1
17	*Batch normalization*			
18	ReLU			
19	Convolucional	n	9×9	1
20	Saída			

Fonte: Autor

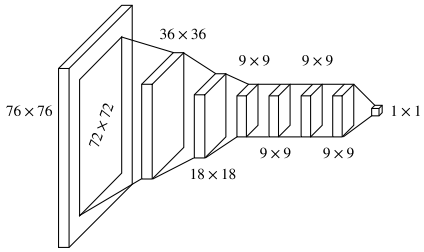
3.1.1 Rede MMOD

Para algumas tarefas de detecção de objetos, redes relativamente simples podem obter performance semelhante ao estado da arte [34] quando treinadas com uma *loss function* conhecida como *max-margin object detection* (MMOD) [35]. A rede treinada com MMOD *loss* tenta maximizar a diferença (margem) entre o número de detecções corretas e o número de detecções incorretas (detecções falsas, falhas na detecção e múltiplas detecções). Essas redes podem ser treinadas do zero com mais facilidade devido ao menor número de camadas, mas teoricamente não têm a mesma capacidade de generalização de redes mais profundas.

A rede utilizada aqui possui 19 camadas, sendo 7 delas convolucionais. Na Tabela 3.1 estão ilustradas as camadas de tal rede, assim como o número, tamanho e *stride* dos filtros das camadas convolucionais. Os filtros de *stride* 2 foram projetados para diminuir a resolução dos tensores pela metade. Como existem três camadas com filtros de *stride* 2, os tensores de entrada têm sua resolução dividida por oito. Nas camadas convolucionais seguintes, a resolução é mantida, exceto na última camada, onde um filtro de tamanho 9×9 é utilizado para gerar um único valor na saída.

Excetuando a camada final, após as camadas convolucionais são inseridas uma camada de *batch normalization* e camada de ativações do tipo ReLU.

Figura 3.1: Diagrama de camadas da rede MMOD



Fonte: Autor

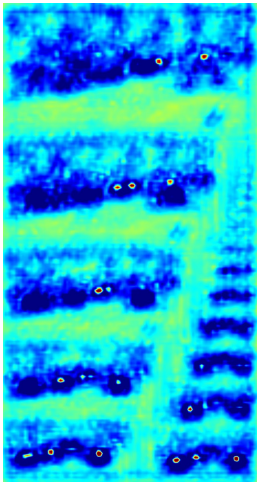
Conforme observa-se na Figura 3.1, para produzir um único valor na saída, a rede deve processar uma região de 76×76 na imagem de entrada. Na prática uma janela de tamanho 76×76 é deslocada sobre toda uma imagem. O *stride*, ou a distância da qual a janela é

deslocada, tem seu valor definido em 8 (valor da redução de resolução da entrada), produzindo uma saída com resolução 8 vezes menor. A saída pode ser interpretada como uma imagem onde cada pixel representa a confiabilidade de detecção.

Em aplicações onde a proporção entre largura e altura dos objetos a serem detectados varia muito entre diferentes imagens, são definidas n *bounding boxes* de referência com proporções fixas. No treinamento, o conjunto de exemplos é separado em n grupos de acordo com a similaridade de proporções, ignorando-se a escala. A última camada convolucional tem seu número de filtros fixado em n, cada um especializado em objetos de determinadas proporções. A especialização surge do processo de treinamento, pois cada exemplo é utilizado para treinar apenas o filtro correspondente com as proporções mais próximas às suas. A quantidade e a proporção das *bounding boxes* de referência são hiperparâmetros do sistema.

Figura 3.2: Relação entre a entrada da rede e os mapas de confiabilidade produzidos. Observa-se que veículos de diferentes escalas são detectados por diferentes níveis na pirâmide

(a) *Image pyramid* de entrada da rede



(b) Mapas de confiabilidade associados a diferentes níveis na pirâmide (vermelho indica maior confiabilidade)

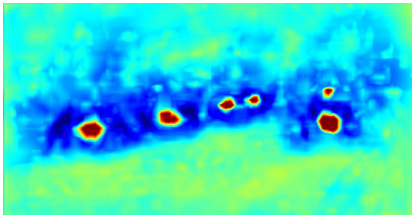
(c) Sobreposição entre mapas de confiabilidade e a pirâmide de entrada

Fonte: King [36]

3.1.1.1 Inferência

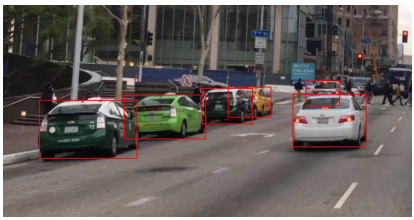
A rede MMOD faz uso de *image pyramids* em sua entrada. Para realizar inferência em uma determinada imagem, versões com diferentes resoluções são geradas e passadas pela rede. Para cada resolução, uma saída de resolução 8 vezes menor é gerada com n canais, cada um representando a confiabilidade de detecção de um objeto de determinadas proporções. Na Figura 3.2, observa-se a *image pyramid* de entrada, as saídas correspondentes e a sobreposição de entrada e saída.

Figura 3.3: Relação entre o mapa de confiabilidade combinado e as detecções na saída



(a) Mapa de confiabilidade combinado (vermelho indica maior confiabilidade)

(b) Sobreposição do mapa de confiabilidade sobre a imagem de entrada



(c) *Bounding boxes* das detecções pós *non-maximum suppression*

Fonte: King [36]

As saídas da rede são combinadas em uma só imagem, cujos valores são comparados a uma confiabilidade mínima a partir da qual se considera que houve detecção. Em pontos onde houver detecção, uma *bounding box* igual à referência correspondente é centrada. O conjunto de *bounding boxes* passa por um processo de *non-maximum suppression*, resultando nas detecções de saída. Na Figura 3.3, observa-se a combinação das saídas de diferentes resoluções, a sobreposição entre essa combinação e a entrada, e as *bounding boxes* obtidas após a aplicação de *non-maximum suppression* [36].

3.1.1.2 Dlib

A rede foi implementada utilizando a ferramenta de aprendizagem de máquina Dlib, que possui código aberto e uma implementação de referência disponível [37].

3.1.2 Retinanet

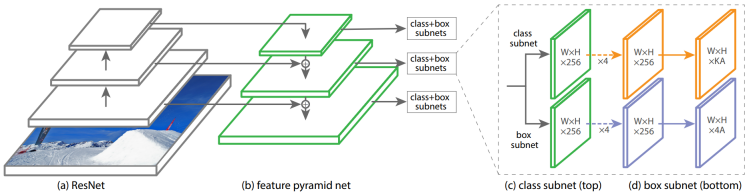
A rede Retinanet representa o estado da arte em detecção de objetos [17]. Tal rede faz uso de diversas técnicas modernas, como conexões residuais e *feature pyramids*, em conjunto com uma *loss function* conhecida como *focal loss*.

A *focal loss* é baseada em uma *loss function* conhecida como *cross entropy*, que tenta maximizar o valor de confiabilidade para detecções corretas e minimizar esse valor para detecções incorretas, supondo que a confiabilidade segue uma distribuição log-normal. A *focal loss* pondera o valor da *cross entropy loss*, priorizando a classificação correta de exemplos difíceis e relevando exemplos muito fáceis.

Devido à profundidade da rede, o seu treinamento é um processo demorado. Porém, a disponibilidade de modelos pré-treinados permite acelerar o treinamento ao otimizar apenas os parâmetros das camadas finais da rede. Essa técnica é conhecida como *fine tuning* e permite obter modelos especializados em uma determinada tarefa sem a necessidade de grande esforço computacional.

A Retinanet tem sua arquitetura baseada na rede residual ResNet, em sua versão com 50 camadas [29]. A partir da ResNet, uma *feature pyramid network* é construída com 5 camadas piramidais de resolução 8, 16, 32, 64 e 128 vezes menor que a entrada, respectivamente. A

Figura 3.4: Arquitetura da rede Retinanet



Fonte: Lin et al [17]

cada uma dessas camadas piramidais, são conectadas ainda duas sub-redes convolucionais: uma para a classificação das detecções e outra para a regressão das *bounding boxes* associadas ao objeto detectado. Ambas são compostas por três camadas convolucionais de tamanho 3×3 seguidas por camadas ReLU, mais uma última camada convolucional de mesmo tamanho.

As sub-redes de classificação têm como saída a confiabilidade de detecção para cada classe. No caso da classificação binária abordado no presente trabalho, um único valor indica a confiabilidade se há ou não detecção. As saídas das sub-redes de regressão de *bounding boxes* são quatro números que refinam a posição e o tamanho de uma bounding box de referência. As saídas das sub-redes associadas a diferentes camadas piramidais são combinadas e *non-maximum suppression* é aplicado.

3.1.2.1 Keras/Tensorflow

A rede foi implementada utilizando a biblioteca de aprendizagem profunda Keras [38], desenvolvida com base no framework Tensorflow [39].

3.2 Aplicações consideradas

A performance de um sistema de detecção depende de fatores que variam de acordo com a aplicação, como a disponibilidade de conjuntos de exemplos bem anotados. Por este motivo, as redes apresentadas serão comparadas em duas aplicações diferentes com seus respectivos conjuntos de dados: detecção de veículos estacionados e detecção de pessoas em ambientes fechados.

3.2.1 Detecção de veículos

Detecção de veículos é uma aplicação já validada com sucesso no contexto da rede MMOD, implementada utilizando Dlib [36]. Em tal implementação, são considerados apenas veículos vistos de um ângulo traseiro presentes em imagens capturadas ao nível do solo. No conjunto de testes utilizado, o modelo atinge precisão de $98,88\%$ e mAP de $47,08\%$.

Neste trabalho, para a aplicação de detecção de veículos foi utilizado um conjunto de exemplos contendo imagens de estacionamentos capturadas de uma vista superior [32]. Batizado de PKLot, o conjunto de exemplos contém imagens de um estacionamento na Universidade Federal do Paraná capturadas a partir do quarto e quinto andares, além de imagens de um estacionamento na Pontifícia Universidade Católica do Paraná capturadas a partir do décimo andar.

O conjunto contém um total de 12417 imagens capturadas sob diferentes condições climáticas. As imagens foram capturadas com resolução 1280×720 e anotadas manualmente. As anotações só foram realizadas em um conjunto fixo de vagas no centro dos estacionamentos, portanto observam-se veículos não anotados presentes nas regiões laterais em diversas imagens. Por esse motivo, o conjunto de exemplos teve 1200 imagens separadas para o treinamento e validação e 300 imagens separadas para o teste. Esses subconjuntos tiveram suas anotações revisadas manualmente para garantir que todos os veículos estejam anotados corretamente.

O conjunto de treinamento e validação foi separado em versões com 1200, 600 e 100 exemplos, permitindo a comparação da performance de modelos treinados com conjuntos de diferentes tamanhos. Um conjunto adicional de 3000 exemplos não revisados foi separado, possibilitando a avaliação do impacto de um conjunto mal anotado na performance do sistema.

3.2.2 Detecção de pessoas

Detecção de pessoas é uma aplicação bastante explorada por sistemas de detecção. Para essa aplicação, o treinamento foi realizado utilizando um conjunto de exemplos contendo imagens de diferentes ambientes no interior de um escritório [40]. O conjunto, conhecido como HDA,

apresenta imagens anotadas de 13 diferentes câmeras, contendo mais de 80 pessoas.

As imagens foram extraídas de sequências de vídeo somando 75207 *frames*. Desses *frames*, um conjunto de 1600 imagens corretamente anotadas foi separado para o treinamento e validação, e outras 360 imagens foram separadas para o teste. O conjunto de treinamento e validação foi separado em versões com 1600, 800 e 100 amostras.

CAPÍTULO 4

Experimentos e Resultados

Este capítulo é dedicado ao detalhamento dos experimentos realizados assim como dos resultados obtidos. Dessa forma, após a introdução dos experimentos e dos parâmetros utilizados, os resultados obtidos são apresentados, seguidos por uma breve análise.

4.1 Experimentos

As redes consideradas representam formas diferentes de atingir o mesmo objetivo: a detecção de objetos para uma determinada aplicação. A rede MMOD é treinada do zero para cada aplicação, sem depender de modelos pré-treinados. A rede Retinanet pode ser treinada do zero, mas como o processo é demorado e depende de um conjunto de dados muito grande é mais vantajoso utilizar um modelo pré-treinado e apenas treinar as camadas finais para a aplicação específica.

Levando em consideração as diferenças entre as redes, os experimentos realizados para as aplicações de detecção de veículos e detecção de pessoas envolveram diferentes estratégias de treinamento.

Tabela 4.1: Experimentos realizados com a rede MMOD

Detecção de veículos	**Detecção de pessoas**
Modelo treinado com 100 exemplos	Modelo treinado com 100 exemplos
Modelo treinado com 600 exemplos	Modelo treinado com 800 exemplos
Modelo treinado com 1200 exemplos	Modelo treinado com 1600 exemplos
Modelo treinado com 3000* exemplos	

Fonte: Autor

4.1.1 Experimentos com a rede MMOD

Para a rede MMOD, foram treinados 7 diferentes modelos do zero utilizando o treinador SGD (*stochastic gradient descent*) da ferramenta Dlib. Os modelos correspondem aos cenários ilustrados na Tabela 4.1, com conjuntos de treinamento de diferentes tamanhos para as aplicações consideradas.

Para a aplicação de detecção de veículos foram treinados modelos com 100, 600 e 1200 exemplos, além de um modelo adicional com 3000 exemplos mal anotados. Foram utilizadas 4 *bounding boxes* de referência, com resoluções 67×70, 70×40, 45×70 e 82×35.

Para a aplicação de detecção de pessoas foram treinados modelos com 100, 800 e 1600 exemplos. Foram consideradas 8 *bounding boxes* de referência, de resoluções 35×90, 48×70, 35×160, 70×58, 70×35, 35×255, 227×35 e 142×35.

O fluxo de treinamento foi o mesmo para todos os modelos. Após a inicialização dos coeficientes com valores aleatórios, a *learning rate* foi inicializada com o valor de $0,1$ e iterações do treinador SGD foram executadas com *batches* 20 amostras sorteadas aleatoriamente do conjunto de treinamento. Quando o método de *early stopping* detectou que 250 iterações do treinador foram executadas sem melhoria no valor de *validation loss*, a *learning rate* foi dividida por 10, até um valor mínimo de $0,0001$.

Após treinar por centenas de milhares de iterações com a *learning*

rate variando entre $0,1$; $0,01$; $0,001$; e $0,0001$, o treinamento foi considerado finalizado e os modelos foram salvos para posterior avaliação.

4.1.2 Experimentos com a rede Retinanet

Para a rede Retinanet, foram considerados 9 diferentes modelos, que correspondem aos cenários ilustrados na Tabela 4.2. Desses modelos, 7 tiveram suas camadas finais treinadas via *fine tuning*, e os outros 2 modelos utilizaram os coeficientes pré-treinados sem refinamento. Para os modelos refinados, o treinador SGD da ferramenta Keras foi utilizado com a *learning rate* fixada em $0,00001$.

Para ambas as aplicações, foram consideradas 9 *bounding boxes* de referência para cada uma das 5 camadas piramidais. A resolução de cada *bounding box* é calculada por $largura = \frac{R}{2} \times prop \times scale$ e $altura = \frac{R}{2} \times \frac{1}{prop} \times scale$, onde $largura \times altura$ é a resolução da *bounding box*, $R \in [8, 16, 32, 64, 128]$ é a resolução da camada piramidal correspondente, $prop \in [\frac{1}{2}, 1, 2]$ é a razão de proporção $\frac{largura}{altura}$ e $scale \in [2^0, 2^{\frac{1}{3}}, 2^{\frac{2}{3}}]$ é um fator de escala.

Aqui, o fluxo de treinamento dos modelos refinados foi mais simples que um treinamento completo. Quando o conjunto de treinamento tem todos os seus exemplos processados uma vez pelo treinador, diz-se que o treinamento completou uma época. No refinamento da rede Retinanet, foram realizadas 20 épocas para cada modelo.

Tabela 4.2: Experimentos realizados com a rede Retinanet

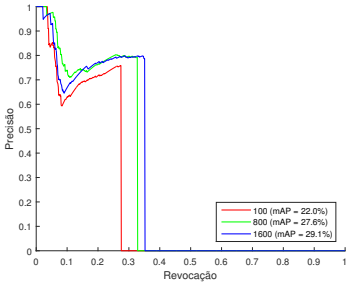
Detecção de veículos	**Detecção de pessoas**
Modelo pré-treinado	Modelo pré-treinado
Modelo refinado com 100 exemplos	Modelo refinado com 100 exemplos
Modelo refinado com 600 exemplos	Modelo refinado com 800 exemplos
Modelo refinado com 1200 exemplos	Modelo refinado com 1600 exemplos
Modelo refinado com 3000* exemplos	

Fonte: Autor

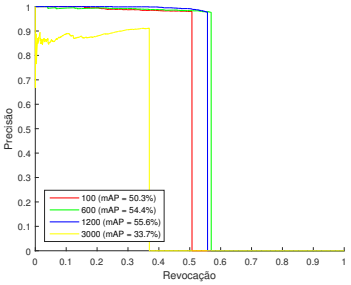
4.2 Resultados

Após o treinamento dos modelos considerados, cada um foi avaliado a partir do conjunto de teste para a respectiva aplicação. Da avaliação se obteve uma lista de valores de precisão e revocação, que foram usados para calcular a medida de mAP, assim como para comparar os modelos de forma gráfica. Valores de mAP próximos a 100% indicam que o modelo detectou uma grande proporção dos objetos (alta revocação) e, dentre todas as detecções, a maior parte consistiu em detecções corretas (alta precisão).

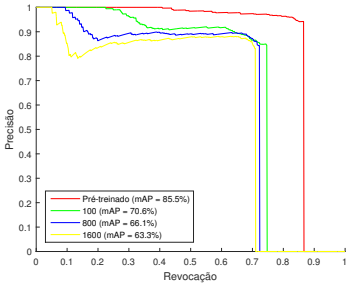
Figura 4.1: Gráficos de precisão/revocação dos modelos obtidos



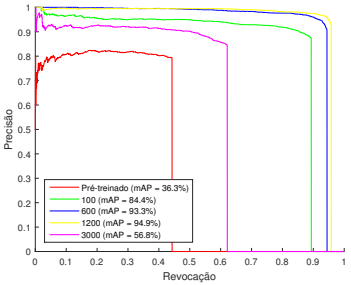
(a) Performance da rede MMOD na detecção de pessoas.



(b) Performance da rede MMOD na detecção de veículos.



(c) Performance da rede Retinanet na detecção de pessoas.



(d) Performance da rede Retinanet na detecção de veículos.

Fonte: Autor

Na Figura 4.1, são apresentados de forma gráfica a relação entre precisão e revocação para cada modelo. Tanto para a aplicação de detecção de veículos quanto para a a detecção de pessoas, observa-se

que os melhores modelos foram obtidos a partir da rede Retinanet.

Figura 4.2: Exemplo de detecção de pessoas utilizando os melhores modelos obtidos para cada rede, aplicados a uma imagem arbitrária do conjunto de exemplos. Em verde são destacadas as detecções realizadas corretamente, e, em vermelho, os objetos que não foram detectados

(a) Detecções obtidas a partir do modelo da rede MMOD treinado com 1600 exemplos.

(b) Detecções obtidas a partir do modelo pré-treinado da rede Retinanet.

Fonte: Autor

4.2.1 Resultados para detecção de pessoas

Na detecção de pessoas, o modelo com melhor performance foi o modelo pré-treinado da rede Retinanet, com mAP de $85,5\%$. Os modelos treinados com a rede MMOD obtiveram valores de mAP abaixo dos 30%. Para a rede Retinanet, os modelos que passaram por *fine tuning* apresentaram performance decrescente conforme aumentou-se o número de exemplos no conjunto de treinamento. Um exemplo da aplicação dos melhores modelos de cada rede em uma imagem do conjunto de teste é ilustrado na Figura 4.2.

A performance consideravelmente pior da rede MMOD pode ser explicada em parte pelo baixo número de camadas, o que implica em

uma menor capacidade de generalização. A detecção de pessoas é dificultada pelo fato de que diferentes pessoas podem diferir em diversos aspectos, como a aparência física, vestimenta, pose, entre outros. O conjunto de dados utilizado, com imagens capturadas no interior de um ambiente de escritório, apresenta casos que dificultam ainda mais a detecção, como pessoas parcialmente ocultas por objetos e grupos de pessoas muito próximos. Portanto, espera-se que redes com maior capacidade de generalização se saiam melhor nessa tarefa.

Um efeito curioso pode ser observado nos modelos da rede Retinanet: quanto maior o conjunto de exemplos utilizado no processo de *fine tuning*, pior é a performance. Isso indica que o modelo pré-treinado foi atrapalhado pelo processo de refinamento. Possivelmente, isso ocorreu pois os objetos do conjunto de teste não são boas representações do conceito de pessoa aprendido pela rede pré-treinada.

4.2.2 Resultados para detecção de veículos

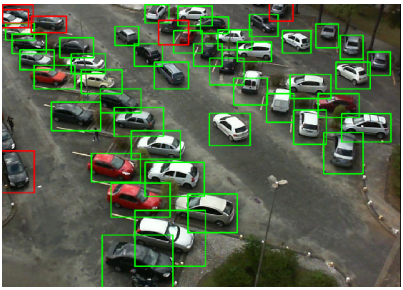
Para a detecção de veículos, o modelo de melhor performance obtido foi aquele obtido através de *fine tuning* a partir do modelo pré-treinado da rede Retinanet, utilizando-se o maior conjunto de treinamento disponível com anotações corretas. Tal modelo obteve um valor de mAP de $94,9\%$, muito acima dos $56,4\%$ obtidos a partir do melhor modelo da rede MMOD. Um exemplo da aplicação de ambos os modelos em uma imagem do conjunto de teste é ilustrado na Figura 4.3.

Conforme ilustrado na Figura 4.1.c, observa-se que, em geral, os modelos da rede MMOD obtiveram taxas de precisão próximas de 100%. Porém, na avaliação de tais modelos ocorreram muitas falhas na detecção. Ou seja, dos objetos presentes nas imagens de teste, muitos não foram detectados, resultando em um valor de mAP menor. Os modelos da rede Retinanet atingiram altas taxas de precisão sem apresentar muitas falhas na detecção.

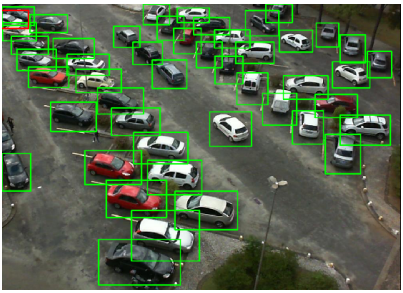
Em ambas as redes, observa-se que o modelo treinado a partir de 3000 exemplos mal anotados resultou em uma performance pior do que os modelos treinados com 100, 600 e 1200 exemplos anotados corretamente. Isso indica que utilizar um conjunto de dados pequeno, mas bem anotado, é mais vantajoso do que simplesmente priorizar o tamanho do conjunto.

O fenômeno observado na aplicação de detecção de pessoas não se

Figura 4.3: Exemplo de detecção de veículos utilizando os melhores modelos obtidos para cada rede, aplicados a uma imagem arbitrária do conjunto de exemplos. Em verde são destacadas as detecções realizadas corretamente, e, em vermelho, os objetos que não foram detectados



(a) Detecções obtidas a partir do modelo da rede MMOD treinado com 1200 exemplos.



(b) Detecções obtidas a partir do modelo da rede Retinanet treinado com 1200 exemplos.

Fonte: Autor

repetiu para a detecção de veículos. Os modelos da rede Retinanet que passaram por *fine tuning* apresentaram performance crescente conforme aumentou-se o tamanho do conjunto de treinamento.

CAPÍTULO 5

Conclusão

Neste trabalho, foram apresentadas diferentes técnicas de aprendizagem profunda aplicadas à detecção de objetos. Em particular, foram abordadas duas redes neurais convolucionais com diferentes níveis de complexidade e capacidade de generalização. A rede MMOD, assim chamada devido à *loss function* utilizada, foi comparada com a rede que representa o estado da arte em detecção de objetos, a Retinanet. As redes foram treinadas e avaliadas nos cenários de detecção de veículos e detecção de pessoas, variando-se o tamanho do conjunto de dados disponível para o treinamento.

Mostrou-se que, em ambas as aplicações, a rede Retinanet obteve performance notavelmente superior. No caso da detecção de pessoas, especificamente, tal rede se saiu melhor quando utilizado um modelo pré-treinado, pronto para o uso sem necessidade de treinamento. Para a detecção de veículos, observou-se que os modelos da rede Retinanet treinados utilizando-se *fine-tuning* obtiveram melhor performance conforme aumentou-se o conjunto de exemplos usado no treinamento, com exceção de conjuntos grandes mal anotados. Neste último caso, mostrou-se que um conjunto de dados bem anotado com 100 exemplos atingiu melhor performance que um conjunto de 3000 exemplos mal

anotados.

Trabalhos futuros

Uma investigação mais profunda poderia ser feita acerca do efeito que o treinamento via *fine tuning* pode ter em uma rede pré-treinada que já apresenta uma boa performance, tal como ocorreu no treinamento da rede Retinanet para a aplicação de detecção de pessoas. Também poderiam ser realizados treinamentos com maiores conjuntos de exemplos mal anotados, visando determinar se existe um número a partir do qual a performance do modelo resultante equivale àquela obtida a partir de um modelo treinado com conjuntos menores e bem anotados. Por fim, seria interessante aplicar os procedimentos de treinamento apresentados neste trabalho em outros conjuntos de exemplos e para diferentes aplicações. Um resumo destes itens é listado abaixo.

(*i*) Investigação do efeito que o procedimento de *fine tuning* tem em redes pré-treinadas que já apresentam boa performance.

(*ii*) Treinamentos utilizando conjuntos com maior números de exemplos mal anotados.

(*iii*) Utilizar os procedimentos descritos neste trabalho para outras aplicações.

Referências bibliográficas

[1] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, 2016.

[2] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, et al. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716*, 2015.

[3] Wanli Ouyang and Xiaogang Wang. Joint deep learning for pedestrian detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2056–2063, 2013.

[4] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[6] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

[7] Yoram Yakimovsky. Boundary and object detection in real world images. *Journal of the ACM (JACM)*, 23(4):599–618, 1976.

[8] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2001.

[9] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[10] Xin Zhang, Yee-Hong Yang, Zhiguang Han, Hui Wang, and Chao Gao. Object class detection: A survey. *ACM Computing Surveys (CSUR)*, 46(1):10, 2013.

[11] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[12] Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *null*, page 1470. IEEE, 2003.

[13] Aude Oliva and Antonio Torralba. Building the gist of a scene: The role of global image features in recognition. *Progress in brain research*, 155:23–36, 2006.

[14] Oren Boiman, Eli Shechtman, and Michal Irani. In defense of nearest-neighbor based image classification. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[15] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic

segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[16] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[17] Tsung-Yi Lin, Priyal Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[18] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

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

[20] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, Berlin, Heidelberg, 2006.

[21] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[22] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In *Advances in neural information processing systems*, pages 2553–2561, 2013.

[23] Chunhui Gu, Joseph J. Lim, Pablo Arbeláez, and Jitendra Malik. Recognition using regions. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1030–1037. IEEE, 2009.

[24] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[25] Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, 2009.

[26] Gaetano Licata. Are neural networks imitations of mind? *JOURNAL OF COMPUTER SCIENCE AND SYSTEMS BIOLOGY*, 8(3) 124-126 (2015)-124):124–126, 2015.

[27] Martin Schumacher, Reinhard Roßner, and Werner Vach. Neural networks and logistic regression: Part i. *Computational Statistics & Data Analysis*, 21(6):661–682, 1996.

[28] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2015.

[30] Tsung-Yi Lin, Piotr Dollár, Ross B Girshick, Kaiming He, Bharath Hariharan, and Serge J Belongie. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016.

[31] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[32] Paulo R.L. de Almeida, Luiz S. Oliveira, Alceu S. Britto Jr., Eunelson J. Silva Jr, and Alessandro L. Koerich. Pklot–a robust dataset for parking lot classification. *Expert Systems with Applications*, 42(11):4937–4949, 2015.

[33] Mark Everingham, Luc Van Gool, Christopher K.I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[34] Davis E. King. Easily create high quality object detectors with deep learning, 2016. Disponível em <`http://blog.dlib.net/2016/10/easily-create-high-quality-object.html`>. Acesso em: 28 nov. 2018.

[35] Davis E. King. Max-margin object detection. *arXiv preprint arXiv:1502.00046*, 2015.

[36] Davis E. King. Vehicle detection with dlib 19.5, 2017. Disponível em <`http://blog.dlib.net/2017/08/vehicle-detection-with-dlib-195_27.html`>. Acesso em: 29 nov. 2018.

[37] Davis E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.

[38] François Chollet et al. Keras. `https://keras.io`, 2015.

[39] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[40] Dario Figueira, Matteo Taiana, Athira Nambiar, Jacinto Nascimento, and Alexandre Bernardino. The hda+ data set for research on fully automated re-identification systems. In *European Conference on Computer Vision*, pages 241–255. Springer, 2014.