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

IMPLEMENTAÇÃO DE REDES CONVOLUCIONAIS PARA A SEGMENTAÇÃO DE IMAGENS EM TEMPO REAL COM VISTAS À APLICAÇÃO EM ROBÔS AUTÔNOMOS COM DISPOSITIVOS DE VISÃO DE BAIXO CUSTO

Carlos Alberto de Sousa Parente Rodrigues

Goiânia - Goiás - Brasil
2018

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: **Carlos Alberto de Sousa Parente Rodrigues**

Título do trabalho: **Implementação de redes convolucionais para a segmentação de imagens em tempo real com vistas à aplicação em robôs autônomos com dispositivos de visão de baixo custo**

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.

Carlos Alberto de S. P. Rodrigues
Assinatura do(a) autor(a)[2]

Ciente e de acordo:

Assinatura do(a) orientador(a)[2] Data: 15 / 08 / 18

[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.

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

IMPLEMENTAÇÃO DE REDES CONVOLUCIONAIS PARA A SEGMENTAÇÃO DE IMAGENS EM TEMPO REAL COM VISTAS À APLICAÇÃO EM ROBÔS AUTÔNOMOS COM DISPOSITIVOS DE VISÃO DE BAIXO CUSTO

Carlos Alberto de Sousa Parente Rodrigues

Dissertação apresentada como requisito para obtenção do Título de Mestre em Engenharia Elétrica e de Computação pelo Programa de Pós-Graduação em Engenharia Elétrica e de Computação da Escola de Engenharia Elétrica, Mecânica e de Computação (EMC), sob a orientação do Prof. Dr. Gélson da Cruz Júnior e a co-orientação do Prof. Dr. Cássio Dener Noronha Vinhal.

Goiânia - Goiás - Brasil
2018

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

Rodrigues, Carlos Alberto de Sousa Parente
Implementação de redes convolucionais para a segmentação de imagens em tempo real com vistas à aplicação em robôs autônomos com dispositivos de visão de baixo custo [manuscrito] / Carlos Alberto de Sousa Parente Rodrigues. - 2018.
110 f.

Orientador: Prof. Dr. Gélson da Cruz Júnior; co-orientador 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, 2018.
Bibliografia. Apêndice.
Inclui abreviaturas, símbolos, gráfico, tabelas, algoritmos, lista de figuras, lista de tabelas.

1. Redes neurais artificiais. 2. Visão computacional. 3. Segmentação de imagens. I. Cruz Júnior, Gélson da, orient. II. Título.

CDU 004

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

Ata de Defesa 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 **Carlos Alberto de Sousa Parente Rodrigues**, realizada em 16 de março de 2018.

Aos dezesseis dias do mês de março de dois mil e dezoito, às 14 horas, na sala Caryocar brasiliensis, bloco "A" da Escola de Engenharia Elétrica, Mecânica 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 Júnior – Orientador (EMC/UFG), Cássio Dener Noronha Vinhal – Co-orientador (EMC/UFG), Fabrízzio Alphonsus Alves de Melo Nunes Soares –UFG e Karina Rocha Gomes da Silva - (EMC/UFG) para julgar a Dissertação de Mestrado de **Carlos Alberto de Sousa Parente Rodrigues**, intitulada **"Implementação de Redes Convolucionais para a Segmentação de Imagens em Tempo Real com Vistas à Aplicação em Robôs Autônomos com Dispositivos de Visão de Baixo Custo"**, 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 Júnior, Presidente da Comissão, abriu a sessão e apresentou o candidato que discorreu sobre seu trabalho, após o que, foi arguido(a) pelos membros da Comissão na seguinte ordem: Cássio Dener Noronha Vinhal, Fabrízzio Alphonsus Alves de Melo Nunes Soares e Karina Rocha Gomes da Silva. 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, em anexo a esta Ata, no prazo máximo de 60 dias, ficando o professor orientador responsável por atestar o cumprimento destas 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, 16 de março de 2018.

Comissão Examinadora designada:

Prof. Dr. Gelson da Cruz Júnior - Orientador (EMC/UFG) (Avaliação: _____)

Prof. Dr. Cássio Dener Noronha Vinhal – Co-orientador (EMC/UFG) (Avaliação: APROVADO)

Prof. Dr. Fabrízzio Alphonsus Alves de Melo Nunes Soares - UFG (Avaliação: Aprovado)

Profa. Dra. Karina Rocha Gomes da Silva – EMC/UFG (Avaliação: APROVADO)

Prof. Dr. Rodrigo Pinto Lemos – (EMC/UFG) (Avaliação: _____)

Prof. Dr. Marco Antônio Assfalk de Oliveira - EMC/UFG (Avaliação: _____)

Candidato:

Carlos Alberto de S. P. Rodrigues

Carlos Alberto de Sousa Parente Rodrigues

"Eu não temo os computadores, temo a ausência deles".

Isaac Asimov

A meus pais, Valdenor e Juliana, e a minha irmã Anna Paula, que com muito carinho e apoio foram de suma importância para que eu chegasse até esta etapa da vida.

RESUMO

Este trabalho apresenta um estudo de redes convolucionais para segmentar e classificar imagens. O objetivo desta rede é futuramente deixar o robô LEIA 1 mais autônomo, utilizando as informações de visão computacional no seu processamento. Métodos como esse são tentativas de adaptação do sistema de processamento de visão dos seres vivos. A complexidade desta tarefa está em não haver entendimento suficiente do sistema biológico para modelar um sistema capaz de processar imagens com a mesma velocidade e eficiência que um ser humano. Para realizar este trabalho, duas diferentes arquiteturas de redes completamente convolucionais foram validadas. A primeira rede possui 13 camadas, enquanto a segunda possui 15 camadas, e mais pesos ajustáveis do que a primeira. Para o treinamento e validação, uma parcela do *dataset Playing for Data* foi utlizado e adaptado. O conjunto de treinamento foi composto de 300 imagens, e a rede foi validada utilizando 2500 padrões. Para cada arquitetura, três rotinas de treinamento foram executadas, com os métodos Adam, Nadam e Adamax. Os resultados mais relevantes utilizaram a arquitetura de 15 camadas com o otimizador Adamax.

IMPLEMENTATION OF CONVOLUTIONAL NETWORKS TO REAL TIME SEGMENTATION AIMING AT APPLICATIONS IN AUTONOMOUS ROBOTS WITH VISION DEVICES OF LOW COST

ABSTRACT

This work presents a study of convolutional networks to segment and classify images. The purpose of this network is to eventually give more autonomy to LEIA 1 robot, using the computer vision information in its processing. Methods such as this attempts to adapt the visual perception system of living beings. The complexity of this task lies in not having sufficient understanding of the biological system to model a system capable of processing images with the same speed and efficiency as a human. To accomplish this work, two different convolutional network architectures were validated. The first network has 13 layers, while the second has 15 layers, and more adjustable weights than the first one. For training and validation, a slice of Playing for Data dataset was used and adapted. The training set was composed of 300 images, and the network was validated using 2500 patterns. For each architecture, three training routines were performed, using the Adam, Nadam and Adamax methods. The most relevant results used the 15-layer architecture with Adamax optimizer.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

SUMÁRIO

Pág.

CAPÍTULO 1

Introdução

Desde os primórdios da humanidade o homem busca inventar ferramentas para otimizar seu trabalho. A valorização da ciência aliada ao desejo do progresso sempre moveram os seres humanos a criar mecanismos, sejam eles hidráulicos, mecânicos, ou de outras naturezas. Um dos principais marcos desse processo foi a invenção da máquina a vapor, que possibilitou a manufatura de produtos em larga escala. Esta criação resultou na Primeira Revolução Industrial, que estimulou pesquisadores a criarem novas tecnologias, como o telégrafo e a fotografia.

Após a Segunda Guerra Mundial, o mundo foi apresentado ao início da computação. Os primeiros computadores eram máquinas limitadas a realizar operações matemáticas e lógicas. Essas máquinas foram criadas para lidar com problemas que demandavam uma quantidade considerável de cálculos, como por exemplo previsões meteorológicas. Atualmente, o computador continua sendo uma ferramenta para cálculos lógico-aritméticos. No entanto, os computadores modernos possuem poder de processamento suficiente para, a partir dessas operações básicas unitárias, realizar tarefas complexas (SOARES, 2012).

Uma dessas tarefas complexas é criar uma máquina que interaja de forma autônoma com o meio para realizar uma tarefa específica. O desejo de criar tais máquinas não é atual, e assim como nos primórdios da inteligência artificial, a falta de entendimento sobre a complexidade de se reproduzir o comportamento de seres biológicos resultava em altas expectativas. Por exemplo, em 1949 o primeiro robô autônomo eletrônico, criado por Grey Walter, tinha a ambição de reproduzir comportamentos complexos a partir de um pequeno número de estruturas fortemente interconectadas (HOLLAND, 1997).

Com o passar do tempo, os pesquisadores passaram a acreditar que com mais poder computacional seria possível alcançar um grau satisfatório de autonomia. Um exemplo dessa fase foi a época de euforia exagerada sobre as redes neurais, que almejavam evoluir a ponto de se comparar com um cérebro humano em um curto período de tempo (RUSSELL; NORVIG, 2009).

Após essa fase romântica, percebeu-se que, além de poder computacional, faltava

entendimento dos sistemas biológicos capazes de possibilitar a reprodução dessa autonomia para uma máquina. Dessa forma, o atual estado da arte lida com algoritmos adaptativos, que simulam de forma matemática certos processos biológicos.

Um desses processos biológicos que os pesquisadores tentam adaptar é o da visão dos seres vivos. Por exemplo, um humano consegue, com aparente facilidade, reconhecer informações e padrões complexos da cena com quase perfeição. No entanto, desenvolver um sistema que interpreta uma imagem com a mesma competência de uma criança de dois anos ainda não é possível (SZELISKI, 2010). Essa dificuldade vem de que a visão é um *problema inverso*, no qual informações insuficientes são utilizadas pra encontrar as incógnitas necessárias para a solução do sistema. Modelos físicos, matemáticos e probabilísticos são utilizados para desenvolver possíveis soluções. No entanto, modelar o mundo visual é algo complexo, e que demanda consideráveis recursos de processamento e memória computacional.

Movido por esse desafio, essa dissertação de mestrado apresenta um estudo de redes neurais convolucionais para classificar e segmentar as imagens de um agente robótico, de forma que seja possível torná-lo mais autônomo. Para uma contextualização desse problema, são apresentados alguns trabalhos relevantes a seguir.

1.1 Revisão Biliográfica

O uso de técnicas de inteligência artificial, mais especificamente de redes neurais artificiais, para classificação e segmentação de cenas é um campo em crescimento, sendo alguns dos trabalhos dessas áreas descritos a seguir.

Em Badrinarayanan e Cipolla (2016), é retratada uma arquitetura de rede neural totalmente convolucional aplicada à segmentação semântica pixel a pixel, nomeada de *SegNet*. Essa arquitetura é composta por uma rede de codificação, uma de decodificação, seguida de uma camada de classificação em pixels. O grande diferencial apresentado no trabalho de Badrinarayanan e Cipolla (2016) está na rede de decodificação, a qual foi composta por uma hierarquia de decodificadores, cada um ligado a um codificador correspondente, similar a um *autoencoder*. Assim, os decodificadores apropriados usam os índices de agrupamento máximo recebidos do seu codificador para realizar a interpolação (*upsampling*) não-linear de seu mapa de recursos de entrada. De acordo com os autores, a principal motivação da *SegNet* foi a necessidade de uma arquitetura eficiente para a correta compreensão de estradas e cenas

internas, levando em consideração tanto a eficiência de memória quanto de esforço computacional requerido. A *SegNet* foi comparada à outras arquiteturas, utilizando uma ampla base de dados, apresentando bons resultados em relação ao tempo de processamento e consumo de memória, demonstrando a viabilidade de seu uso.

Já em He X. Zhang e Sun (2015), motivados pela dificuldade de treinamento de uma rede neural profunda, é apresentada uma estrutura de aprendizagem residual profunda. Nesse artigo, as camadas são reformuladas com base nas funções de aprendizado residual, as quais se referenciam com a camada de entrada. Os resultados apresentados demonstram a efetividade de tal metodologia. Esse artigo obteve o primeiro lugar na ILSVR (*Imagenet Large Scale Visual Recognition Challenge*) 2015 para tarefa de classificação, e também primeiro lugar na *COCO (Commmon Objects in Context) Challenge 2015* nas tarefas de detecção, localização e segmentação.

Os autores Peng X. Zhang e Sun (2017) propuseram uma rede convolucional global (*Global Convolutional Network - GCN*) para problemas de classificação e localização de segmentação semântica. Diferentemente dos algoritmos tradicionais dessa área, os quais concentram-se principalmente nos problemas de localização, esse trabalho teve a proposta de focar também no problema de classificação simultaneamente. Para tanto, uma estrutura de rede do tipo totalmente convolucional foi utilizada como estrutura básica, enquanto que a GCN foi empregada para gerar mapas de pontuação semântica. Os resultados obtidos demonstraram a necessidade de grandes *kernels* para conseguir avaliar a contradição entre localização e classificação, o que explicitou a importância da GCN. Os resultados dos experimentos, considerando o campo receptivo válido e o número de parâmetros, comprovaram a eficiência da metodologia proposta.

No trabalho de Simonyan e Zisserman (2017), foi investigado o efeito da profundidade da rede convolucional em sua precisão quando aplicadas ao reconhecimento de imagem. O foco principal do trabalho foi a avaliação completa de redes de profundidade crescente (até 19 camadas de pesos) utilizando filtros de convolução muito pequenos ($3x3$) em todas as camadas. Os resultados apresentados indicaram uma melhora na precisão de classificação da rede. Tais resultados foram apresentados no *ImageNet Challenge 2014* ganhando o primeiro lugar na tarefa de localização e segundo lugar na de classificação.

Long et al. (2015), implementou uma rede totalmente convolucional a qual foi capaz

de receber uma entrada de tamanho arbitrário e produzir uma saída de tamanho correspondente, considerando uma aprendizagem e inferência eficientes. Tanto o aprendizado quanto a inferência foram realizados em toda a imagem simultaneamente por uma computação densa *feedforward* e retroalimentação. Abordando a dificuldade de balanceamento entre a resolução dos problemas de semântica e localização, o trabalho definiu uma arquitetura que combinou informações semânticas de uma camada profunda e não tão refinada, com informações de aparência de uma camada superficial e mais refinada, produzindo assim segmentações precisas e detalhadas.

Por sua vez, em Shuai Z. Zuo e Wang (2016) uma DAG-RNN (*Directed Acyclic Graph - Recurrent Neural Network*) foi desenvolvida. Tal rede teve como finalidade a captura das ricas dependências contextuais em regiões da imagem, em mapas de recursos conectados localmente. Em outras palavras, redes neurais recorrente (RNN) foram responsáveis por processar imagens estruturadas por grafos acíclicos direcionados (DAG), o que permitiu que a rede modelasse dependências semânticas de longo alcance entre unidades de imagem. Dessa forma, o DAG-RNN foi posicionada no topo do extrator de recursos pré-treinado com a intenção de incorporar o contexto em recursos locais, melhorando assim, sua capacidade representativa.

Por fim, Szegedy W. Liu (2015) elaborou uma rede neural convolucional profunda, chamada *Inception*, para visão computacional. A característica fundamental dessa arquitetura foi a melhor utilização dos recursos computacionais dentro da rede. Para tanto, a profundidade e a largura da rede foram ampliadas cuidadosamente, de forma a manter o custo computacional constante. O potencial da *Inception* foi analisado experimentalmente nos desafios de classificação e detecção da ILSVRC 2014. Os resultados obtidos forneceram evidências da viabilidade de aproximação da estrutura esparsa ideal esperada por blocos de construção densos prontamente disponíveis, melhorando assim, as redes neurais para visão computacional. A principal vantagem deste método foi o ganho de qualidade significativo em um modesto aumento de requisitos computacionais em comparação com arquiteturas mais rasas e mais estreitas.

1.2 Justificativa

A motivação desse trabalho é usar uma rede completamente convolucional para classificar as imagens da câmera do robô LEIA 1, do laboratório LEIA (Laboratório para Educação e Inovação em Automação) da Escola de Engenharia Elétrica, Mecânica

e da Computação, da Universidade Federal de Goiás (UFG).

A tarefa de classificação de uma imagem é dependente do tipo de problema abordado. Por exemplo, existem casos em que a imagem inteira é processada e vinculada a uma classe. Por outro lado, é possível encontrar na literatura trabalhos que rotulam a imagem pixel a pixel, o que acarreta em uma segmentação simultânea.

Uma máquina capaz de identificar e processar o que vê está um passo mais próxima da autonomia plena. Apesar dos esforços e do poder computacional disponível, um robô completamente autônomo é um sonho distante. Um dos principais motivos é a falta de um entendimento considerável de como o cérebro biológico realiza tarefas como processamento sensorial do meio, tomada de decisão de alto nível, reconhecimento de padrões, entre outros. Assim, como forma de contornar a incapacidade de reproduzir fielmente essa autonomia plena, os pesquisadores da área subdividem a tarefa em fragmentos menos complexos, que são mais fáceis de validar individualmente.

Nesse contexto, a atividade de classificar pixel a pixel a entrada visual de um robô é capaz de ajudar nas tarefas de mapeamento, localização, tomada de decisão, entre outros. Ao "entender" o ambiente a sua volta, o robô tem mais informações pra executar melhor sua tarefa, ou mesmo lidar com adversidades. Dessa forma, o estudo dessas técnicas de visão computacional voltadas à rotulação de imagens é relevante para a autonomia plena de uma máquina.

1.3 Definição do Problema

O sistema visual nos mamíferos inicia-se no olho. A cavidade ocular, assim como uma câmera fotográfica, projeta imagens nítidas do mundo sobre a retina. Tais imagens são enviadas, na forma de estímulos, para o córtex, onde será interpretada e lembrada. No córtex, cada um dos diferentes tipos de células será responsável por parte do processamento visual, e.g., células sensíveis à diferença de cor irão tratar da percepção de padrão e forma, enquanto que células "cegas" à cor irão lidar com a detecção de movimento e percepção de profundidade, além das que lidam com cantos e ângulos. Toda essa informação é transmitida para outras áreas cerebrais para que o processamento da imagem realmente ocorra (BEAR B. W. CONNORS; PARADISO, 2002).

Esse processo como um todo depende da iteração de muitos subsistemas específicos.

Ainda é um grande desafio descobrir como tais sistemas integram os pedaços de informação, formando um todo perceptivo coerente. Tal desafio é ainda maior quando se pensa em implementá-los, de maneira viável, em um sistema computacional. Dessa forma, como ainda não é possível a reprodução artificial completa desse sistema, o que se tem são algumas soluções específicas para algum problema, como por exemplo, leitura de texto, reconhecimento visual, entre outros.

Dessa forma, nesse trabalho é abordada a rotulação e segmentação da imagem de entrada. A figura 1.1 ilustra um resultado esperado desse processo. A imagem à direita é a classificação segmentada da figura à esquerda. Vale ressaltar que o exemplo foi ilustrado com o *ground truth* da figura segmentada, isso é, ela exemplifica uma saída com total acurácia. Essas imagens sem erros são comumente utilizadas no treinamento da rede, e no geral são produzidas de forma manual.

Figura 1.1 - Segmentação e classificação simultânea (adaptado de Long et al. (2015)).

No entanto, um sistema computacional heurístico não costuma apresentar resultados tão perfeitos, o que é, no caso de redes neurais, devido a classes ambíguas, proximidade das fronteiras de decisão, carência de padrões durante o treinamento, entre outros. Além disso, algoritmos supervisionados normalmente não lidam muito bem com padrões não treinados, e nem aprendem novos em tempo de execução.

Abordar a rotulação de objetos presentes na cena sem considerar seu contexto é uma metodologia que tende a limitar a acurácia do método. O próprio ser humano

utiliza fortemente o ambiente na sua tomada de decisão. O contexto facilmente muda a decisão do que está sendo visto. Um exemplo está na figura 1.2, em que é fácil para um observador humano estimar objetos nas imagens, mesmo em condições não ideais, e mesmo os objetos visualmente quase idênticos são estimados com certa acurácia. O contexto muda inclusive a estimativa de tamanho do objeto. A figura 1.3 demonstra outro exemplo de decisão, em que um caractere, ou conjunto deles, é interpretado de forma diferente dependendo dos que o acompanham.

Figura 1.2 - Exemplo de decisão baseada no contexto. É possível para um observador humano identificar com certa facilidade vários objetos presentes em (a) e (b). Em particular, os objetos circulados em (c) e (d) possuem diversas características em comum, como formato e tonalidade. Os objetos circulados possuem a aparência similar a (e) após rotacionados (SZELISKI, 2010).

Figura 1.3 - Exemplo de decisão baseada no contexto. O B da imagem (a) é idêntico ao 13 da imagem (b). Os caracteres da vizinhança influenciam na tomada de decisão de um observador humano (SZELISKI, 2010).

Percebe-se a importância do contexto para a correta interpretação de uma cena por um ser humano. Para um sistema computacional, isso não é diferente, no entanto, lidar com o contexto é algo complexo. Primeiramente, como já citado anteriormente, não há conhecimentos concretos acerca do funcionamento do sistema visual biológico. Além disso, um computador é uma máquina de fazer operações lógico-aritméticas, o que significa que, para a máquina, uma imagem não passa de uma sequência de valores distribuídos em uma matriz (figura 1.4). Não há informações adicionais

que correlacionem formas, contornos, entre outros, na estrutura de uma imagem. Cabe aos algoritmos de visão computacional encontrar esses parâmetros baseados em valores de luminescência.

135	112	75	47	33	24	15	23	24	14
131	111	77	51	37	27	11	10	10	12
129	110	78	53	40	30	19	12	11	15
127	106	74	50	37	27	25	21	18	17
123	101	68	43	30	20	15	19	18	13
124	93	67	44	30	29	23	27	25	17
110	86	72	53	31	22	21	27	28	22
84	65	72	39	18	13	19	31	34	29
76	59	45	24	13	20	37	42	43	35
55	42	32	17	13	25	51	54	50	38

Figura 1.4 - Matriz de valores de um pedaço de uma imagem.

Além disso, existem limitações de *hardware* que podem influenciar nos algoritmos utilizados, como por exemplo poder computacional e memória disponível, por parte da CPU, e também resolução, taxa de quadros e se a imagem capturada é colorida ou não, por parte da câmera. A aquisição das imagens em tons de cinza pode ser considerado vantagem e desvantagem. Uma vantagem é que há menos informação a ser processada, e os algoritmos se preocupam apenas em lidar com textura, formas, etc. Para imagens coloridas, uma rede neural, por exemplo, deveria ser treinada com mais padrões, com diferentes cores. Uma desvantagem de lidar com entradas com um canal é que a cor é a característica semântica mais importante de determinadas classes, como por exemplo vegetação.

Uma rede *perceptron* é uma tentativa de utilizar um modelo matemático para simular uma estrutura biológica de forma simplificada. A motivação de se construir tais redes é devido à sua adaptabilidade, visto que seus parâmetros internos se ajustam de acordo com o modelo a qual foi submetida. Dessa forma, a rede aprende a correlacionar os dados de forma a reduzir o erro da classificação. A desvantagem de se utilizar uma *multilayer perceptron* é a quantidade de parâmetros necessários para lidar com imagens de resoluções aceitáveis.

Redes convolucionais (*Convolutional Neural Networks* - CNN) são modelos visuais

atualmente usados para classificação de imagens, detecção de objetos, segmentação de cenas, entre outras tarefas. Comparada a uma rede densa (completamente interconectada), as CNN possuem menos parâmetros, pois utilizam um conjunto de camadas em que seus pesos descrevem filtros de convolução similar aos usados em processamento de imagens. O objetivo dessas camadas de convolução é aprender filtros específicos para classificar os dados, transformando informações da imagem (bordas, formas, texturas, entre outros) em uma representação compacta, com melhor capacidade pra lidar com ruídos da imagem, rotação, translação, entre outros.

Com base no que foi apresentado, é possível notar a relevância de estudos de redes convolucionais aplicadas à visão computacional.

1.4 Objetivos do Trabalho

A seguir são apresentados os objetivos deste trabalho.

1.4.1 Objetivo geral

O presente trabalho tem como objetivo geral a implementação de uma rede neural convolucional para rotular e segmentar imagens de cenas de obtidas por um agente em movimento, como automóveis ou robôs.

1.4.2 Objetivos específicos

Como objetivos específicos, tem-se:

- definição das técnicas a serem utilizadas para o problema de classificação de imagens;
- definição dos conjuntos de treinamento e validação para a rede convolucional;
- implementação da rede completamente convolucional capaz de rotular e segmentar simultaneamente a imagem;
- validação dos resultados obtidos.

1.5 Organização do Trabalho

Este trabalho está dividido em 5 capítulos. No capítulo 1, é exposta uma breve discussão sobre a busca da humanidade por representar processos biológicos no meio computacional. Também é apresentada uma revisão bibliográfica acerca da utilização de redes neurais artificias para a classificação e segmentação de cenas. Por último, a justificativa, definição do problema e objetivos são descritos.

O capítulo 2 descreve a fundamentação teórica a qual foi base do trabalho. Para tanto, os principais conceitos das redes neurais *perceptron*, *autoencoders*, redes convolucionais e visão computacional são discutidos.

Já no capítulo 3 é retratada a metodologia empregada para a implementação de uma rede completamente convolucional, voltada à classificação e segmentação de um conjunto de imagens de cenas exteriores.

No capítulo 4 os resultados obtidos dessa classificação e segmentação são apresentados e analisados. As conclusões relativas ao trabalho são expostas no capítulo 5, e finalizando, o capítulo 6 que contém as referências bibliográficas utilizadas no presente trabalho.

CAPÍTULO 2

Fundamentação Teórica

Nesse capítulo serão abordados os conceitos básicos dos paradigmas simbólico e conexionista da inteligência artificial, assim como os conceitos das redes neurais artificias, principalmente do tipo perceptron. Por fim, serão apresentados os *autoencoders*, as redes convolucionais e as redes *full convolutional*.

2.1 Inteligência Artificial

A ideia de se construir máquinas capazes de imitar a capacidade humana de raciocinar, perceber o mundo e identificar os objetos ao nosso redor, e até mesmo de falar e compreender nossa linguagem não é nova. No entanto, somente com o desenvolvimento dos computadores modernos, após a Segunda Guerra Mundial, a implementação de tais ideias se tornou possível.

Segundo Russell e Norvig (2009), o nascimento oficial da Inteligência Artificial (IA) pode ser associado à conferência de verão em Dartmouth College, Estados Unidos, no ano de 1956. Tal conferência foi composta dos pesquisadores: John McCarthy (Dartmouth), Marvin Minsky (Hardward), Nathaniel Rochester (IBM) e Claude Shannon (Bell Laboratories), com o intuito de realizar "um estudo durante dois meses, por dez homens, sobre o tópico inteligência artificial". Essa conferência não chegou a introduzir novos avanços, no entanto, teve papel fundamental ao aproximar os grandes pesquisadores da área desse momento.

Após esse nascimento veio a fase do entusiasmo inicial e das grandes expectativas, também conhecida como fase clássica (1956-1970), que tinha como grande objetivo simular a inteligência humana, usando para isso métodos solucionadores gerais de problemas e lógica. Tal fase, no entanto, não obteve bons resultados ao subestimar a complexidade computacional dos problemas a serem resolvidos.

De 1970 a 1980, veio a fase romântica visando simular a inteligência humana em situações pré-determinadas. Os métodos utilizados eram ligados ao formalismo de representação de conhecimento adaptados ao tipo de problema, além de mecanismos de ligação procedural visando maior eficiência computacional. Porém, encontraram grandes problemas ao subestimarem a quantidade de conhecimento necessária para tratar problemas reais.

Já na fase moderna, de 1980-1990, possou-se a utilizar sistemas de regras, representação da incerteza e o conexionismo para simular o comportamento de um especialista humano ao resolver problemas em domínios específicos. Todavia, novamente, esta fase não foi bem sucedida, uma vez que subestimou a complexidade da aquisição de conhecimento.

É usual encontrar na literatura de IA a descrição de dois paradigmas distintos voltados a descrever os sistemas e processos inteligentes: o simbólico e o conexionista. A linha simbólica segue a tradicão lógica e foi fortemente defendida pelos pesquisadores McCarthy e Newell. No artigo *Physical Symbol System* (NEWELL, 1980), os princípios dessa linha são apresentados.

A linha conexionista, por sua vez, têm como principal característica a modelagem da inteligência humana por meio da simulação dos componentes do cérebro humano, ou seja, da simulação dos seus neurônios e suas interligacões. Um ponto fundamental para essa linha ocorreu em 1943, quando o neuropsicólogo McCulloch e o lógico Pitts apresentaram um primeiro modelo matemático para o neurônio. A partir deste, um modelo inicial de interligação dos neurônios, ou seja, de uma rede neural. Esta rede foi chamada de Percetron e foi proposta por Rosenblatt. Com o surgimento e popularização dos microprocessadores essa área tornou-se cada vez mais ativa, dando origem a área de redes neurais artificiais.

2.2 Rede Neural Artificial

As redes neurais artificias (RNA) podem ser entendidas como uma analogia à forma que o cérebro humano processa informações, aplicadas ao meio computacional. Diferentemente de um computador digital convencional, o cérebro humano é capaz de realizar certos processamentos, por exemplo o reconhecimento de padrões, de uma forma mais otimizada que o computador, devido à sua estrutura neural. Aproveitando-se dessa característica, as RNAs, procuram simular a maneira que o cérebro humano realiza uma tarefa, conseguindo assim aprender através de exemplos e generalizar esse conhecimento adquirido.

Como já mencionado anteriormente, o início da RNA está ligado ao trabalho de McCulloch e Pitts, o qual apresentou um modelo matemático de um neurônio biológico artificial chamado de *Psychon*, além de uma máquina inspirada no cérebro humamo. Apesar da grande contribuição de tal modelo, um grande problema do mesmo era a

incapacidade de realizar a tarefa de aprendizado.

Marvin Minsky, no ano de 1951, desenvolveu o primeiro neurocomputador (Snark). O Snark era capaz de, a partir de um ponto de partida, ajustar seus pesos de forma automática, servindo como um modelo inicial para futuras estruturas. Como continuidade desses esforços, em 1958, Frank Rosenblatt e Charles Wightman, juntamente com outros estudiosos, chegaram a um neurocomputador bem sucedido, sendo conhecidos como os fundadores da neurocomputação. Seus estudos possibilitaram os modelos do tipo *perceptron*.

Apesar do sucesso inicial, os modelos baseados na *perceptron* apresentavam graves problemas, os quais foram apresentados na trabalho de "Perceptrons: an introduction to computational geometry" (MINSKY; PAPERT, 1969). Os autores provaram matematicamente que tais modelos, na forma como foram descritos, não eram capazes de aprender funções linearmente não-separáveis, como por exemplo a função lógica XOR (ou exclusivo).

Já nos anos 80 em diante os trabalhos voltados as RNA retomaram o crescimento, graças ao desenvolvimento de computadores mais potentes e ao surgimento de novos modelos de RNA. Em 1982, John Hopfield, apresentou uma rede com conexões recorrentes, onde o sinal não se propagava exclusivamente para frente, além de um aprendizado não supervisionado com a competição entre os neurônios. Em 1986, teve-se o reaparecimento das redes *perceptron*, mas agora incorporando a teoria de redes em multinível treinadas com aprendizado por retropropagação (*backpropagation*), desenvolvida por Rumelhart, Hilton e Willians. Este algoritmo de treinamento deu início a um maior número de pesquisas em neurocomputação.

Devido à natureza neurobiológica das RNAs, a seguir será apresentada brevemente a estrutura e funcionamento básico do cérebro humano.

2.2.1 O Cérebro Humano

De acordo com Haykin (2001), o sistema nervoso de um ser humano pode ser representado, de forma simplificada, como um esquema de três estágios, como apresentado na Figura 2.1.

A Rede Neural, que aparece no centro do sistema, representa o cérebro desse sistema, e recebe e envia informações de forma contínua. Os receptores recebem os estímulos

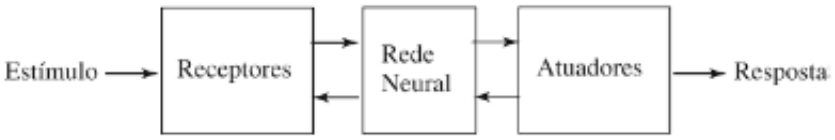

Figura 2.1 - Diagrama dos sistemas nervoso humano. (HAYKIN, 2001).

do corpo humano ou do ambiente externo e convertem tais estímulos em impulsos elétricos transmitindo assim a informação à rede neural (cérebro). Os atuadores, por sua vez, convertem os impulsos elétricos oriundos do cérebro em respostas, ou saídas dos sistema. É importante observar a presença de setas apontando para a direita (transmissão para frente) e setas para esquerda (realimentação do sistema).

Graças ao trabalho de Ramón e Cajál, em 1911, os neurônios foram percebidos como principais estruturas do cérebro humano. Conectados uns aos outros através de sinapses, eles formam uma grande rede neural, capaz de processar e armazenar um grande número de informações. Ainda de acordo com Haykin (2001), o córtex humano apresenta cerca de 10 bilhões de neurônios com 60 trilhões de conexões. Tais conexões, ou sinapses, são responsáveis pela interação entre os mesmos, excitando-os ou inibindo-os.

Observando a Figura 2.2 é possível identificar a estrutura simplificada de um neurônio. O mesmo é constituído de:

- Dendritos: responsáveis por receber os sinais de entrada;
- Corpo: também conhecido como *somma*, é responsável por coletar e combinar as informações oriundas dos demais neurônios;
- Axônios: representam as vias de transmissão do sinal de uma extremidade à outra do neurônio;
- Terminais sinápticos: responsáveis por transmitir os sinais processados aos próximos neurônios.

De uma forma simplificada, os dentritos captam os sinais recebidos em um determinado período de tempo e os envia ao corpo do neurônio, onde são processados. Quando os estímulos recebidos atingem determinado limite, o corpo da célula envia

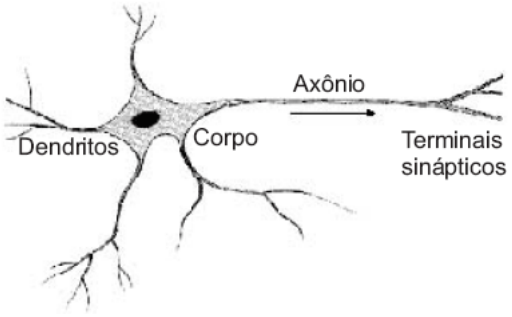

Figura 2.2 - Representação simplificada de um neurônio humano (FERNEDA, 2006).

um novo impulso que se propaga pelo axônio e é transmitido às células vizinhas pelos terminais sinápticos.

É importante ressaltar que apesar dos esforços, os neurônios artificiais que compõem um RNA são bem primitivos quando comparados às estruturas presentes no cérebro humano. Como o foco dessa dissertação é computacional, o estudo da representação computacional dos neurônios se faz necessária.

2.2.2 Modelo de um Neurônio Artificial

Assim como no sistema nervoso humano, o neurônio é a unidade de processamento fundamental de uma rede neural artificial. O neurônio artificial, da mesma forma que o neurônio humano, é composto de estruturas fundamentais (Figura 2.3). Tais estruturas são:

- Sinapses: também conhecidos como elos de conexão, são sinais de entrada com pesos ou força própria. Uma sinapse x_m, conectada a um neurônio k, é multiplicada por um peso sináptico w_{kj}. Esse peso possui a função de indicar a importância de determinado sinal. Diferentemente dos neurônios naturais, os artificias podem apresentar valores de pesos negativos (sinapses inibitórias) ou positivos (sinapses excitatórias);
- Somador: atuando como um combinador linear, esta estrutura soma os sinais de entrada, ponderando-os pelos valores das sinapses do neurônio;
- Função de ativação: tem como principal função determinar um intervalo permissível de amplitude do sinal de saída de um neurônio. Normalmente esse intervalo assume os valores $[0, 1]$ ou $[-1, 1]$;

- *Bias*: serve como um valor constante e adaptável, influenciando diretamente na saída real da função de ativação.

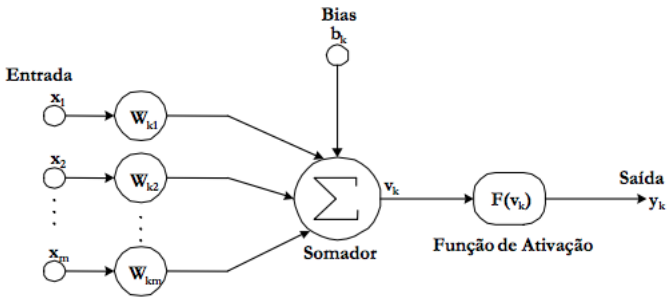

Figura 2.3 - Representação simplificada de um neurônio artificial. (HAYKIN, 2001).

Em notação matemática um neurônio k pode ser descrito como:

$$y_k = \varphi(u_k + b_k) \tag{2.1}$$

Onde:

$$u_k = \sum_{j=1}^{m} w_{ij} x_j \tag{2.2}$$

Sendo que $x_1, x_2, ..., x_n$ representam os sinais de entrada; $w_{k1}, w_{k2}, ..., w_{kn}$ correspondem aos pesos sinápticos do neurônio k; O valor u_k é a saída do somador de acordo com os sinais de entrada; b_k é o bias; $\varphi(\cdot)$ é a função de ativação e y_k é o sinal de saída do neurônio.

2.2.3 Função de Ativação

De acordo com Haykin (2001) existem três tipos básicos de função de ativação: função de limiar; função linear por partes e função sigmóide. Tais funções têm como objetivo determinar a saída de um neurônio de acordo com o campo local induzido.

a) Função de limiar: comumente referida como função de Heaviside, representa

a saída do neurônio k expressa matematicamente como:

$$y_k = \begin{cases} 1 & \text{se} \quad v_k \geq 0 \\ 0 & \text{se} \quad v_k < 0 \end{cases} \tag{2.3}$$

Onde o campo local induzido do neurônio (v_k) é:

$$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k \tag{2.4}$$

b) Função linear por partes: atua como uma aproximação de um amplificador não-linear. Quando a região linear de operação não entra em saturação, surge um combinador linear. Porém, se o fator de amplificação da região linear assume valores muito grandes, a função se reduz à função limiar. Matematicamente temos:

$$\varphi(v) = \begin{cases} 1 & \text{se} \quad v_k \geq +\frac{1}{2} \\ v & \text{se} \quad +\frac{1}{2} > v > -\frac{1}{2} \\ 0 & \text{se} \quad v_k \leq -\frac{1}{2} \end{cases} \tag{2.5}$$

c) Função sigmóide: pode ser definida como uma função semilinear, limitada e monótona. Estritamente crescente é capaz de apresentar um balanceamento adequado entre o comportamento linear e o não-linear. Um exemplo para essa função é a função logística, definida como:

$$\varphi(v) = \frac{1}{1 + \exp(-av)} \tag{2.6}$$

Sendo que a representa a inclinação da função sigmóide. Em alguns casos é necessário que a função abranja de -1 a +1, assumindo assim uma forma anti-simétrica em relação à origem. Para tanto, pode-se utilizar a função tangente hiperbólica $\varphi(v) = tanh(v)$.

Além destas funções básicas, também podem ser utilizadas funções como:

a) Funcão tangente hiperbólica ($tanh$): é similar à função logística, mas possui imagem entre -1 e 1:

$$\varphi(v) = tanh(v) = \frac{sinh(v)}{cosh(v)} \tag{2.7}$$

Esta função costuma apresentar melhores resultados de treinamento quando comparados à função logística. Isso ocorre devido ao fato de que valores muito negativos tendem a zero. Uma vez que o algoritmo *backpropagation* utiliza os gradientes da função de ativação para o ajuste de pesos, isso pode deixar a convergência dos parâmetros mais lenta.

b) Função linear retificada (ReLU): alguns trabalhos recentes utilizam essa função na camada oculta. Sua saída é a sua própria entrada para valores positivos, e zero para valores negativos.

$$\varphi(v) = max(0, v) \tag{2.8}$$

É uma das funções não-lineares mais simples de ser implementada. Possui vantagens de treinamento em relação às sigmóides, já que seu gradiente não tende a valores muito pequenos para uma parte do domínio da função, o que resulta em um treinamento mais rápido. No entanto, sua principal desvantagem é a de permitir que neurônios "morram" durante o treinamento. Isso acontece quando suas ativações convergem cada vez mais para valores negativos, e pela característica da função, esses pesos não são mais atualizados.

c) Função *softmax*: É similar à função logística, mas diferente dela, possui uma saída normalizada. Enquanto a ativação logística retorna a saída real da função, a softmax divide cada saída de modo que a soma de todas elas seja 1. Desta forma, esta saída pode ser interpretada como a probabilidade classe a classe da pertinência do padrão de entrada.

2.2.4 Arquiteturas de uma RNA

A estrutura (arquitetura) de uma rede neural está diretamente vinculada ao algoritmo de aprendizado a ser utilizado para o treinamento da mesma. Dessa forma, conhecer as principais arquiteturas de uma RNA é importante. É possível destacar três principais arquiteturas:

a) **Redes diretas de uma camada**

É um tipo de rede acíclica, ou seja, não possui laços de realimentação. Seus neurônios estão organizados em uma única camada e recebem informações de forma simultânea. A figura 2.4 apresenta essa arquitetura, onde a camada de entrada de neurônios recebe sinais de excitação do meio externo e se conecta à camada de saída de neurônios, que por sua vez, conecta-se aos nós de saída da rede.

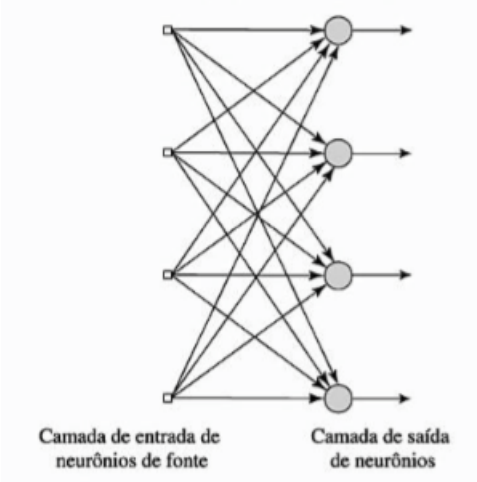

Figura 2.4 - Rede direta de uma camada. (HAYKIN, 2001).

b) **Redes diretas com múltiplas camadas**

Assim como as redes diretas de uma camada, esta arquitetura apresenta um fluxo de informação em uma única direção, no entanto, possui uma ou mais camadas ocultas (neurônios ocultos). A grande vantagem desses neurônios ocultos está na capacidade de extração de características complexas, adquirindo assim uma perspectiva global, apesar de sua conectividade local (HAYKIN, 2001).

Nas redes de múltiplas camadas, Figura 2.5, cada uma das camadas são destinadas a uma função específica. Os nós de fonte presentes na camada de entrada fornecem o padrão de ativação e atuam como sinais de entrada dos neurônios da segunda camada. Os sinais de saída da segunda camada

servem então como entrada para a terceira camada e assim sucessivamente por todas as camadas, até atingirem a camada de neurônios de sáida, a qual apresentará a resposta da rede em relação ao padrão de ativação oriundo da camada de entrada. As camadas intermediárias são fundamentais para a extração de características apresentadas nos padrões de entrada.

Essa rede pode ainda ser totalmente conectada, quando cada um dos nós de uma camada está conectado a todos os nós da camada seguinte, ou parcialmente conectada. Na prática, não é comum interligar parcialmente duas camadas de neurônios, pois isso de certa forma limita a capacidade de aprendizado da rede. Além disso, caso uma sinapse seja realmente irrelevante, o próprio algoritmo de treinamento tenderá seu valor a zero.

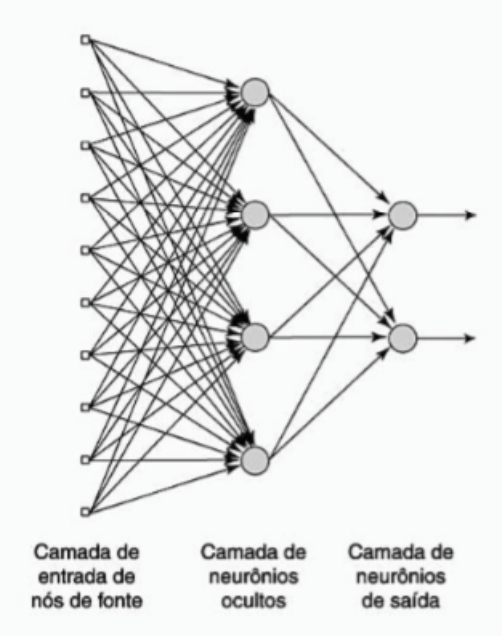

Figura 2.5 - Rede direta com múltiplas camadas. (HAYKIN, 2001).

c) Redes recorrentes

Diferente das redes apresentadas anteriormente, as redes recorrentes apresentam ao menos um laço de realimentação (Figura 2.7). Esse tipo de rede pode apresentar ou não neurônios ocultos (Figura 2.6). Devido a essa realimentação, a rede pode ser capaz de melhorar seu aprendizado, impactando

assim no seu desempenho. Além disso, devido a presença dos elementos de atraso unitário (z^{-1}) a rede incorpora um comportamento dinâmico não-linear.

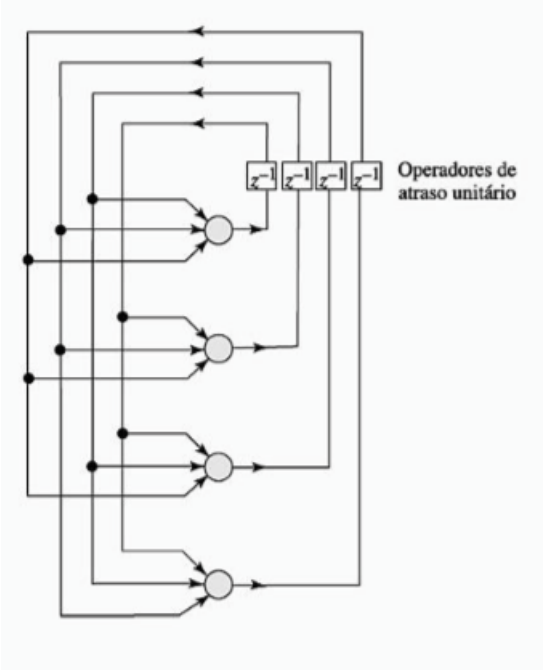

Figura 2.6 - Rede recorrente sem neurônios ocultos. (HAYKIN, 2001).

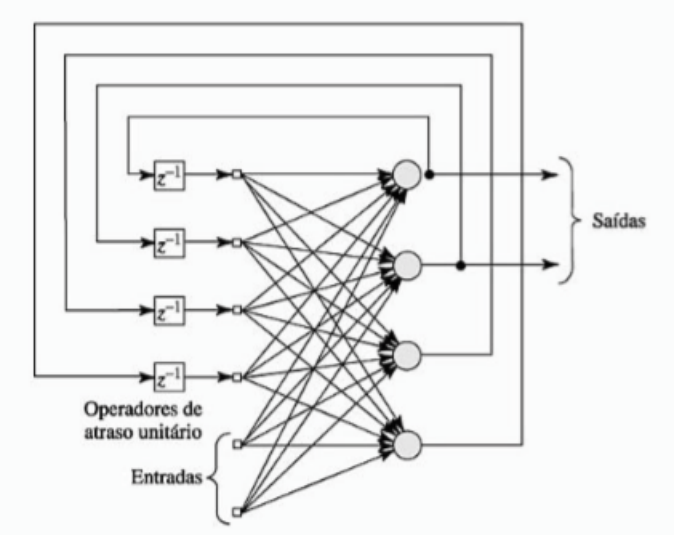

Figura 2.7 - Rede recorrente com neurônios ocultos. (HAYKIN, 2001).

2.2.5 Aprendizagem em RNA

A capacidade de aprender de acordo com o seu ambiente melhorando gradativamente o seu desempenho é um dos principais fatores que caracterizam uma rede neural. O foco dessa aprendizagem nada mais é que determinar a intensidade das conexões entre os neurônios. O conjunto de regras para adaptação dos parâmetros de uma RNA, fazendo com que a mesma aprenda determinada função, é chamado de algoritmo de aprendizagem. Não existe um único algoritmo de aprendizagem, mas sim uma grande variedade de algoritmos com características e vantagens específicas. De forma suscinta, os algoritmos distinguim-se em relação ao ajuste de peso sináptico dos neurônios e como a mesma se relaciona com o seu ambiente (HAYKIN, 2001).

Podemos classificar a aprendizagem em três grandes paradigmas: Aprendizagem supervisionada, por reforço e não-supervisionada. A escolha do tipo de aprendizagem depende do problema a ser tratado pela rede.

2.2.5.1 Aprendizagem supervisionada

Como o próprio nome diz esse tipo de aprendizagem conta com uma supervisão: um "professor", que possui conhecimento a respeito do ambiente do problema a ser solucionado. A figura 2.8 apresenta esquematicamente o processo de aprendizagem supervisionada. Nesse processo, o professor, a partir de seu conhecimento a respeito do ambiente, é capaz de fornecer à rede padrões de resposta desejados para o vetor de entrada. A diferença entre o resultado esperado e o apresentado pela rede é conhecido como sinal de erro. A cada iteração os parâmetros da rede são ajustados de acordo com o vetor de treinamento e o sinal de erro. Dessa forma, a rede é treinada com base no conhecimento desse "professor". Ao final do treinamento, quando esse erro é minimizado, a rede encontra-se treinada, não precisando mais da figura do "professor", esperando-se assim que ela lide com o ambiente por si mesma. Tal forma de aprendizagem é dita aprendizagem por correção de erro. Um exemplo de algoritmo para treinamento com aprendizado supervisionado é o *backpropagation*.

2.2.5.2 Aprendizagem por reforço

Quando não é possível contar com o padrão e resposta desejados, é possível utilizar a interação com o ambiente no processo de aprendizagem, contando com resultados da sua própria experiência no processo de aprendizado. Empregando uma analogia com os seres humanos, é o processo de recompensa de esforço de uma criança que, por

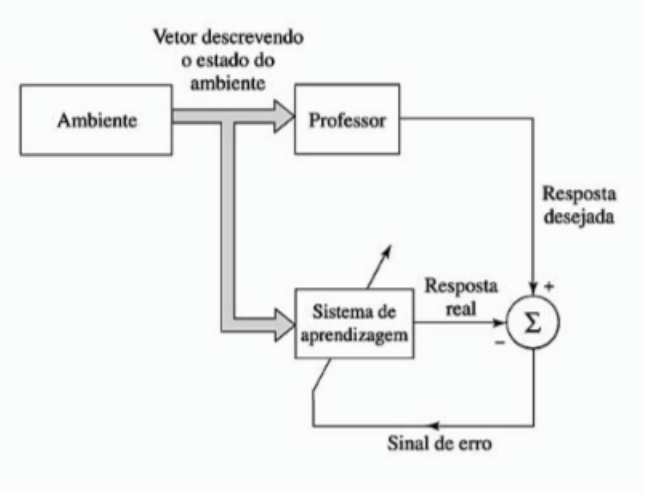

Figura 2.8 - Aprendizagem supervisionada. (HAYKIN, 2001).

exemplo, ao realizar uma ação receba um *feedback*. Ao receber um elogio, a criança tenderá a repetir a ação, enquanto que ao sofrer uma crítica, irá evitar tal ação no futuro.

A Figura 2.9 apresenta o esquema desse processo de aprendizagem. O processo de aprendizado é desenvolvido em torno da figura de um *crítico*. Este *crítico* recebe o sinal reforço primário proveniente do ambiente e melhora a qualidade desse sinal, criando assim, o refoço heurístico. Se essa ação provocar uma piora na qualidade do resultado apresentado pela rede, o reforço será negativo e tal ação tenderá a ser desencorajada no futuro. Caso contrário, têm-se um reforço positivo e o encorajamento da repetição de tal ação. Desta forma o algoritmo converge para um padrão de comportamento esperado.

2.2.5.3 Aprendizagem não-supervisionada

Nesse tipo de aprendizagem além de não se ter a figura do "professor" externo, também não se tem o *crítico* para supervisionar o processo de aprendizagem. Estas redes utilizam então a busca de características ou de padrões significativos nos dados de entrada.

Através dos valores de entrada fornecidos pelo ambiente a rede desenvolve uma correlação entre tais valores, isso é, uma representação interna das características dos mesmos (figura 2.10). Para a implementação desse tipo de rede é possível utilizar,

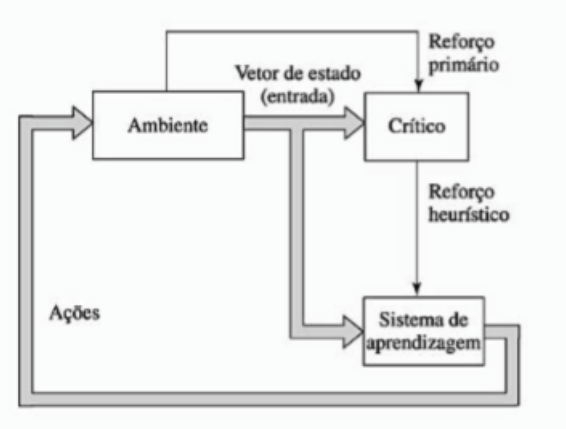

Figura 2.9 - Aprendizagem por reforço. (HAYKIN, 2001).

por exemplo, uma rede de duas camadas, onde uma camada recebe os dados de entrada (camada de entrada) enquanto que a outra camada utiliza a competição entre seus neurônios pela oportunidade de responder aos estímulos da camada de entrada (camada competitiva).

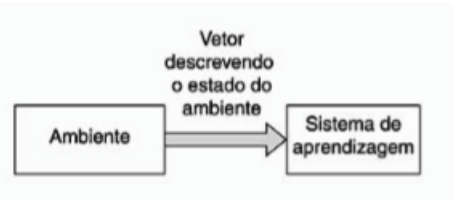

Figura 2.10 - Aprendizagem auto-organizada. (HAYKIN, 2001).

2.2.5.4 Aplicações dos métodos de aprendizado

Como já mencionado anteriormente, o método de aprendizado depende da aplicação do algoritmo. Os algoritmos de aprendizado por reforço mapeiam os estados possíveis do mundo e as ações possíveis para cada estado, sendo assim mais utilizados em sistemas de tomada de decisão de alto nível.

As aprendizagens supervisionada e não-supervisionada apresentam similaridades, divergindo basicamente na abordagem do problema. A abordagem supervisionada separa os padrões em *classes*, enquanto a não-supervisionada separa em *grupos*. Na prática, uma classe possui um título que a descreve, enquanto um grupo é apenas um

conjunto de padrões considerados similares, sem título definido. Supondo que uma rede deva lidar com imagens de dígitos manuscritos, a abordagem supervisionada retornaria a classe a qual ela julgou a entrada pertinente, neste exemplo o dígito de 0 a 9. Já o algoritmo não-supervisionado agruparia todos os dígitos similares em um mesmo conjunto, podendo inclusive criar mais de um grupo para padrões que naturalmente deviam ficar juntos.

É importante notar que o aprendizado supervisionado só é capaz de lidar com as classes que foram previamente apresentadas à rede durante o treinamento, o que torna tal abordagem ideal para problemas de classificação. Por outro lado, os algoritmos não-supervisionados são capazes de reconhecer novos padrões e agrupá-los corretamente, lidando melhor com padrões previamente desconhecidos, sendo frequentemente encontrados em, por exemplo, aplicações de mineração de dados.

2.3 Rede Neural Perceptron

Um dos primeiros modelos de redes neurais são as *perceptron*. Estas redes são constituídas de uma única camada, sendo uma das formas mais simples de representação de RNA, empregadas principalmente quando se trabalha com classificação de padrões linearmente separáveis (padrões em lados opostos de um hiperplano) (HAYKIN, 2001). De forma simplificada, uma rede perceptron de uma única camada consiste de uma unidade neural, com seus pesos sinápticos ajustáveis. O algoritmo voltado ao treinamento desse tipo de rede surgiu com Rosenblatt.

A Figura 2.11 apresenta o esquema de uma rede perceptron de um neurônio. É possível observar que essa rede é basicamente um somador (combinador linear) combinado a um limitador. O somador calcula uma combinação linear das entradas aplicadas aos pesos sinápticos, e tal resultado é submetido ao limitador, o qual realiza a função sinal (semelhante a função limiar, a saída é levada para +1 se a entrada for positiva e para -1 se for negativa).

Os pesos sinápticos do perceptron podem ser representados por $w_1, w_2, ..., w_n$; as entradas como $x_1, x_2, ..., x_n$ e o bias externo b. Dessa forma, podemos descrever, matematicamente, o campo local induzido do neurônio como:

$$v = \sum_{i=1}^{m} w_i x_i + b \tag{2.9}$$

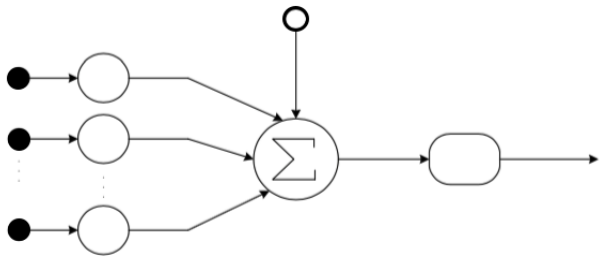

Figura 2.11 - Rede perceptron de 1 neurônio.

O objetivo da rede *perceptron* é conseguir classificar corretamente o conjunto de estímulos da entrada (x), em uma de duas classes ξ_1 ou ξ_2. Se a saída do perceptron for $+1$ as entradas x_n serão associadas à classe ξ_1, enquanto que se a saída for -1, serão associadas à ξ_2. Para garantir o bom funcionamento dessa rede é altamente importante que essas duas classes sejam linearmente separáveis. Na figura 2.12a temos o exemplo das classes suficientemente separadas, formando assim um hiperplano (nesse caso a linha pontilhada servindo como uma fronteira de decisão). Já em 2.12b as classes estão muito próximas tornando-se não linearmente separáveis.

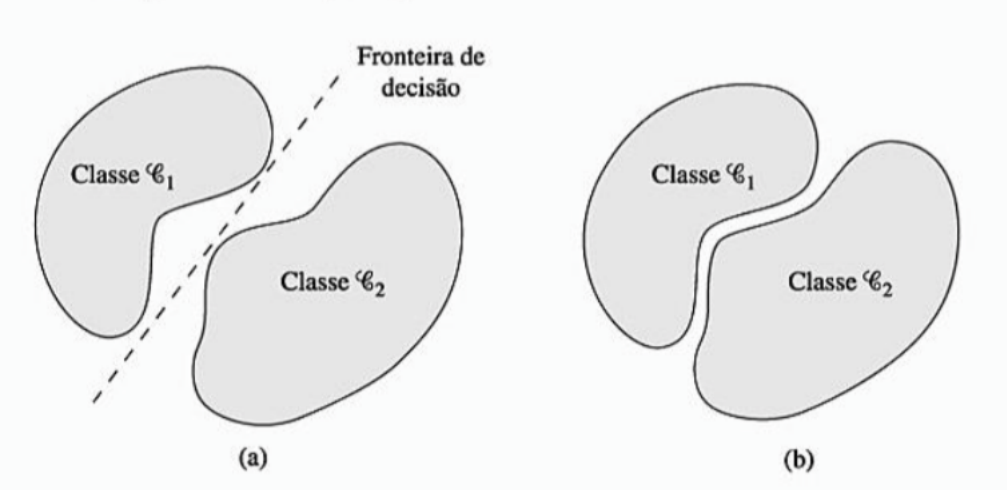

Figura 2.12 - (a) classes linearmente separáveis. (b) classes não linearmente separáveis.(HAYKIN, 2001).

Esta limitação de separabilidade foi demonstrada por Nils Nilson, Minsky e outros pesquisadores, o que expôs uma grande fragilidade deste modelo, apesar dos bons resultados iniciais. Um exemplo de problema não tratado pela *perceptron* de uma

camada é o ou-exclusivo.

Para este caso do ou-exclusivo, ao analisarmos uma rede perceptron de duas camadas (x_1, x_2) e dois pesos (w_1, w_2) e um limiar r, essa rede deveria encontrar os pesos designados, considerando a sua tabela verdade (figura 2.13) e as seguintes premissas:

x_1	x_2	saída
1	1	0
1	0	1
0	1	1
0	0	0

Figura 2.13 - Tabela verdade do ou-exclusivo.

- linha 1 da tabela verdade: $w_1 * 1 + w_2 * 1 < r$
- linha 2 da tabela verdade: $w_1 * 1 + 0 > r$
- linha 3 da tabela verdade: $0 + w_2 * 1 > r$
- linha 4 da tabela verdade: $0 + 0 < r$

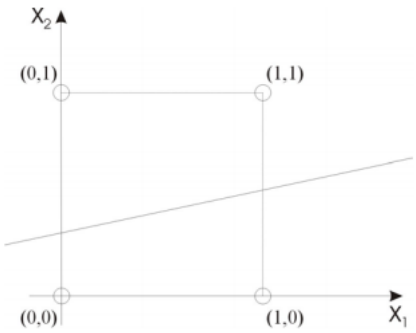

Figura 2.14 - Gráfico com as combinações possíveis do ou-exclusivo.

Uma vez que as duas classes não são linearmente separáveis, como demonstrado na figura 2.14, a perceptron de uma única camada não consegue convergir para tal problema. É perceptível a impossibilidade de se plotar uma linha reta capaz de separar em duas dimensões os pontos $\{(0,0);(1,1)\}$ de $\{(0,1);(1,0)\}$.

Para resolver essa limitação, seriam necessárias redes perceptron de múltiplas camadas, ou *multilayer perceptron* (MLP) (figura 2.15). Uma *multilayer perceptron* nada mais será que uma rede constituída de uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. A camada de entrada (nós fonte) têm a função de apresentar os padrões de entrada à rede, não realizando nenhum processamento. As camadas intermediárias, por sua vez, têm a função de extrair características que influenciarão nos pesos sinápticos. Por fim, a camada de saída apresenta os padrões de resposta da rede.

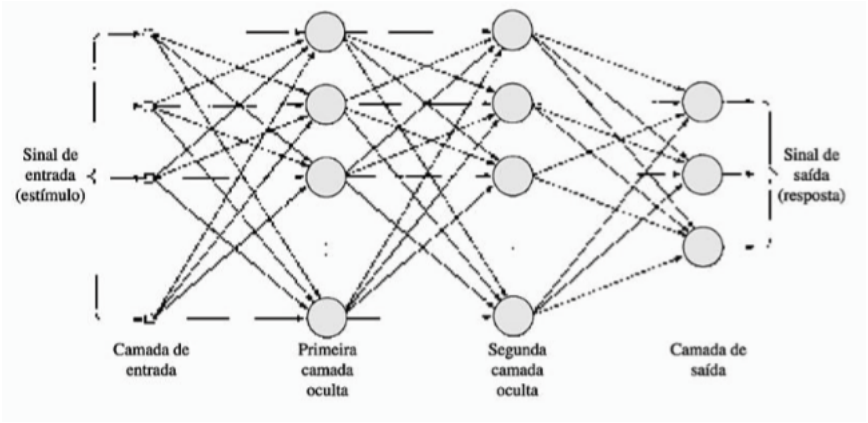

Figura 2.15 - Mulilayer perceptron com duas camadas ocultas.(HAYKIN, 2001).

Apesar de melhorar o desempenho da rede, a inserção de camadas ocultas implica diretamente no algoritmo de treinamento a ser adotado nessas redes. Esse problema foi um dos principais fatores que levaram a atenuação das pesquisas em RNA em meados da década de 70. Tal limitação foi contornada com o desenvolvimento do algoritmo de treinamento *backpropagation* ou retropropagação.

2.3.1 Algoritmo *backpropagation*

Em 1974 Werbos, na Universidade de Harvard, propôs em sua tese de doutorado "*Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Science*", a utilização desse método de aprendizado. Já em 1985 Parker aplicou as descobertas de Werbos no relatório do MIT "*Learning Logic*". Além deles, Rumelhart e demais membros do grupo de pesquisa Processamento Paralelo e Distribuído, contribuíram para a solidificação do *backpropagation*.

O *backpropagation*, assim como a regra delta, é baseado no gradiente descendente, o qual realiza o ajuste de pesos a fim de reduzir o erro da rede. Esse método pode ser dividido em duas partes: a propagação e a retropropagação.

Fase de propagação

a) Os padrões de saída são apresentados à camada de entrada de rede;

b) Para cada uma das camadas c_i, a partir da camada de entrada:

   - Os sinais de saída do neurônio da camada c_i servirá como entradas da camada c_{i+1}, e assim sucessivamente;

c) Os sinais da camada de saída serão comparados aos sinais esperados.

Fase de retropropagação

a) Partindo da última camada da rede até a primeira;

   - Os neurônios da camada atual ajustam seus pesos sinápticos de forma a reduzir o erro;
   - O erro de um neurônio é calculado através do valor de erro retropropagado dos neurônios das camadas seguintes conectados a ele, os quais serão ponderados considerando os pesos das conexões entre eles.

Algoritmo

Considerando dois neurônio j e k, uma descrição básica do algotimo *backpropagation* pode ser realizada através dos seguintes passo:

- Passo 1

  Os pesos sinápticos da rede são inicializados;

- Passo 2

  São apresentados à rede os padrões de saída desejados na forma:

$$\psi = (x_l^d, y_l^d)_{l=1}^L \tag{2.10}$$

Esse processo pode ser realizado apresentando-se um par entrada/saída por vez, ou todos de uma única vez. Uma diferença básica entre esses dois modos está na quantidade de memória requerida em cada ação. Outra diferença está no ajuste fino de pesos da rede.

- Passo 3

O próximo passo é o cálculo do erro da rede. Considerando o campo local induzido do neurônio j como:

$$v_j(n) = \sum_{i=0}^{m} w_{ij}(n) * y_i(n) + b_j(n) \tag{2.11}$$

Sendo m o número de entradas aplicadas ao neurônio j e $w_{j0}(n) = b_j(n)$. Dessa forma, a saída do neurônio j será:

$$y_j(n) = F_j(v_j(n)) \tag{2.12}$$

Onde $F_j(\cdot)$ representa a função de ativação, que representa a relação da entrada/ saída associada ao neurônio j.

Considerando agora o campo local induzido do neurônio k, temos:

$$v_k(n) = \sum_{i=0}^{m} w_{kj}(n) * y_j(n) + b_k(n) \tag{2.13}$$

Onde m agora representa o número de entradas aplicadas ao neurônio k e $w_{k0}(n) = b_k(n)$. Assim, a saído do neurônio k será:

$$y_k(n) = F_k(v_k(n)) \tag{2.14}$$

Sendo agora $F_k(\cdot)$ a função de ativação que simboliza a relacão da entrada/saída do neurônio k.

Com base nas equações 2.10 e 2.12 é possível generalizar a equação do campo local induzido $v_j^l(n)$, de um neurônio j, situado na camada l:

$$v_j^l = \sum_{i=0}^{m_l} w_{ji}^l(n) * y_i^{l-1}(n) \tag{2.15}$$

Onde m_l descreve o tamanho da camada l; $w_{ji}^l(n)$ o peso sináptico do neurônio j na camada l, o qual é alimentado pelo neurônio i da camada $l-1$ (camada anterior) e $y_i^{l-1}(n)$ a saída do neurônio i na camada $l-1$, na iteração n. Resultando em:

$$y_j^l(n) = F_j(v_j^l(n)) \tag{2.16}$$

Considerando que a primeira camada da rede *multilayer perceptron* não realiza nenhum processamento, se o neurônio j estiver localizado na primeira camada oculta sua entrada será:

$$y_j^0(n) = x_j^1(n) \tag{2.17}$$

Dessa forma, o índice 0 simboliza que a saída localiza-se nos nós de entrada e o índice 1 que o termo a direita da igualdade é entrada da primeira camada oculta.

Alternativamente, se o neurônio j pertencer a camda de saída, teremos:

$$y_j^L(n) = y_j^{out}(n) \tag{2.18}$$

Com $y_j^{out}(n)$ simbolizando o j-ésimo termo do vetor de saída da rede y^{out}, na iteração n, com L sendo a última camada da rede.

- Passo 4

Como sequência, calcula-se o erro da rede e ajusta-se os pesos sinápticos da mesma.

Para a camada de saída da rede o erro é calculado por:

$$e_k(n) = d_k(n) - y_k^{out}(n) \tag{2.19}$$

Onde d_k é o k-ésimo termo do vetor de respostas desejadas.

Para o atualização dos pesos sinápticos dos neurônios da camda de saída, temos:

$$w_{kj}(n+1) = w_{kj}(n) + \eta\delta_k(n) * y_k(n) \tag{2.20}$$

Já para as camadas ocultas da rede, a equação a ser utilizada é:

$$w_{ji}(n+1) = w_{ji}(n) + \eta\delta_j(n) * y_j(n) \tag{2.21}$$

O processo de apresentação do padrões de entrada e correção dos pesos sinápticos é repetido até que se atinja um número máximo de iterações (épocas), ou quando o erro médio quadrático for menor que um valor pré-detrminado, ou ainda quando o desempenho de generalização atingir o máximo.

Algumas boas práticas produzem melhores resultados ao se utilizar este algoritmo de treinamento (BROWNLEE, 2011):

- ajustar os parâmetros pra um lote de padrões costuma ser mais efetivo do que um por um para alguns problemas complexos;
- é vantajoso expor os padrões de treinamento em uma ordem aleatória a cada iteração do algoritmo;
- a taxa de aprendizado pode ser variada durante as iterações do *backpropagation*, evitando que a rede se torne demasiadamente sensível apenas aos padrões de treinamento.

2.4 *Autoencoders*

Os *autoencoders*, no geral, são um tipo de rede neural que lidam com codificação de dados, seja para compactá-los, reduzir ruidos, entre outros. São também um caso especial das redes diretas. Em outras palavras, um autoencoder é uma classe de rede que pode ser treinada para aprender características de forma não supervionada, sendo que essas características podem ser utilizadas em tarefas de aprendizado supervisionado posteriormente, como por exemplo, reconhecimento de objetos, entre outras tarefas da visão computacional. Normalmente, uma rede autoencoder é treinada para reproduzir na saída a sua entrada e não predizer alguma classe.

Os conceitos que compõem os autoencoders não são novos, já estavam presentes no ambiente de pesquisa de redes neurais há décadas, como em LeCun, 1987 (*"Modèles connexionistes de l'apprentissage"*); Bourlard and Kamp, 1988 (*"Auto-association by multilayer perceptrons andsingular value decomposition."*); Hinton and Zemel,1994 (*"Autoencoders, minimum description length, and Helmholtz free energy"*).

A arquitetura básica de uma rede autoencoder é apresentada na Figura 2.16. Esse tipo de rede é formada por pelo menos uma camada intermediária, além das camadas de entrada e saída possuirem o mesmo número de unidades.

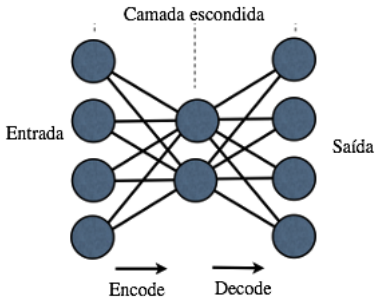

Figura 2.16 - Rede autoencoder.

O modelo já existente de rede neural direta, quando utilizado como autoencoder, se comporta como duas funções básicas para realizar as transformações na entrada: a função de extração de características e a de reconstrução. A de extração de características, também conhecida como codificadora (*encoder*), é responsável por mapear as variáveis do conjunto de dados de treinamento para uma representação latente. Já a de reconstrução, ou decodificadora (*decoder*) mapeia a representação produzida pela codificadora de volta ao espaço original.

O principal objetivo, durante o treinamento dessa rede, é minimizar o erro de reconstrução, isso é, a diferença entre a entrada apresentada e a saída obtida. De forma ideal, os parâmetros resultantes do treinamento devem ser capazes de representar a entrada por meio de uma redução de dimensionalidade, porém mantendo o máximo de informação acerca da distribuição de entrada (GOODFELLOW et al., 2016). Podemos ter casos de autoencoders subcompletos (*undercomplete*) e supercompletos (*overcomplete*):

- **subcompletos**: a representação codificada possui menos dimensões que a representação original, servindo para compactar a entrada;

- **supercompletos**: a representacão codificada possui mais dimensões que a representação original, buscando um padrão mais reduntante.

Assim como qualquer rede neural supervisionada, a acurácia de um *autoencoder* está diretamente ligada à seu conjunto de treinamento. Redes supervisionadas lidam bem com padrões previamente treinados, e podem apresentar comportamentos inesperados para entradas desconhecidas. Além destas considerações herdadas de uma rede supervisionada, os *autoencoders* subcompletos ainda possuem as suas próprias. Uma destas é que essa rede realiza uma compressão com perdas (*lossy compression*).

É possível encontrar algumas variações das redes autoencoders para casos ainda mais específicos: autoencoder com filtragem de ruído (*(denoising autoencoder*); autoencoder contrativo (*contractive autoencoder*) e autoencoder esparso (*sparse autoencoder*). Mais de uma técnica pode ser implementada ao mesmo tempo dependendo da aplicação da rede.

- **filtragem de ruído**: busca tornar a representação aprendida robusta a ruídos nos dados de entrada. Isso pode ser conseguido apresentando diversos padrões corrompidos de entrada para um mesmo padrão íntegro (sem ruído) de saída. Desta forma, a rede aprende a reconstruir a imagem original mesmo a partir de entradas danificados (com ruído);

- **contrativo**: sua ideia é fazer com que a representação aprendida seja robusta a pequenas mudanças em torno dos padrões de treinamento. Isto é alcançado adicionando-se um novo termo na função de erro da rede, que penaliza as representações indesejadas no espaço latente;

- **esparso**: esta abordagem desativa aleatoriamente, segundo uma função probabilística, neurônios da camada oculta durante o treinamento. Desta forma, reforça-se a redundância do aprendizado, não sendo necessário mais a ativação de todos os neurônios para a correta reconstrução do padrão. Seu uso é muito vantajoso para determinadas funções de ativação, como por exemplo a ReLU: mesmo no caso em que alguns neurônios "morram", os outros tem informação e treinamento suficiente para reconstruir a entrada.

2.4.1 *Stacked autoencoders*

Um *stacked autoencoder* (*empilhado*) é uma rede composta de múltiplos *autoencoders* (SENTHILKUMAR et al., 2015). A saída das camadas codificadoras são ligadas a outras camadas codificadoras, e o mesmo ocorre com as camadas decodificadoras. A vantagem de se empilhar estas estruturas é a mesma de se utilizar uma rede neural de múltiplas camadas. Mesmo que o *autoencoder* tenha duas camadas, cada uma tem uma tarefa específica. Ao usar apenas uma estrutura, tanto a codificação quanto a decodificação apresentam resultados satisfatórios apenas para problemas linearmente separáveis. Empilhando essas redes, cada camada consegue aprender um nível de representação diferente do padrão de entrada. A figura 2.17 ilustra um *stacked autoencoder*, este chamado de Neocognitron Fukushima.

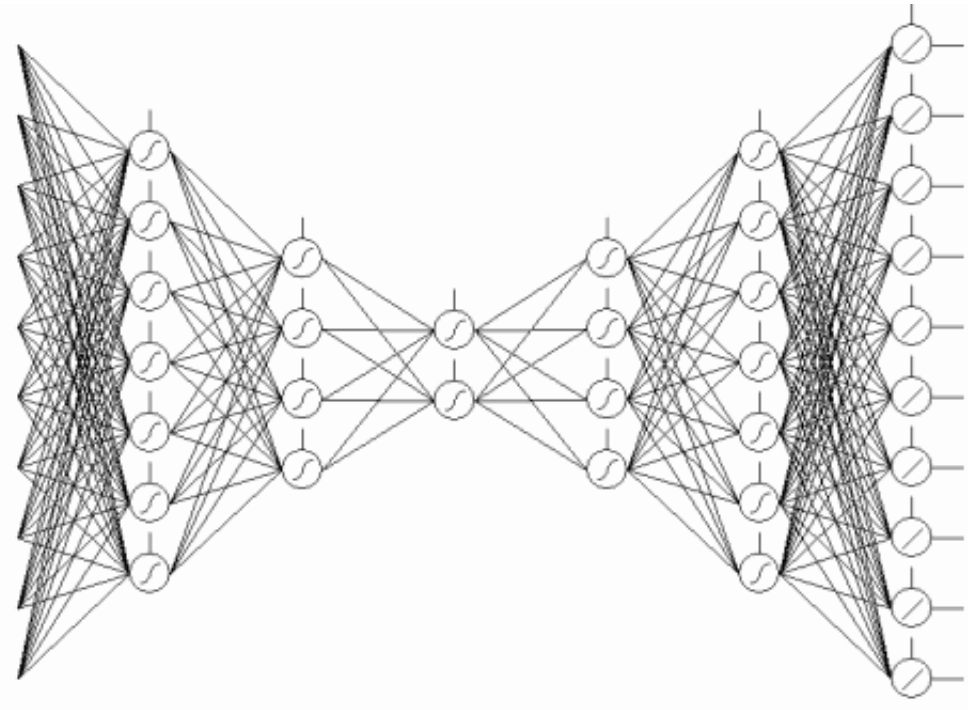

Figura 2.17 - Rede Neocognitron Fukushima como exemplo de *stacked autoencoder* (PERERVENKO, 2015).

2.5 Redes Convolucionais

As redes convolucionais, ou *convolutional neural networks* (CNN), são uma variação das redes multilayer perceptron. Sua principal característica é utilizar múltiplas

camadas heterogêneas, sendo as primeiras camadas responsáveis por pré-processar a imagem, reconhecendo padrões da imagem (forma, textura, ou até mesmo de cores caso possível). As últimas camadas são de neurônios perceptron, realizando a classificação do padrão de entrada.

Esse tipo de rede têm sido bastante aplicada a problemas de visão computacional, principalmente para tarefas de clasificação e detecção de objetos em imagem, substituindo as abordagens anteriores, que utilizavam extração manual das características das imagens para depois aplicar algum método de classificação. Um exemplo desses métodos é o SVM (*support vector machines*) ((SERMANET K. KAVUKCUOGLU; LECUN, 2013) e (SERMANET; LECUN, 2011)).

Considerando o conjunto de dados MNIST (*dataset* de dígitos manuscritos, com 60 mil padrões de treinamento e 10 mil padrões de validação), que é composto por imagens em tons de cinza de dimensões 28×28 pixels. Uma rede qualquer lidando com esses padrões teria 784 entradas. Caso opte-se, por exemplo, por uma rede MLP com apenas uma camada oculta com o dobro de unidades da camada de entrada (1568 para este *dataset*), apenas esta camada teria a ordem de 10^6 parâmetros para ajustar na fase de treinamento. Para sistemas de visão computacional que lidam com maiores resoluções, mesmo consideradas baixas (como por exemplo $300x400$), a quantidade de parâmetros necessária poderia facilmente chegar a ordens de 10^{10} para cada camada oculta.

A solução adotada pelas redes convolucionais para reduzir o número de parâmetros é baseada na metáfora biológica do processamento de imagens realizado pelo córtex do cérebro humano, onde esse processo é dividido em tarefas. Isto significa que a rede utiliza camadas heterogêneas, tendo camadas específicas para processar a imagem, amostrá-la ou classificá-la. Desta forma, é possível reduzir de forma considerável a quantidade de parâmetros treináveis pela rede, sem impactos visíveis na acurácia da rede. Diferentemente de uma rede densa, o número de parâmetros de uma camada convolucional não está ligado ao tamanho da entrada, e sim à resolução e quantidade de filtros utilizados na camada. Desta forma, a rede necessita de menos parâmetros, mesmo em imagens de maior resolução. Vale ressaltar que a redução relevante está em custo de memória, e não em custo de processamento.

2.5.1 Arquitetura de uma CNN

Uma típica arquitetura de uma rede convolucional é dividida em estágios. Cada estágio é formado por camadas de convolução em sequência, seguida por camadas de amostragem. Após a camada de entrada (correspondente à imagem) vários estágios podem ser empilhados, e após o estágio final da rede, são adicionadas uma ou mais camadas completamente conectadas (por exemplo camadas *perceptron*). A figura 2.18 ilustra uma arquitetura de rede convolucional.

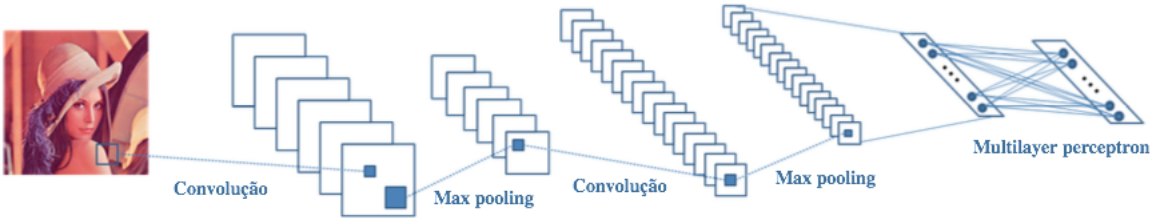

Figura 2.18 - Arquitetura tradicional de uma CNN.

A seguir serão apresentados com mais detalhes os tipos de camadas dessa arquitetura de rede.

2.5.1.1 Camadas convolucionais

Essas camadas aplicam um conjunto de convoluções à sua entrada. As operações de convolução, em processamento de imagens, são técnicas que modificam a intensidade de um pixel baseado nas intensidades de uma janela de sua vizinhança. A figura 2.19 ilustra uma operação de convolução.

Ao se aplicar uma operação de convolução, diferentes características são realçadas ou suavizadas, como demonstra a tabela 2.1. Os *kernels* das camadas convolucionais operam da mesma forma, com a diferença que os valores da máscara são adaptativos, se ajustando para processar características que ajudem a classificar as entradas de forma a minimizar o erro da saída.

Funções de ativação podem ser aplicadas em uma camada convolucional. Neste trabalho, as camadas deste tipo usam a função ReLU, pelas suas vantagens quanto ao gradiente e custo de treinamento.

		Imagem original.
	$\begin{bmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{bmatrix}$	Filtro gaussiano. Suaviza diferenças entre pixels, "borrando" a imagem.
	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$	Filtro de Sobel. Mostra as diferenças nos valores dos pixels em determinada direção. Possui variantes, uma para cada direção.
	$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$	Filtro emboss. Dá a ilusão de profundidade, enfatizando os pixels em determinada direção.
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	Filtro outline. Detecta bordas na imagem.
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	Filtro sharpen. Enfatiza as diferenças de pixels adjacentes.

Tabela 2.1 - Exemplos de filtros de convolução

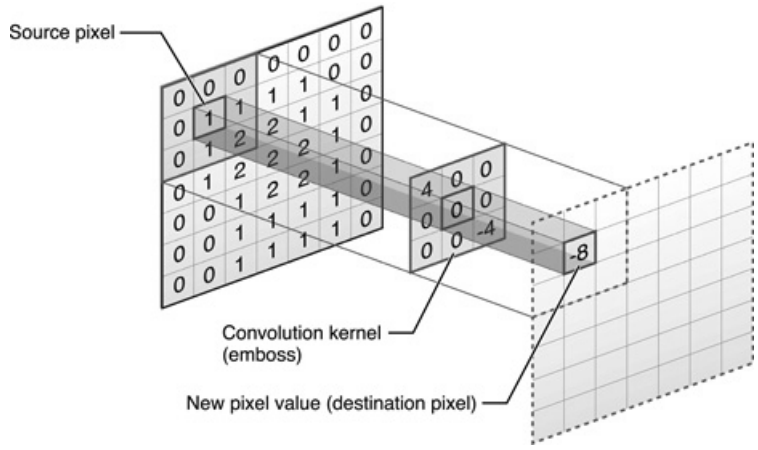

Figura 2.19 - Operação de convolução (adaptada de Apple (2016)).

2.5.1.2 Camadas de subamostragem

As camadas de subamostragem, ou *pooling* são responsáveis por reduzir a dimensão dos seus dados de entrada. Estas camadas costumam possuir três parâmetros principais:

- **tamanho da janela**: dimensões da amostragem que será reduzida para apenas um valor, de acordo com a estratégia adotada;
- **passo (*stride*)**: deslocamento da janela a cada iteração da amostragem. Seu tamanho padrão é o mesmo da janela;
- **estratégia de amostragem**: as estratégias mais comuns são a de valor máximo, que amostra a janela para o valor máximo presente nela, e a de valor médio, que amostra a janela para o valor médio de seus elementos.

A configuração mais comum para essas camadas é a de *pooling* máximo, com janela e passo de dimensões $2x2$, como exemplificado na figura 2.20 (traduzir a figura). Esta operação resulta em uma camada que descarta 75% de sua entrada.

Além de reduzir a dimensão da representação dos dados, a amostragem produz uma invariância a pequenas mudanças e distorções locais. Tanto o modelo de camada convolucional quanto de amostragem são inspiradas diretamente nas características de células simples e complexas da neurociência visual (LECUN; HINTON, 2015).

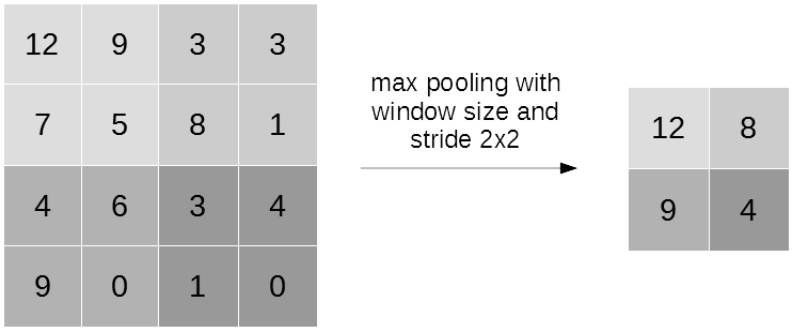

Figura 2.20 - Operação de *pooling* máximo com janela e passo de dimensões 2x2.

2.5.1.3 Camadas completamente conectadas

Normalmente é uma rede *multilayer perceptron*, que recebe a entrada pré-processada das camadas convolucionais e a classifica. Isso mitiga a responsabilidade de processar padrões visuais (como formas, texturas, entre outros) pela parte densa da rede, além de receber uma entrada em que as características úteis para a classificação estão reforçadas. Desta forma, a motivação do uso do modelo convolucional de redes neurais é a busca por uma maior acurácia da rede.

2.5.2 Redes completamente convolucionais

Um arranjo de camadas convolucionais e de amostragem, arquitetura comum de uma CNN, pode ser seguido também por camadas de interpolação e novas camadas convolucionais. Desta forma, é possível fazer a rede comportar-se de forma similar a um *autoencoder*, utilizando as camadas intermediárias para aprender a reduzir a imagem em padrões e reconstruí-la com interpolação e outras convoluções.

A aplicação de redes completamente convolucionais (*fully convolutional network* - FCN) é diferente de uma CNN comum. A rede clássica possui a mesma saída de uma rede *multilayer perceptron*, que é um vetor de pertinências para cada classe que a rede foi treinada para reconhecer, e esta saída é única para a imagem de entrada inteira. Enquanto isso, uma rede FCN interpreta a imagem de entrada e retorna como saída também uma imagem. De forma similar a um *autoencoder*, é possível tanto reconstruir a imagem original, e desta forma utilizar a rede FCN como um compactador, quanto transformá-la para outro espaço de informações.

Uma das utilizações dessas redes é classificar pixel a pixel a imagem a ela submetida.

Um exemplo de rede FCN é apresentado na figura 2.21, arquitetura de rede deste trabalho, baseado no trabalho de Long et al. (2015). A rede deste exemplo rotula cada pixel da imagem recebida dentro de 35 classes. Esta abordagem tem como vantagem segmentar a entrada da rede ao mesmo tempo que a classifica, não sendo necessária outra técnica para seccionar a imagem.

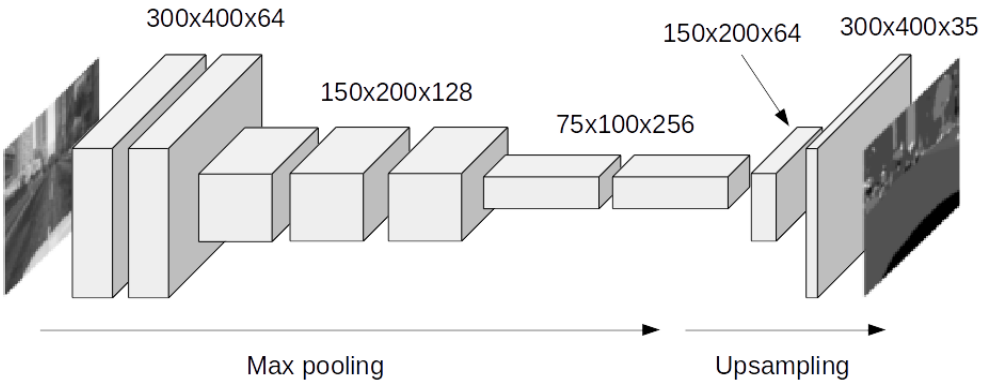

Figura 2.21 - Exemplo de rede completamente convolucional.

2.6 Visão Computacional

A visão computacional está diretamente relacionada à extração de informação em imagens. Em outras palavras, é o detalhamento, de forma clara, dos objetos contidos em uma imagem, de forma a imitar a visão humana. Diferentemente do processamento de imagens, o qual transforma uma imagem em outras, a visão trata da obtenção, manipulação e uso de dados contidos na imagem (SZELISKI, 2010).

Um sistema de visão computacional pode ser dividido em fases de:

- **aquisição**: capta as imagens, tentando simular a ação realizada pelo olho humano. Nessa fase podem ser utilizados dispositivos com: filmadoras, *scanners*, máquinas fotográficas, etc;
- **processamento de imagens**: é responsável por otimizar a imagem, retirando ruídos, destacando bordas, suavizando a imagem, entre outros;
- **segmentação**: subdivide a imagem em regiões de interesse;
- **extração de características**: tem como função obter uma codificação

numérica capaz de representar uma determinada imagem, de forma a possibilitar a identificação da mesma. Gera um conjuntos de características do objeto de interesse;

- **reconhecimento de padrões**: classifica ou agrupa as imagens de acordo com seu conjunto de características.

2.6.1 Segmentação

Segmentação é o processo que subdivide uma imagem em suas partes ou objetos constituintes (ZAITOUN; AQEL, 2015). O nível de subdivisão depende do problema a ser resolvido, isto é, a segmentação deve finalizar assim que os objetos de interesse tenham sido isolados da imagem. Segmentar uma imagem é um passo importante para a análise da informação, permitindo que outros algoritmos processem regiões de interesse isoladas da imagem.

Grande parte dos algoritmos de segmentação trabalha com os princípios de similaridade, que agrupa os pixels que possuem propriedades similares, e de descontinuidade, que utiliza os contornos da imagem para dividi-la em regiões menores (MASOOD et al., 2015).

Um problema que pode reduzir a qualidade da segmentação é o ruído. Este problema pode ser causado pelo *hardware* do sistema de aquisição, e também por problemas na cena, como excesso ou falta de luminosidade, movimento de objetos na cena, entre outros.

A seguir são apresentados alguns métodos de segmentação presentes na literatura (MASOOD et al., 2015).

Limiarização

É um dos métodos de segmentação mais simples, rápidos e fáceis de implementar. No geral, são utilizados para separar um objeto de interesse na imagem utilizando alguma característica notável, normalmente ligada à intensidade do pixel.

Esse algoritmo possui algumas limitações, entre elas a necessidade de um bom contraste entre o objeto a ser segmentado e seu fundo. Além disso, a continuidade do objeto não é assegurada ao utilizar-se esta técnica.

Crescimento de região

Neste método, um ponto inicial é manualmente escolhido e, a partir dele, todos os pontos da vizinhança que possuem uma intensidade similar são iterativamente adicionados à região. Um de seus usos inclui a detecção de tumores na área médica.

Sua principal limitação é a necessidade de inicialização manual para cada região a ser segmentada. Além disso, a escolha dos pontos iniciais influencia diretamente na qualidade final da segmentação. Por lidar com similaridade de intensidades, estes algoritmos também tendem a retornar resultados menos precisos para regiões com efeitos de borrão.

Classificadores

Métodos classificadores usam o reconhecimento de padrões como base para a segmentação. Estes algoritmos são treinados de forma supervisionada para subdividir a imagem através de um parâmetro, comumente a intensidade de cores. Nesta família de técnicas encontram-se as árvores de decisão, os métodos de máxima verossimilhança (*maximum likelihood*), além das redes neurais supervisionadas.

Métodos deformáveis

Estes algoritmos trabalham com o conceito de ajustar os parâmetros de curvas fechadas para traçar o contorno de objetos. Essa abordagem também é chamada de método de contornos ativos. Uma de suas desvantagens é a necessidade de uma boa inicialização da curva que irá assumir o contorno do objeto de interesse. É possível encontrar na literatura o uso deste método para o rastreamento de objetos em vídeo, pois o contorno do quadro anterior é uma boa inicialização para o algoritmo atualizar a silhueta do objeto de interesse.

2.6.2 Visão computacional e redes convolucionais

Antes das redes convolucionais, o processo de visão era modular, respeitando as fases citadas anteriormente que fossem pertinentes à aplicação, uma vez que nem todas as fases são necessárias para determinados problemas. Devido à modularização, a literatura oferece algumas opções de algoritmos para cada uma de suas etapas.

Uma das consequências de haver múltiplas possibilidades para cada fase da visão é de que é possível eleger algoritmos que melhor se adaptam para o problema abordado.

Por exemplo, algoritmos que identifiquem prédios poderiam usar identificadores de retas, uma vez que não é possível confiar em informações de cores. Por outro lado, reconhecedores de árvores provavelmente dariam melhor resultados segmentando por tonalidades de pixels ao invés de formas.

No entanto, apesar de diversas técnicas disponíveis e considerável capacidade computacional, a concretização do desejo de um computador interpretar uma cena em toda sua complexidade e em diferentes níveis (contagem e identificação de objetos, reconhecimento de padrões, simulações de trajetórias, etc.) ainda permanece distante (SZELISKI, 2010).

Nesse sentido, as redes convolucionais trouxeram um novo leque de possibilidades à área de visão computacional. Uma CNN é capaz de processar, segmentar (no caso da FCN), extrair características e classificar a imagem. Diferente das técnicas com abordagem mais modular, um algoritmo de treinamento ajusta os parâmetros da rede para aumentar sua acurácia. Em termos práticos, isto transfere a responsabilidade de escolher os algoritmos de processamento de imagens, extração de características e reconhecimento de padrões do projetista para a rede.

Em contrapartida, igual já citado anteriormente, é necessário levar em consideração o treinamento de uma rede neural, tanto em relação a seu custo em tempo quanto a dependência de um conjunto de treinamento adequado. Além disso, por mais que não seja necessário testar diferentes algoritmos, é vantajoso testar diferentes arquiteturas de rede.

2.6.3 *Datasets*

Em visão computacional, um *dataset* é um conjunto de imagens utilizado para treinamento e validação de algoritmos. Diferentes *datasets* podem ser utilizados dependendo do domínio do problema, seja classificação, detecção, rastreamento, segmentação, aplicação médica, entre outros. A seguir são apresentados alguns exemplos de *datasets* com suas breves descrições:

- **MNIST** [1]: *dataset* de dígitos manuscritos em tamanho 28x28 em tons de cinza, composto por um conjunto de treinamento de 60 mil elementos, e 10 mil padrões de validação;

[1] http://yann.lecun.com/exdb/mnist/

- **Pascal Context** [2]: é um conjunto de imagens do *PASCAL Visual Object Classes Challenge 2010 (PASCAL VOC2010)* alterado, de forma que toda a cena esteja classificada, e não apenas uma região de interesse;
- **COCO** [3]: *dataset* para reconhecimento e segmentação de objetos em determinado contexto, isto é, a composição da cena é necessária para uma classificação precisa;
- **Cityscapes** [4]: contém um conjunto de vídeos estéreo gravadas em ruas de 50 diferentes cidades, com uma parcela de quadros segmentados e rotulados;
- **Playing for Data** [5]: conjunto de imagens sintéticas renderizadas por jogos modernos, com um razoável nível de realismo.

[2]http://www.cs.stanford.edu/ roozbeh/pascal-context/
[3]http://cocodataset.org/
[4]https://www.cityscapes-dataset.com/
[5]https://download.visinf.tu-darmstadt.de/data/from_games/

CAPÍTULO 3

Metodologia

Nesse capítulo serão apresentadas ferramentas e metodologia aplicadas para a obtenção dos resultados, i.e., imagens segmentadas. A abordagem proposta neste trabalho foi dividida em 5 fases. A primeira fase trata da eleição de um *dataset* para o ajuste de pesos da rede. A fase 2 cuida da adaptação dos padrões de treinamento para o problema a ser resolvido. A terceira fase consiste na determinação do tipo de rede a ser utilizada. A fase 4 trata da implementação e validação de diferentes arquiteturas de FCN a fim de obter melhores resultados. Por fim, a quinta fase cuida da validação dos resultados obtidos.

3.1 Obtenção dos Dados de Treinamento

Em qualquer rede neural, os padrões de treinamento dão um ponto de partida para o trabalho da rede. Esses padrões tem que ser compatíveis com a saída desejada do sistema, ou seja, o formato e padrão da saída da rede tem que ser o mesmo dos dados de treinamento. Além disso, a capacidade de classificação da rede *perceptron* está diretamente ligado às classes apresentadas a ela. Um modelo supervisionado não é capaz de aprender novos padrões em tempo de execução, o que significa que o treinamento deve ser planejado e adaptado desde o início.

A qualidade e quantidade de padrões do *dataset* também tende a influenciar a capacidade de aprendizado da rede (BRAGA, 2007). Um conjunto de treinamento de qualidade possui uma quantidade de padrões por classe o mais diversificado possível, pouca ambiguidade entre classes, entre outros.

O *dataset* escolhido foi o *Playing for Data*. Esse conjunto de treinamento é composto de 25 mil imagens sintetizadas por um jogo de computador fotorrealístico (RICHTER et al., 2016). Segundo o autor, alguns jogos virtuais modernos possuem mundos extensos e realistas. Esse realismo não está apenas na fidelidade da aparência dos materiais ou iluminação, mas também está na composição dos objetos e ambientes, texturas, movimentação e gestos de veículos e personagens. A presença de pequenos objetos que enriquecem a cena e a interação entre os personagens e o ambiente também contribuem para o fotorrealismo. A figura 3.1 ilustra amostras de imagens desse *dataset*, demonstrando sua variedade de cenas e fotorrealismo. A figura 3.2 apresenta, em escala logarítmica, a quantidade de *pixels* distribuídos em

algumas classes do *Playing for Data*.

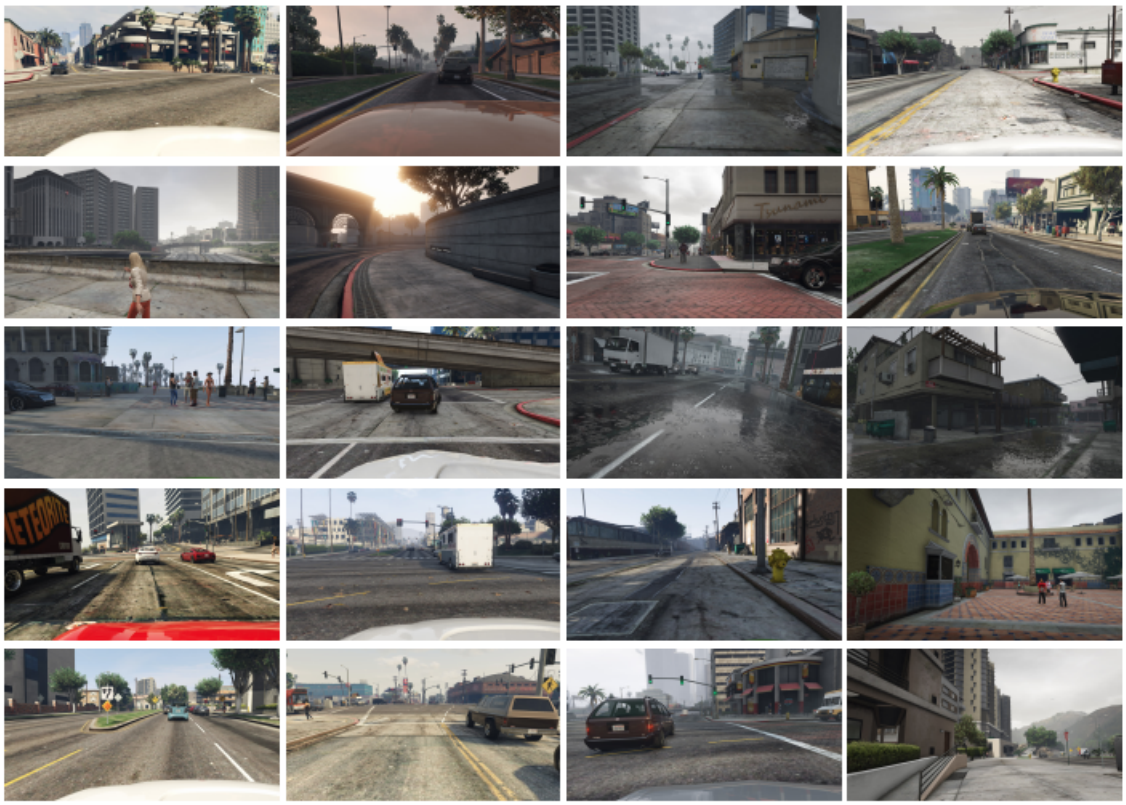

Figura 3.1 - Amostras de imagens do *Playing for Data* (RICHTER et al., 2016).

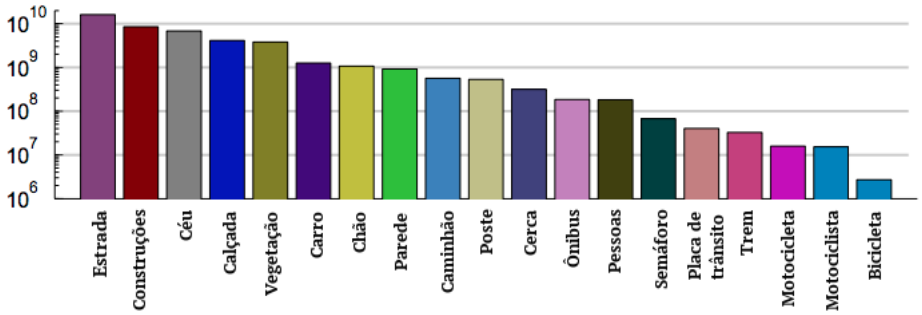

Figura 3.2 - Número de *pixels* pertencentes a determinadas classes, em escala logarítmica (RICHTER et al., 2016).

No geral, mundos virtuais de código-fonte aberto não possuem conteúdo tão exten-

Tabela 3.1 - Informações da placa NVIDIA GTX 1050.

Parâmetros	Valores
Arquitetura de placa de vídeo	Pascal
Núcleos NVIDIA CUDA ®	640
Framebuffer	2 GB GDDR5
Velocidade da memória	7 Gbps
Boost Clock Relativo	1,3x
Boost Clock Real	1.455 MHz

sivo, ou aparência tão realista quanto o de jogos comerciais. Em contrapartida, as operações internas e conteúdo dos *softwares* proprietários são em sua maioria inacessíveis. Por mais que não seja possível acessar todas as operações internas do jogo, é possível injetar um *wrapper* entre o *software* e o sistema operacional, permitindo aos desenvolvedores registrar, modificar e reproduzir comandos de renderização.

Ao interceptar as informações gráficas, é possível criar uma tabela *hash*[1] dos recursos de renderização (*shaders*, texturas, informações geométricas) comunicadas para o *hardware* gráfico. Dessa forma, é possível propagar rótulos na própria imagem, reduzindo o esforço manual de catalogação. Além disso, é possível inclusive propagar rótulos entre diferentes quadros, reduzindo drasticamente o tempo e esforço necessário para elaborar o conjunto de imagens classificadas. A figura 3.3 ilustra o resultado final do processo de rotulação em algumas amostras do *dataset*.

Neste trabalho apenas uma parcela do *dataset* foi utilizado (2.500 quadros). Tal redução foi recomendada devido ao custo computacional envolvido no processo de treinamento da rede. Quando um grande processamento aritmético não é possível, ou desejável, uma alternativa é a utilização de placas gráficas intermediárias. Tais placas possuem um bom custo benefício (poder computacional x custo). A placa adotada no trabalho foi uma NVIDIA GTX 1050 ®. As informações de tal placa são apresentadas na tabela 3.1. Vale ressaltar que uma maior capacidade de processamento é mais desejável na fase de treinamento. Após treinada, a mesma pode ser utilizada em arquiteturas de mais baixo poder computacional ainda, mantendo os parâmetros treinados em fases anteriores.

Devido à aplicações futuras da rede, os padrões de treinamento foram adaptados, ou

[1] tabelas *hash* são estruturas comuns para problemas de armazenagem e busca de dados, que associam chaves de pesquisa a valores.

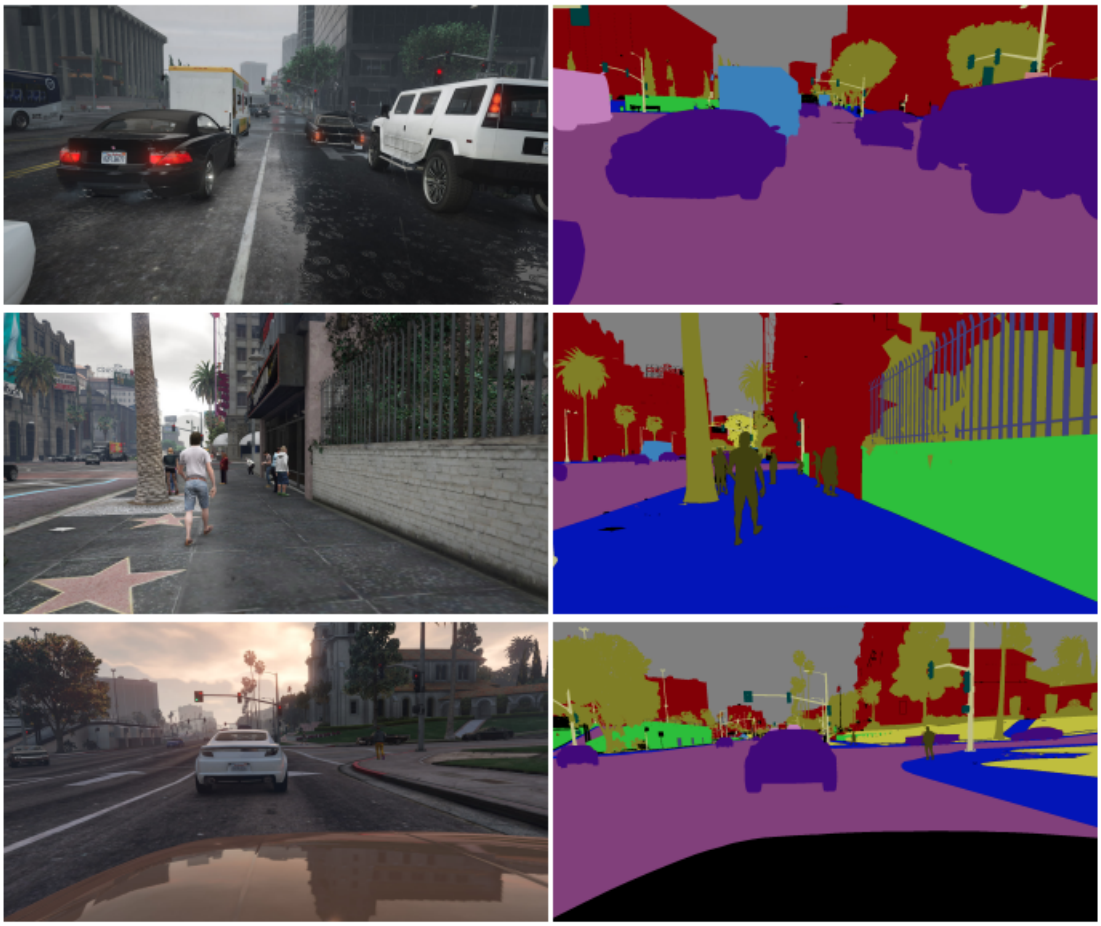

Figura 3.3 - Imagens do *Playing for Data*. À direita seus respectivos *ground truths* (RICHTER et al., 2016).

seja, o conjunto de imagens utilizado foi transformado de imagens coloridas para tons de cinza, além de serem cortadas para uma nova proporção (de *widescreen* para $4:3$) e redimensionadas para a resolução 300×400. Tais modificações foram convenientes, uma vez que espera-se utilizar tal rede para o sistema de visão computacional do robô LEIA 1 [2]. O sistema de captura de imagens deste robô é configurado desta maneira, visando uma melhor taxa de quadros.

3.2 Ferramentas utilizadas

Para o presente trabalho, a linguagem de programação Python foi utilizada, com a biblioteca matemática Theano (BERGSTRA et al., 2010), e a biblioteca Keras (CHOL-

[2] robô do Laboratório para Educação e Inovação em Automação (LEIA) da Escola de Engenharia Elétrica, Mecânica e da Computação, da Universidade Federal de Goiás (UFG).

LET, 2017) para prototipar modelos sequenciais de redes convolucionais.

3.2.1 Python

Python é uma linguagem de programação interpretada, orientada a objetos, de alto nível com tipagem dinâmica (ROSSUM, 1995). Essas características a tornam uma escolha atrativa pelo seu rápido desenvolvimento de aplicações, legibilidade e custo de manutenção de código. Além disso, a linguagem suporta módulos e pacotes, o que encoraja a modularidade e reuso do código.

A seguir, algumas considerações são feitas sobre Python em relação a outras linguagens. Essas comparações concentram-se em assuntos de linguagem apenas, já que a escolha de linguagem, na prática, depende também de custo financeiro, disponibilidade, treinamento e investimentos prévios.

- **Java**: programas escritos em Python tendem a executar mais lentos do que seus equivalentes em Java, mas tipicamente possuem de 3 a 5 vezes menos código. Essa perda de performance é atribuída em grande parte à tipagem dinâmica, que precisa avaliar tipos em tempo de execução;

- **C++**: as considerações de Java também valem pra C++, com a diferença de que um código em Python normalmente é 5 a 10 vezes menor do que seu equivalente nessa linguagem;

- **Perl**: possui algumas características similares, e ambas são frequentemente utilizadas em scripts do sistema UNIX, mas as diferenças estão na filosofia das duas. Perl enfatiza o suporte a leitura de arquivos, geração de relatórios, entre outras tarefas orientadas a aplicativos. A ênfase do Python é de ser uma linguagem mais genérica, orientada a objetos. Como consequência, Perl tem uma performance melhor do que Python no seu nicho específico, mas possui menos domínios de aplicação;

- **Matlab**: enquanto o Python busca ser uma linguagem de programação mais genérica, com recursos de rede, *threading*, entre outros, o Matlab concentra-se no ambiente computacional numérico, destacando-se em problemas de álgebra matricial, processamento de dados e plotagem. Através de pacotes específicos (*numpy*, *scipy* e *matplotlib*) é possível obter funcionalidades similares às do Matlab, com a vantagem de ser uma linguagem livre

e de código aberto. Vale ressaltar que não existem bibliotecas equivalentes ao *Simulink* em Python.

Por mais que tenha menos performance, a legibilidade e manutenção de uma aplicação Python é superior a outras linguagem. O custo de se implementar, depurar e validar uma solução nessa linguagem é muito menor do que em outras consideradas nesse trabalho. Para contornar a perda natural de performance do Python por ser tanto uma linguagem interpretada quanto possuir tipagem dinâmica, a biblioteca *Theano* é utilizada para aproximar seu tempo de execução a outras linguagens compiladas.

3.2.2 Theano

Theano pode ser definida como uma biblioteca Phyton livre, de código aberto, a qual permite definir, otimizar e avaliar expressões matemáticas envolvendo vetores multidimensionais de forma eficiente (BERGSTRA et al., 2010). Em outras palavras, o Theano permite a definição simbólica de expressões matemáticas, e pode calculá-las tanto em CPUs quanto em GPUs (*Graphics Processing Units*) de forma transparente, isto é, sem alterações no código fonte. Essa biblioteca também fornece um tipo de dado de matriz n-dimensional, além de várias funções para indexação, remodelação e execução de cálculo elementares (exponenciação, logaritmo, seno, etc).

Ainda segundo Bergstra et al. (2010), o Theano é capaz de melhorar, em termos de performance e precisão, o processamento matemático, pois armazena as expressões matemáticas como um grafo de variáveis e operações, as quais são podadas e otimizadas em tempo de compilação. A performance é melhorada, por exemplo, ao ignorar as variáveis que não são necessárias ao cálculo final da saída, reutilização de resultados parciais para evitar computação redundante, cálculo local sempre que possível das operações para minimizar o uso de memória, entre outros. Já a precisão pode ser observada na aplicação da estabilidade numérica a fim de evitar ou minimizar o erro devido às aproximações de *hardware*.

Ao utilizar-se essa biblioteca, uma parte da perda de performance por conta da interpretação e tipagem dinâmica é suprimida. Isso é resultado da compilação do grafo de operações em um código C++ equivalente. Por exemplo, ao escrever uma multiplicação de matrizes, um grafo simbólico é gerado e compilado. Esse arquivo é então executado em segundo plano toda vez que essa operação for instanciada. Dessa

forma, é possível unir a performance de uma linguagem compilada com a sintaxe limpa e alta legibilidade do Python.

3.2.3 Keras

Keras é uma API de alto nível para rápida prototipagem de redes neurais (CHOLLET, 2017). Esta biblioteca usa como base o Theano, ou outras bibliotecas (TensorFlow ou CNTK), para implementar diversos tipos de camadas de redes, de diferentes tipos (convolucionais, *pooling*, *perceptron*, recorrentes, entre outras). É possível prototipar diferentes modelos de rede, tanto sequenciais quanto em formato de grafo, com esta biblioteca. O presente trabalho utiliza redes sequenciais para obter seus resultados. Também possui interfaces simples e legíveis, além de ser facilmente estendida.

Esta biblioteca foi utilizada neste trabalho visando a rápida prototipagem de diferentes arquiteturas de redes, diminuindo o tempo necessário para testes e validação da arquitetura da rede a ser utilizada.

3.3 Arquiteturas de rede implementadas

A arquitetura de uma rede neural pode ser interpretada como as características paramétricas das suas camadas e a disposição delas na rede. A escolha da arquitetura influencia diretamente no resultado final da rede. Dependendo do problema, uma rede com poucas conexões pode não convergir satisfatoriamente, devido à escassez de parâmetros ajustáveis. Em contrapartida, uma rede com muitas conexões, pode haver um ajuste excessivo durante o treinamento, que pode prejudicar a capacidade de generalização da rede (FERREIRA, 2004). Normalmente, o número necessário de conexões é avaliado de forma empírica. Isso ocorre pelo fato de que a quantidade de parâmetros comumente está ligada às características do problema abordado.

Neste trabalho foram testadas duas configurações de rede, apresentadas nas tabelas 3.2 e 3.3. Foram utilizadas camadas convolucionais, de *pooling*, *upsampling* e de *dropout*, caracterizando assim redes *fully convolutional*. Vale ressaltar que as camadas de *dropout* apresentam seu comportamento padrão apenas durante o treinamento.

As redes implementadas neste trabalho são do tipo *fully convolutional networks*, pois são capazes de lidar com a classificação pixel a pixel, e por consequência a segmentação, de uma imagem de entrada.

Uma descrição da implementação das arquiteturas, treinamento e validação é apresentada no apêndice A em formato de pseudocódigo.

3.3.1 Primeira arquitetura

A figura 3.4 ilustra a arquitetura da rede da tabela 3.2. É passado como entrada da rede uma imagem em tons de cinza. Uma característica das redes FCN é que apenas o número de canais da imagem é relevante, e a saída pode ser construída para qualquer resolução, mas neste trabalho apenas imagens de resolução 300×400 são utilizadas. Essa resolução foi adotada devido às configurações da câmera do robô LEIA 1 (futura aplicação da rede). Essa imagem de entrada é passada por duas camadas convolucionais, cada uma com 32 filtros, cada filtro com dimensões 5×5. Essas camadas utilizam a função de ativação ReLU, e realçam características específicas das imagens utilizando diversos filtros adaptáveis. Esses filtros são ajustados durante o treinamento para serem sensíveis a padrões específicos, como texturas, formas, entre outras características visuais.

A seguir, a imagem passa por uma camada de *pooling*. Para cada iteração, uma janela da imagem é processada e um valor é retornado, de acordo com a estratégia escolhida. Neste trabalho, a janela possui tamanho 2×2, e o valor máximo é retornado a cada iteração. O passo representa o incremento da janela entre as iterações, e neste trabalho ele possui dimensões 2×2. Esta camada reduz 75% das ativações submetidas à ela, mantendo apenas os pixels mais ativados de cada janela. O próximo passo da rede é submeter a imagem a duas camadas convolucionais em sequência, com 64 filtros de dimensões $5x5$ e função de ativação ReLU. Em termos práticos, essas camadas possuem mais parâmetros ajustáveis que as primeiras, visando um melhor aprendizado de características de mais alto nível.

Após, a figura passa por outra camada de *pooling*, com as mesmas configurações da primeira, e depois por outra camada convolucional, esta com 128 filtros de dimensões $3x3$. Esta é a última camada do processo similar ao de codificação de um *autoencoder*. A partir dessa camada, a imagem passa por crescimentos para voltar às suas dimensões originais. A primeira camada de *upsampling* recebe a saída dessa última convolução e quadruplica a quantidade de *pixels*, crescendo em um fator 2×2 a entrada recebida. A primeira camada de *dropout* é aplicada nesse ponto da rede. Esta camada tem como função desativar uma porcentagem de ativações aleatórias submetidas à ela durante o treinamento. Nesta rede, 50% de ativações são desligadas

ao acaso durante o treinamento.

A seguir, uma camada convolucional é aplicada à imagem, com 64 filtros e janelas de dimensões 1×1. Essas janelas unitárias exercem um papel similar à de uma camada densa *perceptron*. Essa camada é seguida por uma operação de *upsampling* de fator 2×2 também, e seguido por outro *dropout* de 50%. A última camada da rede é convolucional, de 35 filtros de dimensões 1×1. Cada canal dessa camada reflete na pertinência do pixel à determinada classe, isto é, para cada imagem de entrada, é gerada uma figura de saída com 35 canais, e cada canal é uma representação da pertinência de cada pixel à i-ésima classe. Essas 35 classes reconhecidas são as do *dataset* escolhido.

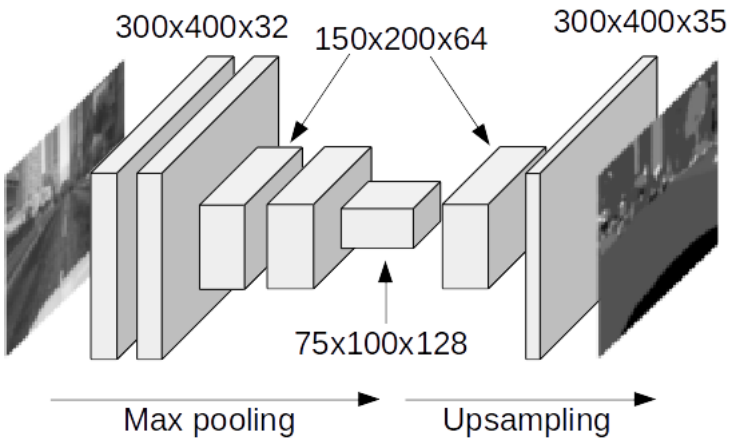

Figura 3.4 - Arquitetura de rede FCN.

3.3.2 Segunda arquitetura

Com o objetivo de aumentar o número de parâmetros ajustáveis, uma segunda arquitetura de rede foi avaliada. Esta arquitetura está ilustrada na figura 3.5, descrita na tabela 3.3 e utilizada no trabalho Rodrigues et al. (2017). Assim como na primeira arquitetura, a entrada é uma imagem em tons de cinza. As duas primeiras camadas, do tipo convolucional, possuem 64 filtros de resolução 5×5, e utilizam a função de ativação ReLU. A próxima camada é de *pooling*, com a janela e o passo de tamanho 2×2, e a estratégia de valor máximo.

A primeira camada de *dropout* vem a seguir, com uma taxa de desativação de 25% de

Tabela 3.2 - Arquitetura 1

Índice	Tipo	Parâmetros	Formato de saída
1	Convolucional	Número de filtros: 32 Janela de convolução: 5×5 Função de ativação: ReLU	$300 \times 400 \times 32$
2	Convolucional	Número de filtros: 32 Janela de convolução: 5×5 Função de ativação: ReLU	$300 \times 400 \times 32$
3	*Pooling*	Estratégia: valor máximo Janela: 2×2 Passo: 2×2	$150 \times 200 \times 32$
4	Convolucional	Número de filtros: 64 Janela de convolução: 5×5 Função de ativação: ReLU	$150 \times 200 \times 64$
5	Convolucional	Número de filtros: 64 Janela de convolução: 5×5 Função de ativação: ReLU	$150 \times 200 \times 64$
6	*Pooling*	Estratégia: valor máximo Janela: 2×2 Passo: 2×2	$75 \times 100 \times 64$
7	Convolucional	Número de filtros: 128 Janela de convolução: 3×3 Função de ativação: ReLU	$75 \times 100 \times 128$
8	*Upsampling*	Fator de crescimento: 2×2	$150 \times 200 \times 128$
9	*Dropout*	Taxa de desativação: 50%	$150 \times 200 \times 128$
10	Convolucional	Número de filtros: 64 Janela de convolução: 1×1 Função de ativação: ReLU	$150x \times 200 \times 64$
11	*Upsampling*	Fator de crescimento: 2×2	$300 \times 400 \times 64$
12	*Dropout*	Taxa de desativação: 50%	$300 \times 400 \times 64$
13	Convolucional	Número de filtros: 35 Janela de convolução: 1×1 Função de ativação: *softmax*	$300 \times 400 \times 35$

ativações aleatórias. Após, três camadas convolucionais com 128 filtros são utilizadas, a fim de extrair parâmetros de mais alto nível. Outra camada de *pooling*, com as mesmas configurações da primeira, é utilizada para reduzir a dimensionalidade da imagem e deixar a rede sensível a ruídos leves.

A seguir, outra camada de *dropout* é utilizada, com a mesma taxa de desativação da primeira. Estas camadas buscam diminuir o sobreajuste[3] na rede. Após, duas camadas convolucionais com 256 *kernels* e janela 1×1 são empregadas. Assim como na primeira arquitetura, essa é a última camada que exerce um papel similar à codificação de um *autoencoder*, e nesse ponto a imagem está com a menor representação durante a sua propagação na rede.

Por fim, uma camada de *upsampling* de fator 2×2 é utilizada, seguida por uma camada convolucional de 64 filtros de tamanho 1×1. Após, outra camada *upsampling* com as mesmas configurações da primeira aumenta a imagem.

A última camada dessa rede é do tipo convolucional, mas diferente das demais, essa camada tenta fazer uma operação transposta à convolução utilizada pelo resto da rede.

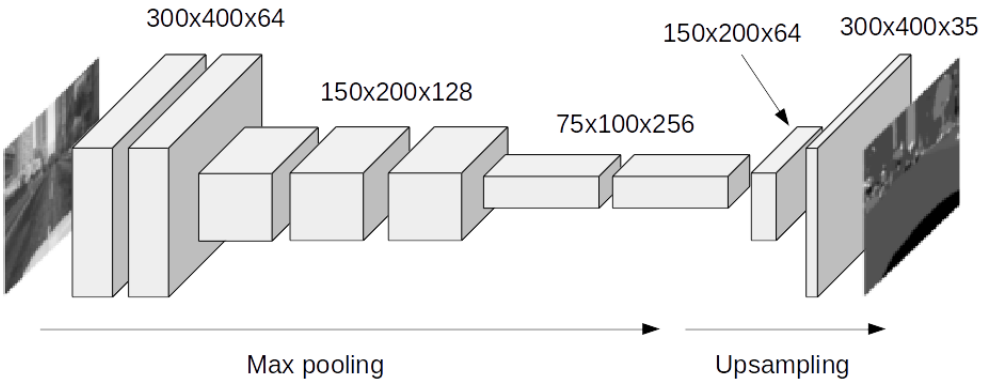

Figura 3.5 - Arquitetura de rede FCN.

[3]sobreajuste (*overfitting*) é um termo estatístico utilizado para indicar que um modelo se adequa bem ao conjunto de dados previamente observados, mas que não é eficaz para prever resultados fora do conjunto de treinamento

Tabela 3.3 - Arquitetura 2

Índice	Tipo	Parâmetros	Formato de saída
1	Convolucional	Número de filtros: 64 Janela de convolução: 5×5 Função de ativação: ReLU	$300 \times 400 \times 64$
2	Convolucional	Número de filtros: 64 Janela de convolução: 5×5 Função de ativação: ReLU	$300 \times 400 \times 64$
3	*Pooling*	Estratégia: valor máximo Janela: 2×2 Passo: 2×2	$150 \times 200 \times 64$
4	*Dropout*	Taxa de desativação: 25%	$150 \times 200 \times 64$
5	Convolucional	Número de filtros: 128 Janela de convolução: 5×5 Função de ativação: ReLU	$150 \times 200 \times 128$
6	Convolucional	Número de filtros: 128 Janela de convolução: 5×5 Função de ativação: ReLU	$150 \times 200 \times 128$
7	Convolucional	Número de filtros: 128 Janela de convolução: 5×5 Função de ativação: ReLU	$150 \times 200 \times 128$
8	*Pooling*	Estratégia: valor máximo Janela: 2×2 Passo: 2×2	$75 \times 100 \times 128$
9	*Dropout*	Taxa de desativação: 25%	$75 \times 100 \times 128$
10	Convolucional	Número de filtros: 256 Janela de convolução: 1×1 Função de ativação: ReLU	$75 \times 100 \times 256$
11	Convolucional	Número de filtros: 256 Janela de convolução: 1×1 Função de ativação: ReLU	$75 \times 100 \times 256$
12	*Upsampling*	Fator de crescimento: 2×2	$150 \times 200 \times 256$
13	Convolucional	Número de filtros: 64 Janela de convolução: 1×1 Função de ativação: ReLU	$150 \times 200 \times 64$
14	*Upsampling*	Fator de crescimento: 2×2	$300 \times 400 \times 64$
15	Convolucional Transposta	Número de filtros: 35 Janela de convolução: 5×5 Função de ativação: *softmax* Fator de dilatação: 8×8	$300 \times 400 \times 35$

3.4 Parâmetros de validação

A taxa de erro é uma medida natural e bastante utilizada de desempenho de classificação. No entanto, essa medida não diferencia erros oriundos de exemplos positivos dos encontrados em exemplos negativos. Uma solução para tal problema é a utilização de uma matriz de confusão, a qual, contabiliza os acertos e os erros gerados pela hipótese avaliada. Em outras palavras, a classe de interesse é definida como positiva, ao mesmo tempo que, as demais são definidas como negativas (BRETT., 2013).

Sendo n_{ii} o número de pixels da classe i preditos corretamente nesta classe, e t_i o número total de pixels da classe i, a acurácia global dos pixels (*global pixel accuracy* - GPA) é calculada por:

$$\frac{\sum_i n_{ii}}{\sum_i t_i} \tag{3.1}$$

O GPA representa a taxa de pixels corretamente classificados na imagem.

Já a intersecção sobre a união média é expressa por:

$$(\frac{1}{n_{cl}}) \sum_i \frac{n_{ii}}{(t_i + \sum_j n_{ji} - n_{ii})} \tag{3.2}$$

Onde n_{cl} é o número total de classes reconhecidas pela rede e n_{ji} representa o número de pixels da classe j classificados erroneamente como classe i.

A acurácia global dos pixels prioriza a acurácia da classificação sobre a segmentação, pois ele se detém a informar apenas quantos pixels foram corretamente rotulados. A segmentação influencia no seu valor, pois se o pixel está errado, a região estará errada, mas esse não é o foco dessa métrica. Para validar a segmentação, a intersecção sobre a união média é mais indicada, pois ela compara as áreas do *ground truth* com as retornadas pela rede.

CAPÍTULO 4

Resultados e Discussão

Nesse capítulo serão apresentados os resultados obtidos através da aplicação de uma rede neural *fully convolutional* para rotular e segmentar imagens de cenas de exteriores. Para tanto, serão testadas duas diferentes arquiteturas de rede, a fim de se determinar a arquitetura com melhores resultados.

4.1 Arquitetura 1

As arquiteturas implementadas foram previamente descritas na seção 3.3. No entanto, vale a pena relembrar os parâmetros utilizados nas mesmas. A primeira arquitetura a ser testada contou com a configuração apresentada na tabela 4.1. Foram utilizadas 13 camadas, sendo 7 camadas para o processo de *encoding* e 6 camadas para *decoding*. Resumidamente, essa rede recebe uma imagem em tons de cinza, em resolução $300x400$, classificando pixel a pixel e segmentando a mesma.

Tabela 4.1 - Parâmetros da primeira arquitetura.

Camada	Parâmetros	Formato de saída
Convolucional	Filtros: $5 \times 5 \times 32$	$300 \times 400 \times 32$
Convolucional	Filtros: $5 \times 5 \times 32$	$300 \times 400 \times 32$
Max Pooling	Janela e passo: 2×2	$150 \times 200 \times 32$
Convolucional	Filtros: $5 \times 5 \times 64$	$150 \times 200 \times 64$
Convolucional	Filtros: $5 \times 5 \times 64$	$150 \times 200 \times 64$
Max Pooling	Janela e passo: 2×2	$75 \times 100 \times 64$
Convolucional	Filtros: $3 \times 3 \times 128$	$75 \times 100 \times 128$
Upsampling	Fator de crescimento: 2×2	$150 \times 200 \times 128$
Dropout	Taxa de desativação: 50%	$150 \times 200 \times 128$
Convolucional	Filtros: $1 \times 1 \times 64$	$150 \times 200 \times 64$
Upsampling	Fator de crescimento: 2×2	$300 \times 400 \times 64$
Dropout	Taxa de desativação: 50%	$300 \times 400 \times 64$
Convolucional	Filtros: $1 \times 1 \times 35$	$300 \times 400 \times 35$

A tabela 4.2 retrata os parâmetros utilizados no treinamento das redes. Foram utilizadas 300 imagens do *dataset Playing for Data*, adaptadas conforme discutido na seção 3.1. Para a fase de validação foram utilizadas 2.500 imagens, a fim de ratificar a capacidade de generalização da rede. O treinamento foi composto por 200 épocas,

de modo a comprovar a convergência da rede. A cada iteração de treinamento, o conjunto de imagens era apresentado em uma ordem diferente. Essa é uma prática sugerida, discutida na seção 2.3.1, buscando-se melhores resultados. Outra boa prática é a de atualizar os pesos para lotes (*batches*) de padrões, e neste trabalho foram adotados lotes de tamanho 2.

Tabela 4.2 - Parâmetros de treinamento das redes neurais.

Parâmetro	Valor
Conjunto de treinamento	Imagens de índice 1 a 300
Conjunto de validação	Imagens de índice 1 a 2500
Função de perda	*Categorical crossentropy*
Número de épocas do treinamento	200
Tamanho do batch	2

Além disso, foram utilizados três diferentes tipos de algoritmos otimizadores: Adam, Adamax e Nadam. Esses otimizadores ajustam a taxa de aprendizado em tempo de execução, visando uma melhor convergência ou evitar *overfitting* (RUDER, 2016).

Adam

Adam (*Adaptive Moment Estimation*) é um método de otimização que pode ser usado no lugar do algoritmo clássico de gradiente descendente estocástico (SGD - *Stochastic Gradiente Descent*) para atualizar pesos de redes neurais baseado em dados de treinamento. Esse método apresenta como principais vantagens a eficiência computacional, pouco requerimento de memória, boa adaptação para problemas ruidosos ou com gradientes esparsos, entre outras.

Diferentemente do gradiente descendente estocástico, que mantém uma única taxa de aprendizado invariável para todos os pesos, o Adam utiliza taxas de aprendizado para cada parâmetro, que são adaptadas separadamente durante o treinamento.

Segundo Kingma e Ba (2014), este algoritmo combina as vantagens de dois outros métodos baseados no SGD:

- AdaGrad (DUCHI JOHN; SINGER, 2011): mantém uma taxa de aprendizado por parâmetro que aprimora os resultados em problemas com gradientes esparsos (como processamento de linguagem natural e visão computacio-

nal);

- RMSProp (TIELEMAN; HINTON, 1980): também mantém uma taxa por parâmetro, que são adaptadas baseado na média das magnitudes recentes dos gradientes de cada peso, que tende a melhorar os resultados para aplicações online ou problemas ruidosos.

Ao invés de adaptar as taxas de aprendizado baseando-se apenas na média do primeiro momento, como no RMSProp, o Adam também utiliza a média do segundo momento dos gradientes.

O pseudocódigo do Adam é descrito como:

Algoritmo 1: Pseudocódigo do Adam

Entradas: α : taxa de aprendizado
$\beta_1, \beta_2 \in [0, 1)$: taxas de decaimento exponenciais para estimativas de momento
$f(\theta)$: função objetivo estocástica com parâmetros θ
θ_0 : vetor inicial de parâmetros

$m_0 \leftarrow 0$ (inicialização do vetor de primeiros momentos)
$v_0 \leftarrow 0$ (inicialização do vetor de segundos momentos)
$t \leftarrow 0$ (inicialização do passo de tempo)
enquanto θ_t *não convergir* **faça**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (calcular os gradientes da função objetivo no tempo t)
  $m_t \leftarrow \beta_1 * m_{t-1} + (1 - \beta_1) * g_t$ (atualizar a estimativa do primeiro momento)
  $v_t \leftarrow \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2$ (atualizar a estimativa do segundo momento)
  $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (computar a estimativa do primeiro momento)
  $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (computar a estimativa do segundo momento)
  $\theta_t \leftarrow \theta_{t-1} - \alpha * \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (atualizar parâmetros)
fim
retorne θ_t *(parâmetros resultantes)*

Segundo Kingma e Ba (2014), bons valores iniciais, considerando problemas de aprendizado de máquina, são: $\alpha = 0.001$; $\beta_1 = 0.9$; $\beta_2 = 0.999$ e $\epsilon = 10^{-8}$.

Como m_t e v_t são inicializados como vetores de zeros, observa-se que seus valores tendem a zero, especialmente durante as primeiras iterações, e também quando as taxas de decaimento são baixas (β_1 e β_2 próximas a 1). Esse tendenciamento é corrigido no cálculo de $\hat{m}_t$ e $\hat{v}_t$ (RUDER, 2016).

Adamax

Uma variação do Adam é o Adamax, o qual se baseia na norma do infinito. No Adam, o fator v_t, na regra de atualização, escala o gradiente de forma inversamente proporcional a norma ℓ_2 dos gradientes passados (termo v_{t-1}) e do corrente ($\mid g_t \mid^2$). É possível generalizar a regra de atualização baseada na norma ℓ^2 para uma regra de atualização baseando na norma ℓ^p:

$$v_t = \beta_2^p v_{t-1} + (1 - \beta_2^p) \mid g_t \mid^p \tag{4.1}$$

Segundo Ruder (2016), em geral, valores altos para p tendem a resultar em uma instabilidade numérica, sendo essa a razão para as normas ℓ_1 e ℓ_2 serem mais utilizadas na prática. No entanto, quando ℓ_∞ é utilizado, essa instabilidade tende a ser evitada. Dessa forma, os autores do Adam propuseram o Adamax (KINGMA; BA, 2014), provando que v_t com ℓ_∞ é capaz de convergir para valores mais estáveis. Para evitar confusão com o Adam, no Adamax, v_t foi substituído por u_t. Dessa forma, a regra de atualização do Adamax é:

$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t}\hat{m}_t \tag{4.2}$$

Com u_t como:

$$\begin{aligned} u_t &= \beta_2^\infty v_{t-1} + (1 - \beta_2^\infty) \mid g_t \mid^\infty && (4.3) \\ &= \max(\beta_2 * v_{t-1}, \mid g_t \mid) && (4.4) \end{aligned}$$

Assim, o pseudocódigo do Adamax é descrito como:

Algoritmo 2: Pseudocódigo do Adamax

Entradas: α : taxa de aprendizado

$\beta_1, \beta_2 \in [0, 1)$: taxas de decaimento exponenciais para estimativas de momento

$f(\theta)$: função objetivo estocástica com parâmetros θ

θ_0 : vetor inicial de parâmetros

$m_0 \leftarrow 0$ (inicialização do vetor de primeiros momentos)

$u_0 \leftarrow 0$ (inicialização do vetor de segundos momentos)

$t \leftarrow 0$ (inicialização do passo de tempo)

enquanto θ_t *não convergir* **faça**

  $t \leftarrow t + 1$

  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (calcular os gradientes da função objetivo no tempo t)

  $m_t \leftarrow \beta_1 * m_{t-1} + (1 - \beta_1) * g_t$ (atualizar a estimativa do primeiro momento)

  $u_t \leftarrow \max(\beta_2 * u_{t-1}, \mid g_t \mid)$ (atualizar o peso ponderado exponencialmente pela norma do infinito)

  $\theta_t \leftarrow \theta_{t-1} - (\alpha/(1 - \beta_1^t)) * m_t/u_t$ (atualizar parâmetros)

fim

retorne θ_t *(parâmetros resultantes)*

Ainda de acordo com Kingma e Ba (2014), bons valores adotados para configurações padrões, para problemas de aprendizado de máquina, são: $\alpha = 0.002$; $\beta_1 = 0.9$ e $\beta_2 = 0.999$.

Nadam

Ainda como uma variação do Adam, o Nadam (*Nesterov-accelerated Adaptive Moment Estimation*) utiliza o gradiente acelerado de Nesterov, conhecido como NAG (*Nesterov Accelerated Gradient*), o qual foi proposto em 1983 para solucionar problemas de otimização convexa. Na prática, para incorporar o NAG no Adam é preciso modificar o termo de momento m_t.

A regra de atualização do momento é geralmente descrita como:

$$g_t = \nabla_{\theta_t} J(\theta_t) \tag{4.5}$$

$$m_t = \gamma m_{t-1} + \eta g_t \tag{4.6}$$

$$\theta_{t+1} = \theta_t - m_t \tag{4.7}$$

Sendo J a função objetivo, γ o termo de decaimento do momento e η o tamanho do passo.

Expandindo a equação de θ, tem-se:

$$\theta_{t+1} = \theta_t - (\gamma m_{t-1} + \eta g_t) \tag{4.8}$$

Desse modo, com a aplicação do NAG é possível executar um passo mais preciso na direção do gradiente, atualizando os parâmetros com a medida do momento antes de calcular o gradiente. Para tanto, é necessário somente modificar o gradiente g_t. Uma modificação no NAG, proposta por Dozat, sugere que ao invés de utilizar a medida do momento para atualizar tanto g_t, quanto θ_{t+1}, ela seja aplicada diretamente em θ_{t+1}:

$$\theta_{t+1} = \theta_t - (\gamma m_t + \eta g_t) \tag{4.9}$$

Vale ressaltar que a diferença entre a equação 4.8 e a 4.9 é que agora ao invés de se utilizar o momento anterior (m_{t-1}) é empregado o momento atual (m_t). Assim, de forma prática, para incorporar o NAG no Adam é preciso apenas repassar essa modificação do termo do momento. Relembrando que no Adam a regra de atualização é descrita como:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{4.10}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{4.11}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{4.12}$$

Ampliando a equação de θ com as definições de $\hat{m}_t$ e m_t:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\left(\frac{\beta_1 m_{t-1}}{1-\beta_1^t} + \frac{(1-\beta_1)g_t}{1-\beta_1^t}\right) \tag{4.13}$$

Nesse caso, $\frac{\beta_1 m_{t-1}}{1-\beta_1^t}$ é simplesmente a estimativa corrigida do vetor de momento do passo anterior, podendo ser substituído por $\hat{m_{t-1}}$:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\left(\beta_1 \hat{m_{t-1}} + \frac{(1-\beta_1)g_t}{1-\beta_1^t}\right) \tag{4.14}$$

Agora é possível, assim como anteriormente, trocar o cálculo do momento anterior ($\hat{m_{t-1}}$) pelo momento atual ($\hat{m}_t$), tendo assim, a regra de atualização do Nadam:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\left(\beta_1 \hat{m}_t + \frac{(1-\beta_1)g_t}{1-\beta_1^t}\right) \tag{4.15}$$

Resultados obtidos

Após a aplicação dos algoritmos otimizadores, o próximo passo foi analisar a convergência desses métodos. Para isso, as duas arquiteturas foram treinadas com os três algoritmos, e foram registrados o GPA médio e a saída da função de perda a cada iteração. Esse passo permitiu a análise do processo de convergência dos diferentes métodos.

A figura 4.1 apresenta o gráfico da saída da função de otimização para os três métodos de treinamento estudados. Para esta arquitetura, o Nadam apresentou um resultado levemente inferior aos demais, enquanto que o Adam e o Adamax exibiram resultados similares, com o algoritmo Adam produzindo maiores instabilidades durante o treinamento. Tal comportamento pode ser explicado pelas características do Adamax, como comentado anteriormente.

Em relação ao GPA médio, métrica que indica a porcentagem de pixels corretamente classificados para todas as imagens do treinamento, a figura 4.2 ilustra sua evolução ao longo das 200 iterações de treinamento. Assim como na análise da função de perda, o algoritmo Adamax se mostrou mais estável, e com uma acurácia comparável ao Adam. O método Nadam novamente se mostrou levemente inferior para esse

problema específico.

Esses dados foram calculados utilizando apenas o conjunto de treinamento, para fins de verificação da convergência dos métodos. Uma validação posterior, e mais detalhada, foi aplicada com os pesos finais da rede para cada método de treinamento, otimizando assim o esforço computacional e temporal para treinar e validar as arquiteturas.

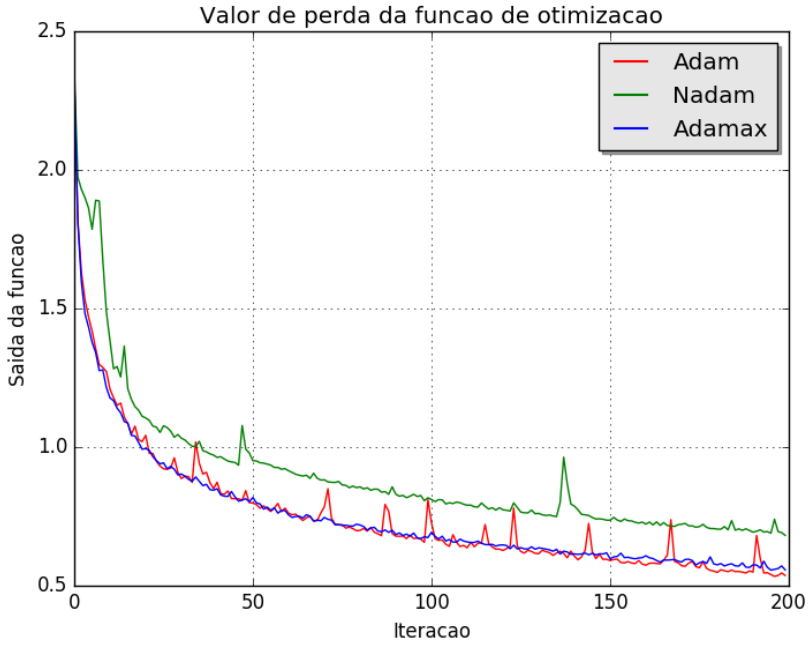

Figura 4.1 - Convergência da saída da função de perda - arquitetura 1.

Esta análise detalhada das redes e dos métodos utilizaram 2.500 imagens, tendo suas estatísticas apresentadas na tabela 4.3. É possível observar que o comportamento dos três algoritmos de treinamento foram similares, tanto para o GPA quanto para o IoU médio, e o algoritmo Adamax apresentou um resultado ligeiramente melhor que os outros métodos. Acredita-se que o IoU médio apresentou um resultado um pouco mais baixo quando comparado ao GPA pois é uma estatística que leva em consideração todas as classes presentes na imagem, e todas possuem o mesmo peso no valor final. Mesmo assim, por ser um parâmetro de validação que reflete informações sobre falsos positivos e verdadeiros negativos presentes na imagem, seus valores indicam as informações de classes e regiões que estão presentes na saída da rede e

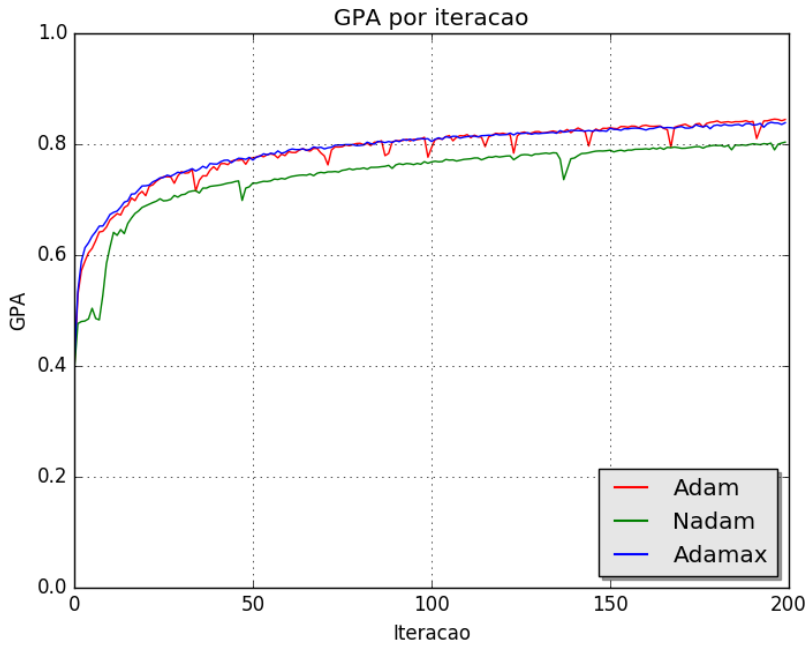

Figura 4.2 - GPA médio do treinamento - arquitetura 1.

não deveriam estar.

Tabela 4.3 - Estatísticas de validação da arquitetura 1 para 2.500 imagens.

	Adam	Adamax	Nadam
GPA			
Valor máximo	0.957	0.954	0.929
Média	0.771	0.784	0.743
Mediana	0.783	0.798	0.756
Desvio padrão	0.086	0.084	0.088
IoU médio			
Valor máximo	0.353	0.361	0.352
Média	0.199	0.215	0.186
Mediana	0.197	0.216	0.186
Desvio padrão	0.044	0.043	0.040

Para ilustrar saídas da rede, seis imagens foram utilizadas, separadas em duas imagens bem avaliadas (figura 4.3), duas com desempenho mediano (figura 4.4) e duas abaixo da média (figura 4.5). Essas duas imagens de cada caso foram escolhidas

com base nos parâmetros de validação utilizados neste trabalho, sendo que as figuras à esquerda representam a avaliação em relação à média do GPA, e à direita as avaliadas em relação à média do IoU médio.

A figura 4.6 ilustra as saídas acima da média dos algoritmos Adam, Adamax e Nadam, respectivamente. Para este caso, foi escolhida uma imagem com bons resultados de GPA em relação ao conjunto de validação (à esquerda), e uma figura com bons IoU médio em relação ao conjunto de validação (à direita). A rede obteve resultados significativos para a segmentação e classificação do asfalto e do céu. Além disso, os contornos dos objetos comumente são mantidos, mesmo que não sejam classificados corretamente.

Por outro lado, a presença de segmentações excessivas de um único objeto foi observada com frequência. Tal característica é comumente observada em classes com certa ambiguidade, como por exemplo caminhões e veículos de passeio. Estas classes possuem algumas informações visuais parecidas, como pneus, vidros, entre outros. Tais resultados evidenciam a influência do contexto na correta classificação de uma imagem.

Na figura 4.7 são apresentadas saídas da rede com parâmetros de validação medianos. É possível observar que, para estas imagens, a classificação do asfalto e do céu permanecem satisfatórias, no entanto a segmentação excessiva é mais presente, sendo um dos principais fatores da queda dos índices de validação. Tal problema torna-se ainda mais evidente no conjunto de imagens abaixo da média (figura 4.8). Além disso, a importância do contexto torna-se ainda mais visível. Em contrapartida, mesmo com uma leve supersegmentação, o asfalto e o céu ainda podem ser facilmente distinguidos.

Apesar dos resultados ainda necessitarem de refinamentos, a arquitetura foi promissora devido à sua relativa simplicidade, o que reduz o poder computacional requerido pela mesma, uma boa característica para o uso da mesma em sistemas embarcados.

Visando preservar um custo-benefício favorável da rede e aprimorar os resultados, principalmente em relação à alta pixelização da imagem de saída, a segunda arquitetura foi desenvolvida e validada, e seus resultados são apresentados a seguir.

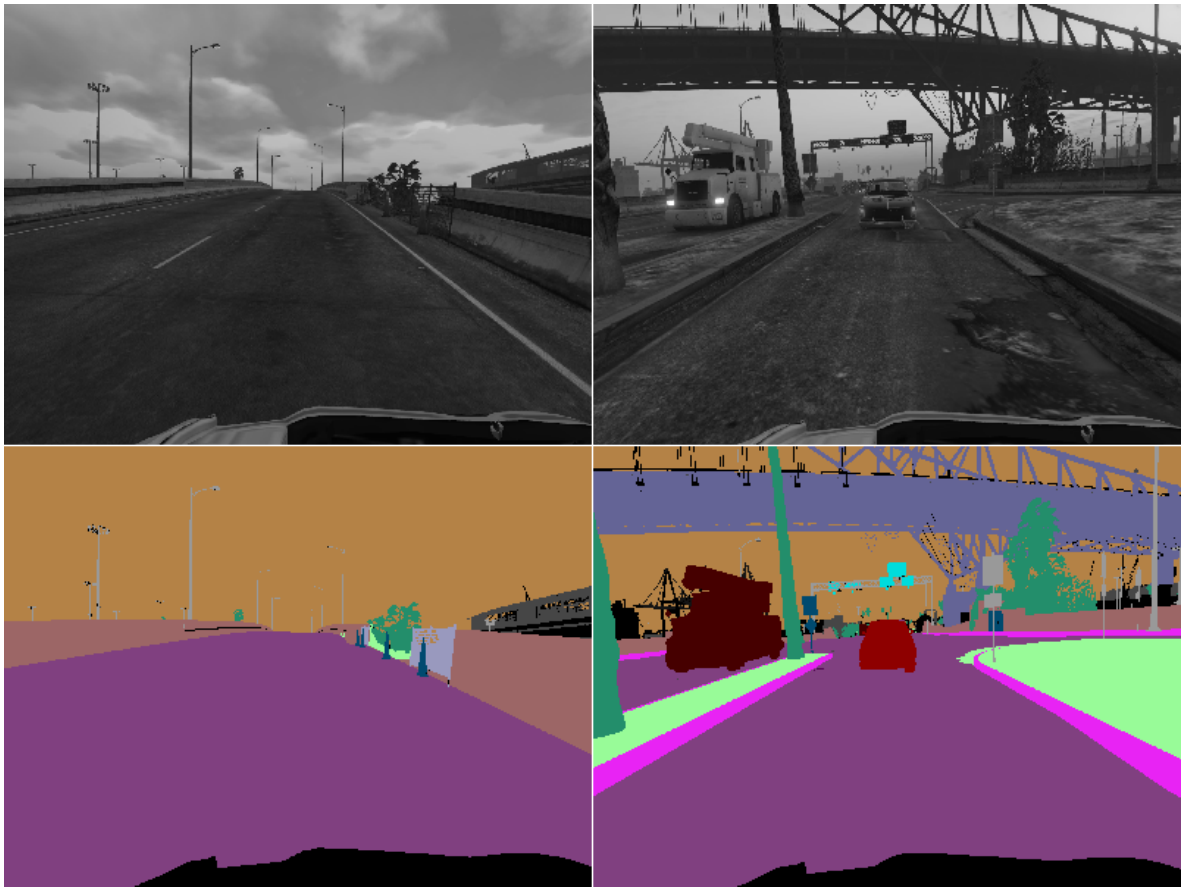

Figura 4.3 - Conjunto de entradas e saídas esperadas para o sistema. Imagens com boas segmentações e classificações em ambas arquiteturas.

4.2 Arquitetura 2

A segunda arquitetura foi implementada adotando os mesmos parâmetros de treinamento da arquitetura 1, apresentados na tabela 4.2. Esta rede possui mais pesos ajustáveis que a primeira, visando um aumento nos valores de acurácia da rede. Esta FCN possui 15 camadas, sendo 11 para o processo de *encoding* e 4 para o *decoding*, e recebe como entrada uma imagem em dimensões $300x400$ em tons de cinza. O objetivo desta rede também é classificar pixel a pixel e rotular a imagem de entrada. Sua arquitetura resumida está descrita em 4.4.

Esta rede também foi treinada com os algoritmos Adam, Adamax e Nadam. Os parâmetros utilizados para o treinamento foram os mesmos da arquitetura anterior e são descritos na tabela 4.5.

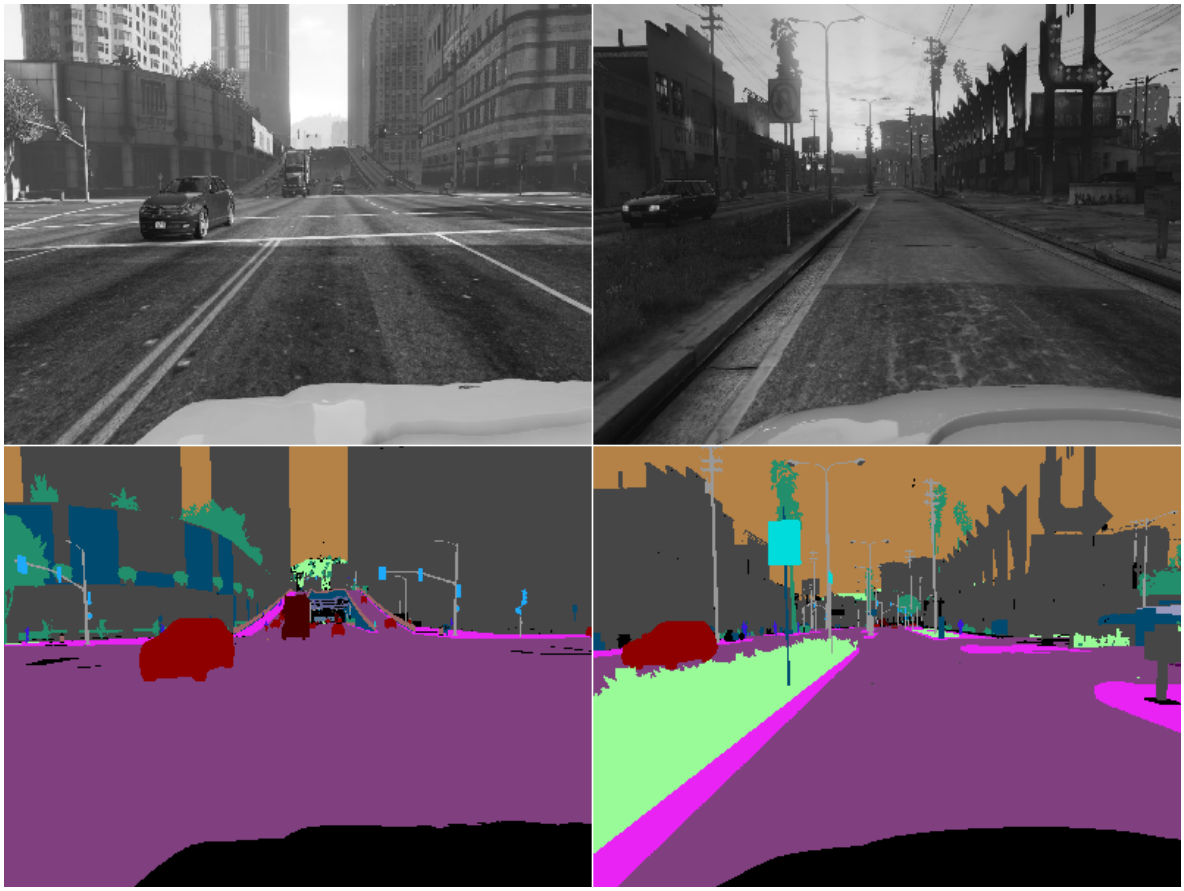

Figura 4.4 - Conjunto de entradas e saídas esperadas para o sistema. Imagens com rotulação e segmentação medianas.

Assim como na primeira arquitetura, a saída da função de otimização e o GPA médio do conjunto de treinamento foram calculados a cada iteração do treinamento, a fim de verificar a convergência do sistema. A figura 4.9 demonstra o comportamento da função de perda para os três algoritmos.

Para essa arquitetura específica, novamente o método Adamax se mostrou mais estável e com melhores resultados. No entanto, diferentemente da primeira rede, o algoritmo Adam não se comportou de forma similar ao Adamax..

Em relação ao GPA médio (figura 4.10), o Adamax também apresentou melhores resultados, inclusive quando comparados à primeira arquitetura, de x na primeira rede para y na segunda. Acredita-se que esse ganho de precisão é devido ao aumento de parâmetros ajustáveis na rede, possibilitando o aprendizado de mais padrões. Em contrapartida, o esforço (tempo) necessário pra treinar essa segunda arquitetura,

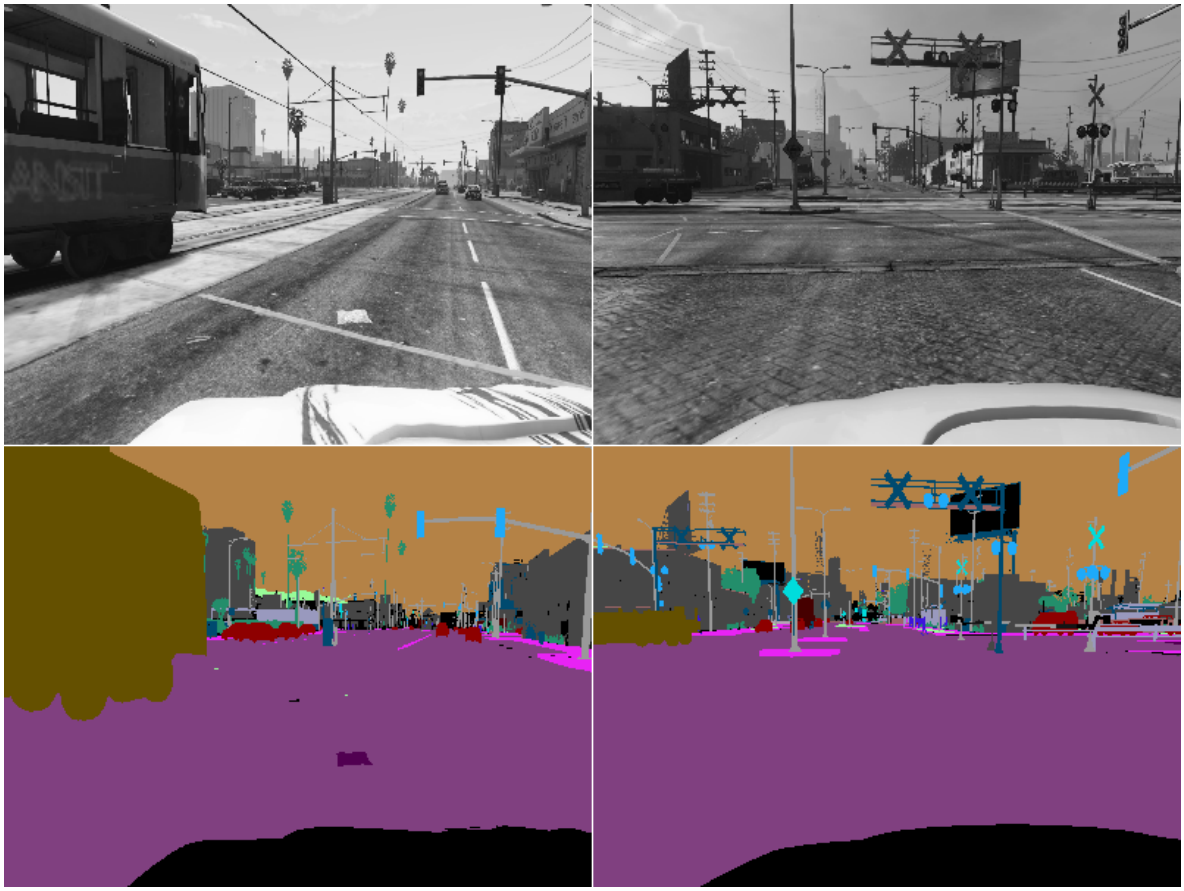

Figura 4.5 - Conjunto de entradas e saídas esperadas para o sistema. Imagens com classificação e segmentação abaixo da média.

utilizando um computador pessoal com uma GPU GTX1050, foi cerca de 50% maior.

Esta análise também utilizou as mesmas 2.500 imagens da primeira, e as estatísticas da validação são apresentadas na tabela 4.6. Novamente os três algoritmos de treinamento tiveram comportamento similar, tanto para o GPA quanto para o IoU médio. Quando comparados aos resultados da arquitetura 1, uma ligeira melhora pode ser observada em todas as estatísticas.

O conjunto de imagens que ilustram a saída da rede 2 também é oriundo das figuras 4.3, 4.4 e 4.5. A imagem 4.11 ilustra, respectivamente, saídas dos algoritmos Adam, Adamax e Nadam acima da média. Assim como na arquitetura 1, os casos à esquerda são relativos ao GPA, e os à direita relativos ao IoU médio. A rede também obteve bons resultados quanto à segmentação do céu e do asfalto, além de uma redução expressiva na pixelização da cena. Uma menor pixelização permite a

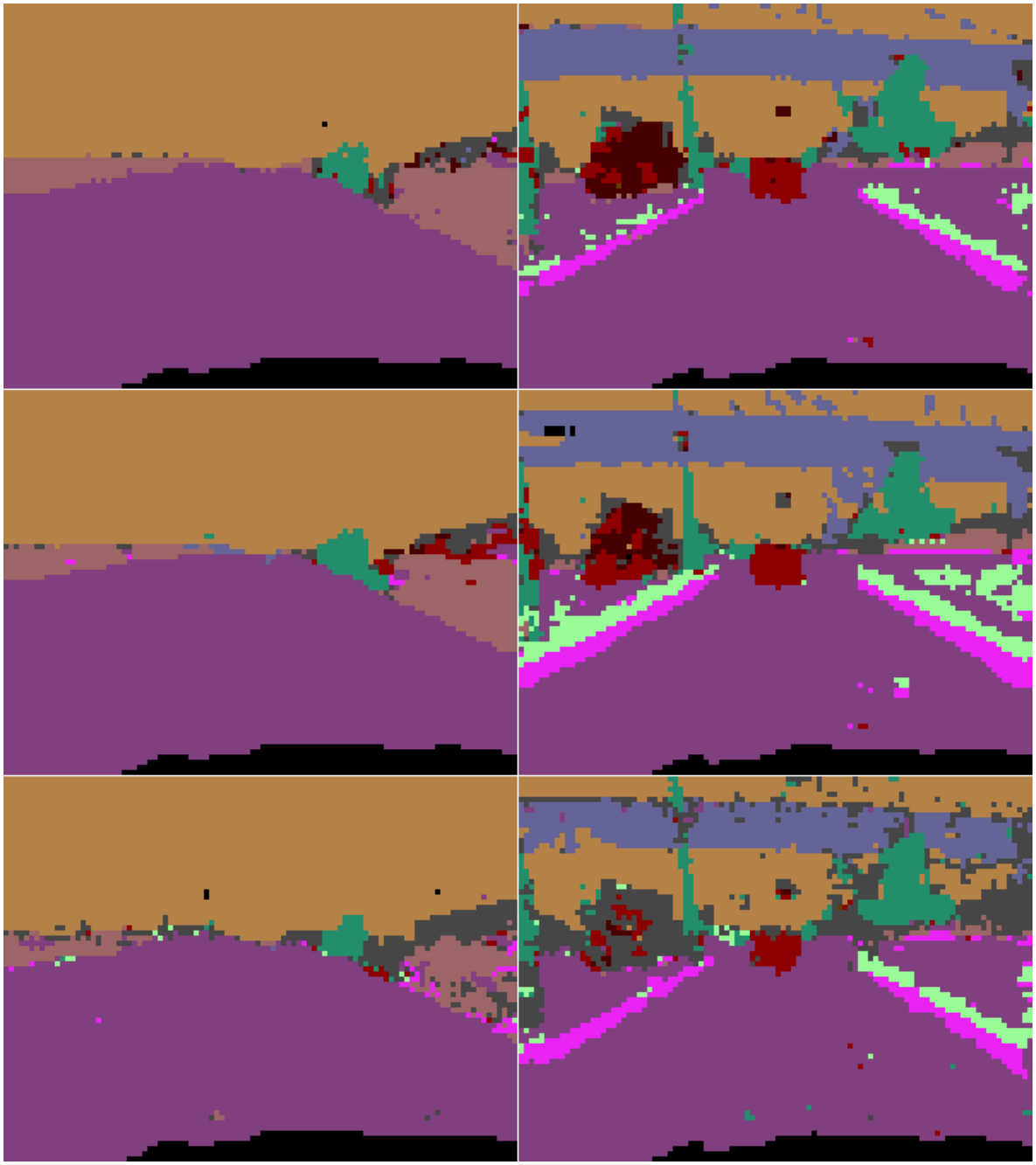

Figura 4.6 - Imagens com parâmetros de validação acima da média, utilizando a primeira arquitetura de rede. Da esquerda pra direita, de cima pra baixo: a) GPA: 0.957 e IoU médio: 0.300; b) GPA: 0.797 e IoU médio: 0.330; c) GPA: 0.954 e IoU médio: 0.298; d) GPA: 0.820 e IoU médio: 0.361; e) GPA: 0.928 e IoU médio: 0.291; f) GPA: 0.727 e IoU médio: 0.269. As imagens (a) e (b) são saídas da rede treinada com o algoritmo Adam, (c) e (d) do Adamax e (e) e (f) do Nadam.

rede lidar melhor com objetos considerados pequenos na cena, como por exemplo a placa de trânsito acima do carro, na cena à direita, que foi corretamente classificada

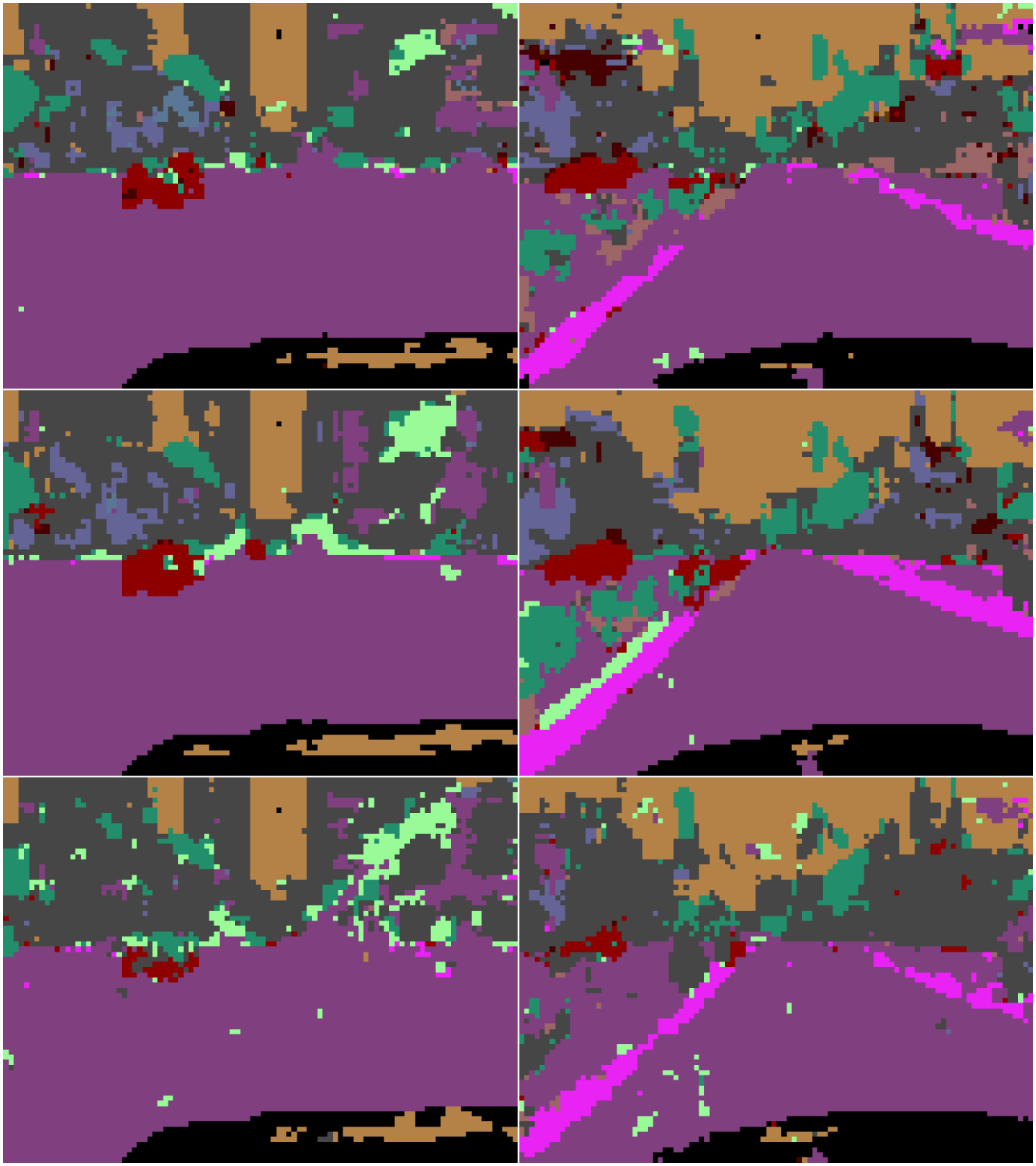

Figura 4.7 - Imagens com parâmetros de validação medianos, utilizando a primeira arquitetura de rede. Da esquerda pra direita, de cima pra baixo: a) GPA: 0.820 e IoU médio: 0.196; b) GPA: 0.698 e IoU médio: 0.186; c) GPA: 0.798 e IoU médio: 0.190; d) GPA: 0.740 e IoU médio: 0.216; e) GPA: 0.792 e IoU médio: 0.170; f) GPA: 0.735 e IoU médio: 0.195. As imagens (a) e (b) são saídas da rede treinada com o algoritmo Adam, (c) e (d) do Adamax e (e) e (f) do Nadam.

na segunda arquitetura. Também é possível notar, para estes casos específicos, uma menor supersegmentação da saída, apesar da mesma não ter sido extinta. Este fato

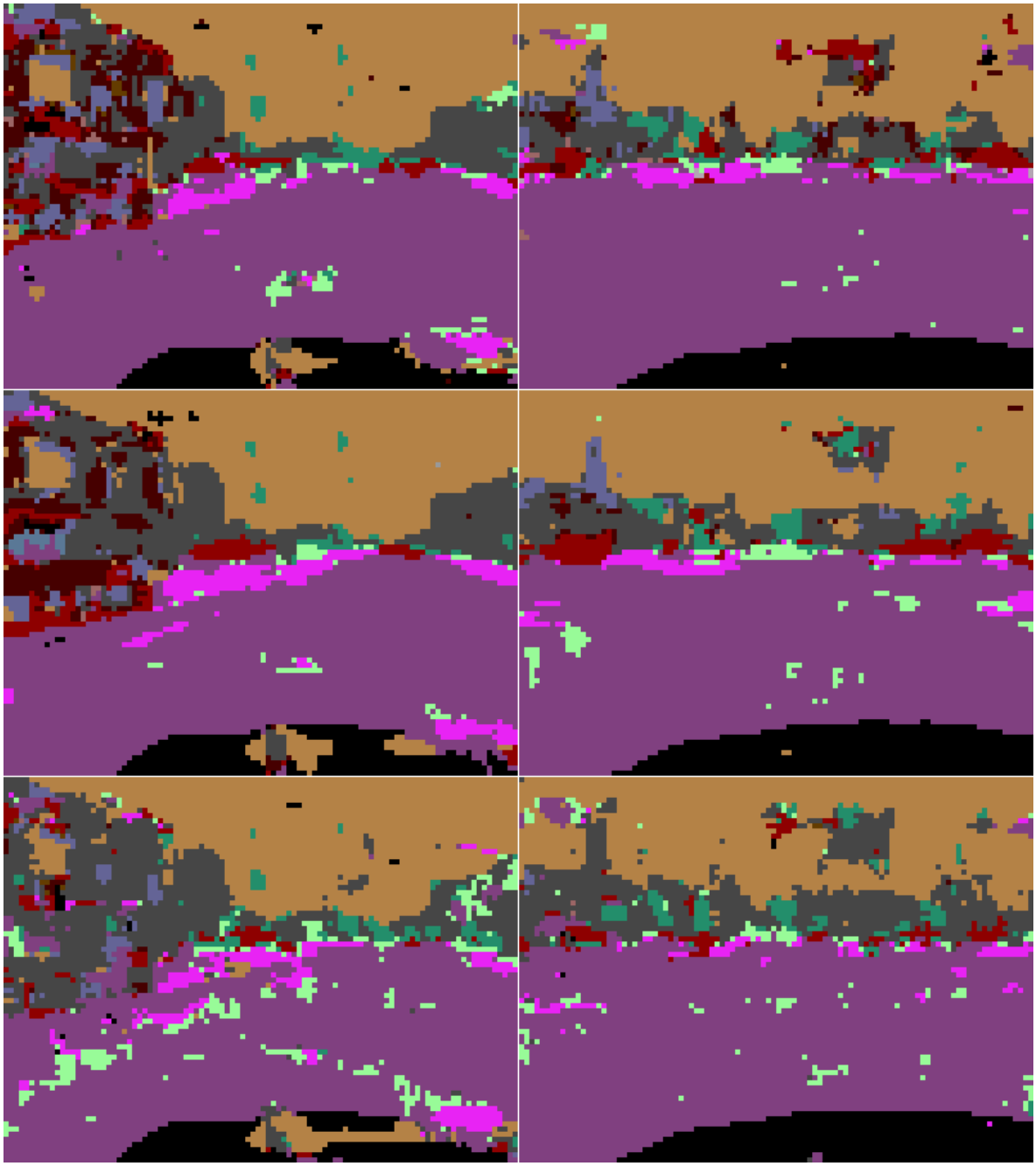

Figura 4.8 - Imagens com parâmetros de validação abaixo da média, utilizando a primeira arquitetura de rede. Da esquerda pra direita, de cima pra baixo: a) GPA: 0.690 e IoU médio: 0.129; b) GPA: 0.822 e IoU médio: 0.146; c) GPA: 0.686 e IoU médio: 0.134; d) GPA: 0.830 e IoU médio: 0.161; e) GPA: 0.639 e IoU médio: 0.115; f) GPA: 0.817 e IoU médio: 0.142. As imagens (a) e (b) são saídas da rede treinada com o algoritmo Adam, (c) e (d) do Adamax e (e) e (f) do Nadam.

influenciou, por exemplo, numa melhor classificação do caminhão.

Tabela 4.4 - Parâmetros da segunda arquitetura.

Camada	Parâmetros	Formato de saída
Convolucional	Filtros: $5 \times 5 \times 64$	$300 \times 400 \times 64$
Convolucional	Filtros: $5 \times 5 \times 64$	$300 \times 400 \times 64$
Max Pooling	Janela e passo: 2×2	$150 \times 200 \times 64$
Dropout	Taxa de desativação: 25%	$150 \times 200 \times 64$
Convolucional	Filtros: $5 \times 5 \times 128$	$150 \times 200 \times 128$
Convolucional	Filtros: $5 \times 5 \times 128$	$150 \times 200 \times 128$
Convolucional	Filtros: $5 \times 5 \times 128$	$150 \times 200 \times 128$
Max Pooling	Janela e passo: 2×2	$75 \times 100 \times 128$
Dropout	Taxa de desativação: 25%	$75 \times 100 \times 128$
Convolucional	Filtros: $1 \times 1 \times 256$	$75 \times 100 \times 256$
Convolucional	Filtros: $1 \times 1 \times 256$	$75 \times 100 \times 256$
Upsampling	Fator de crescimento: 2×2	$150 \times 200 \times 256$
Convolucional	Filtros: $1 \times 1 \times 64$	$150 \times 200 \times 64$
Upsampling	Fator de crescimento: 2×2	$300 \times 400 \times 64$
Convolucional Transposta	Filtros: $5 \times 5 \times 35$	$300 \times 400 \times 35$

Tabela 4.5 - Parâmetros de treinamento da segunda arquitetura.

Parâmetro	Valor
Conjunto de treinamento	Imagens de índice 1 a 300
Conjunto de validação	Imagens de índice 1 a 2500
Função de perda	*Categorical crossentropy*
Número de épocas do treinamento	200
Tamanho do batch	2

Na figura 4.12 são exibidas saídas da rede com parâmetros de validação medianos, e na imagem 4.13 os casos abaixo da média. Observa-se que o recorrente problema de excesso de segmentação também está presente nessas saídas da rede, mas ambos casos de teste possuem uma pixelização bem menos expressiva, e isso implica numa segmentação mais suave da cena.

Com base dos resultados apresentados das arquiteturas 1 e 2, pode-se concluir que ambas são soluções viáveis, e a escolha de qual adotar depende fortemente da aplicação da rede. Para problemas onde é necessário reduzir ao máximo o tamanho da rede, a arquitetura 1 pode ser utilizada sem perdas muito expressivas em relação à outra arquitetura. Em contrapartida, se a pixelização excessiva for um problema, pois há uma necessidade de uma segmentação mais suave ou lidar melhor com objetos de

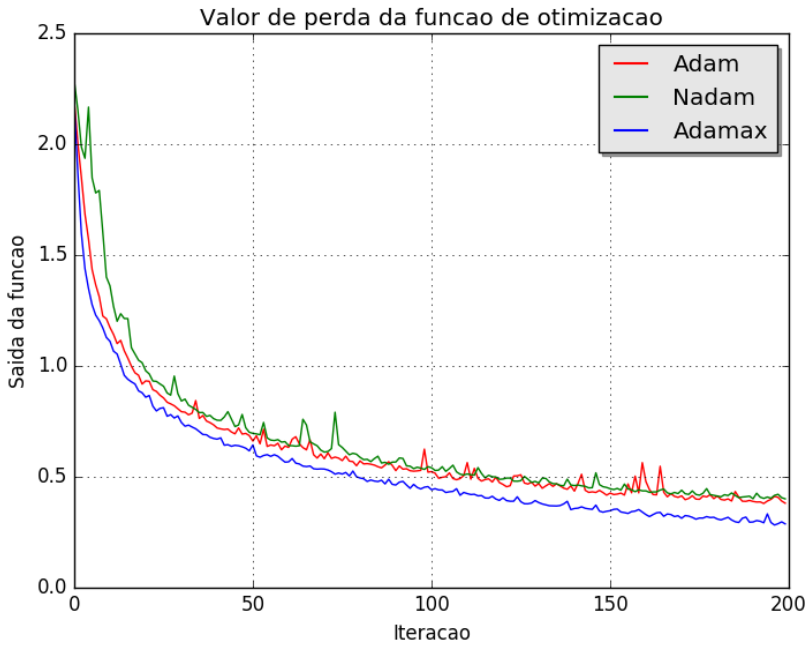

Figura 4.9 - Convergência da saída da função de perda - arquitetura 2.

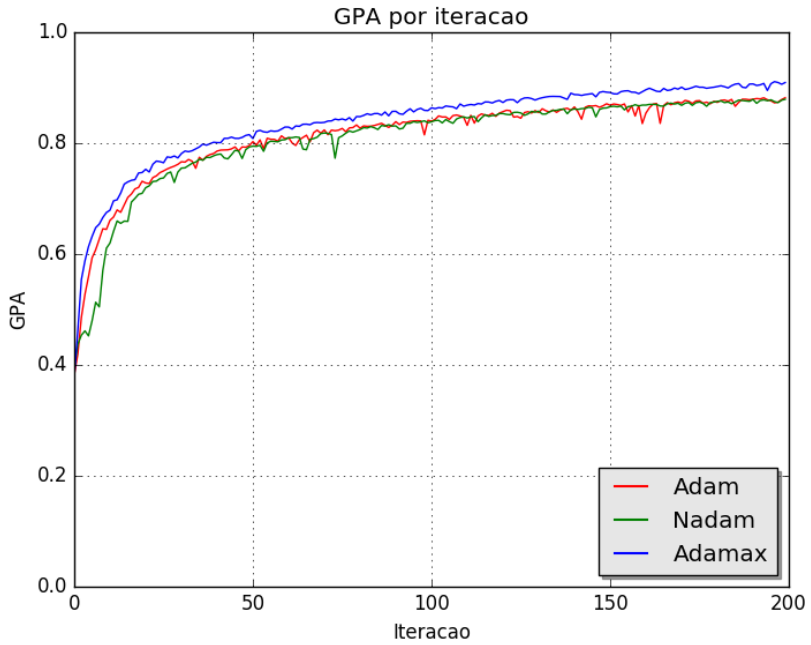

Figura 4.10 - GPA médio do treinamento - arquitetura 2.

Tabela 4.6 - Estatísticas de validação da arquitetura 2 para 2.500 imagens.

	Adam	Adamax	Nadam
GPA			
Valor máximo	0.969	0.979	0.971
Média	0.789	0.805	0.786
Mediana	0.798	0.815	0.797
Desvio padrão	0.089	0.090	0.090
IoU médio			
Valor máximo	0.438	0.620	0.453
Média	0.201	0.217	0.208
Mediana	0.192	0.203	0.201
Desvio padrão	0.056	0.067	0.057

tamanho reduzido, a segunda rede é mais indicada.

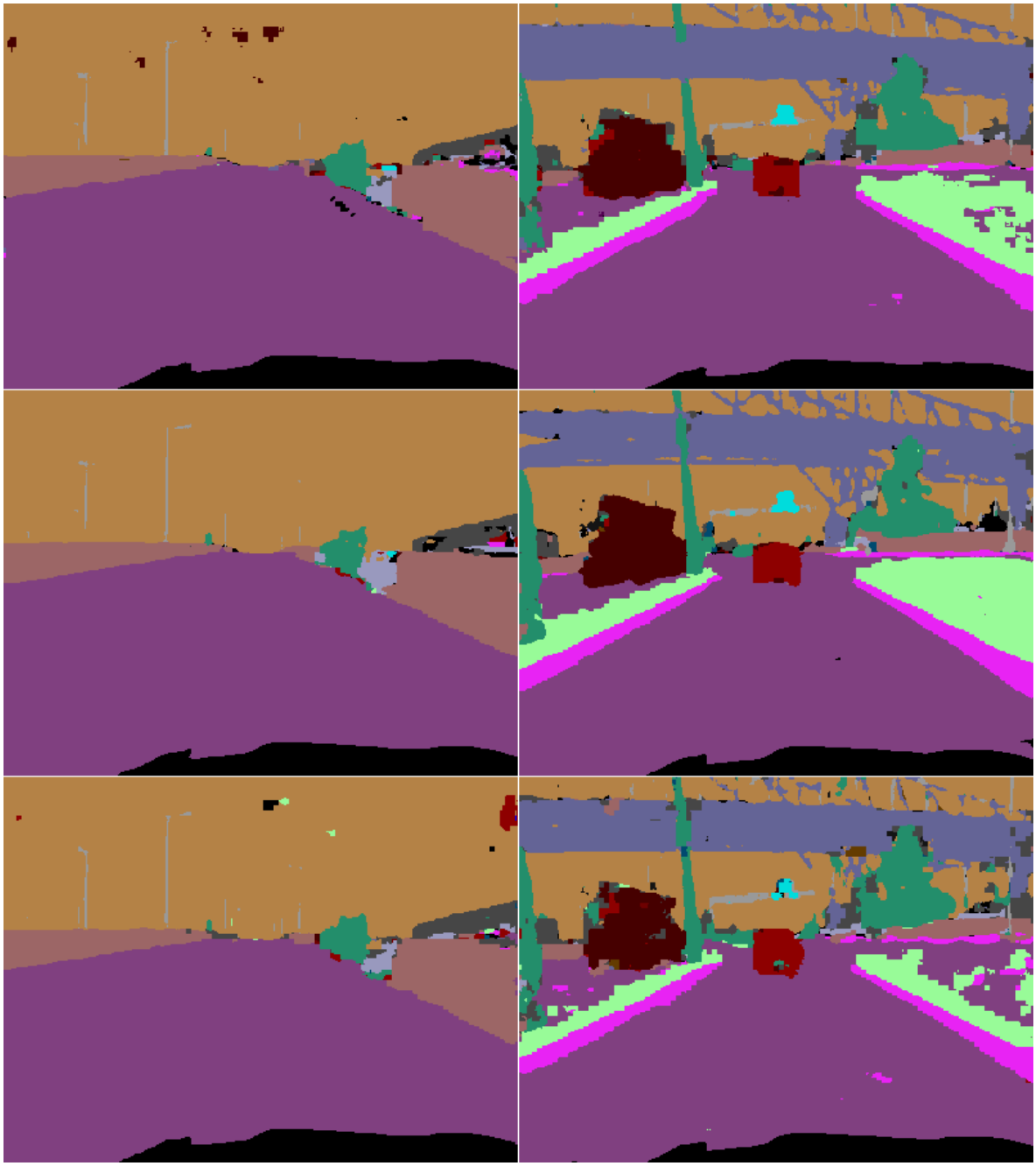

Figura 4.11 - Imagens com parâmetros de validação acima da média, utilizando a segunda arquitetura de rede. Da esquerda pra direita, de cima pra baixo: a) GPA: 0.969 e IoU médio: 0.340; b) GPA: 0.892 e IoU médio: 0.432; c) GPA: 0.979 e IoU médio: 0.417; d) GPA: 0.923 e IoU médio: 0.506; e) GPA: 0.971 e IoU médio: 0.340; f) GPA: 0.854 e IoU médio: 0.407. As imagens (a) e (b) são saídas da rede treinada com o algoritmo Adam, (c) e (d) do Adamax e (e) e (f) do Nadam.

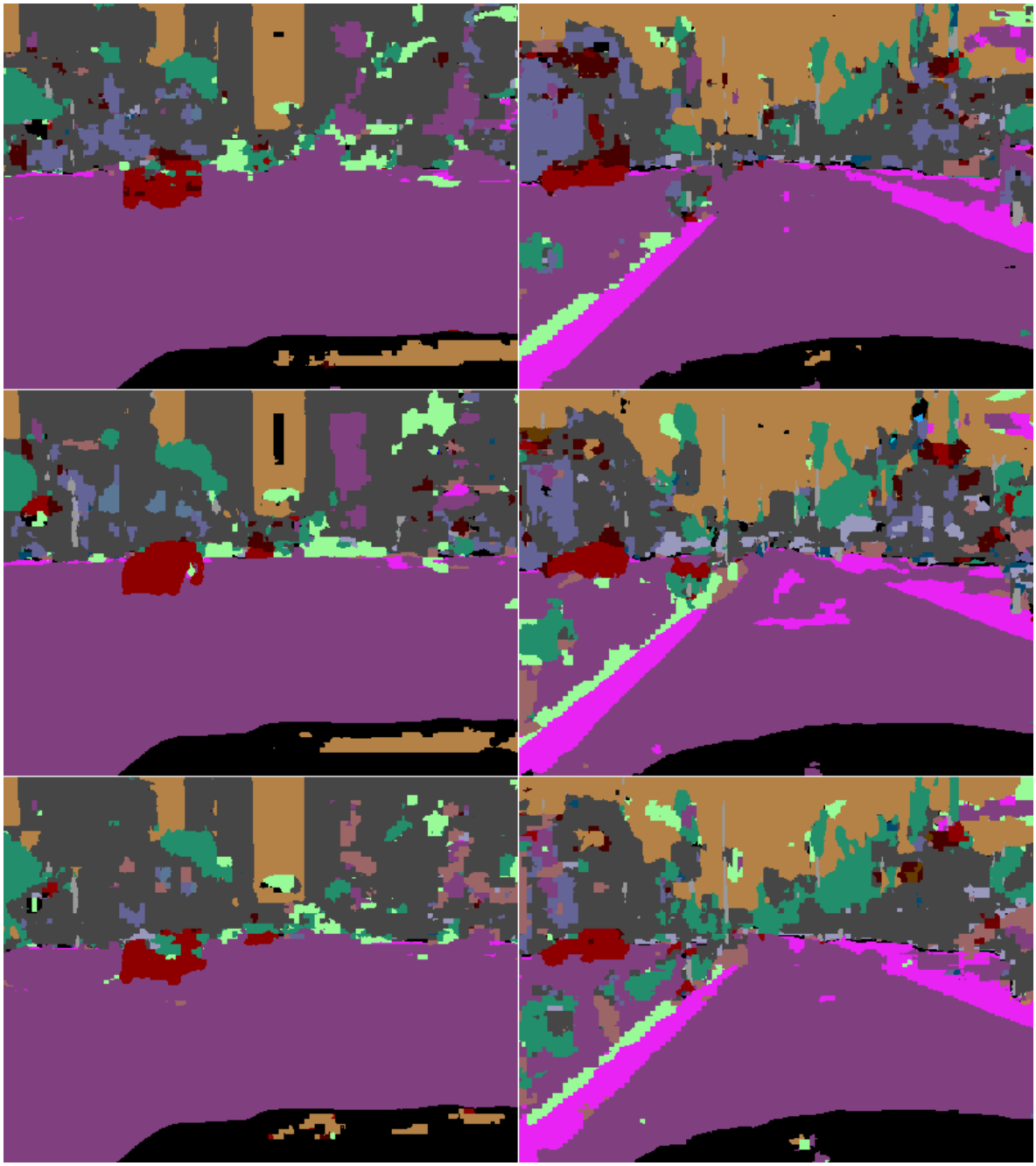

Figura 4.12 - Imagens com parâmetros de validação medianos, utilizando a segunda arquitetura de rede. Da esquerda pra direita, de cima pra baixo: a) GPA: 0.810 e IoU médio: 0.175; b) GPA: 0.699 e IoU médio: 0.190; c) GPA: 0.830 e IoU médio: 0.184; d) GPA: 0.721 e IoU médio: 0.200; e) GPA: 0.851 e IoU médio: 0.203; f) GPA: 0.717 e IoU médio: 0.203. As imagens (a) e (b) são saídas da rede treinada com o algoritmo Adam, (c) e (d) do Adamax e (e) e (f) do Nadam.

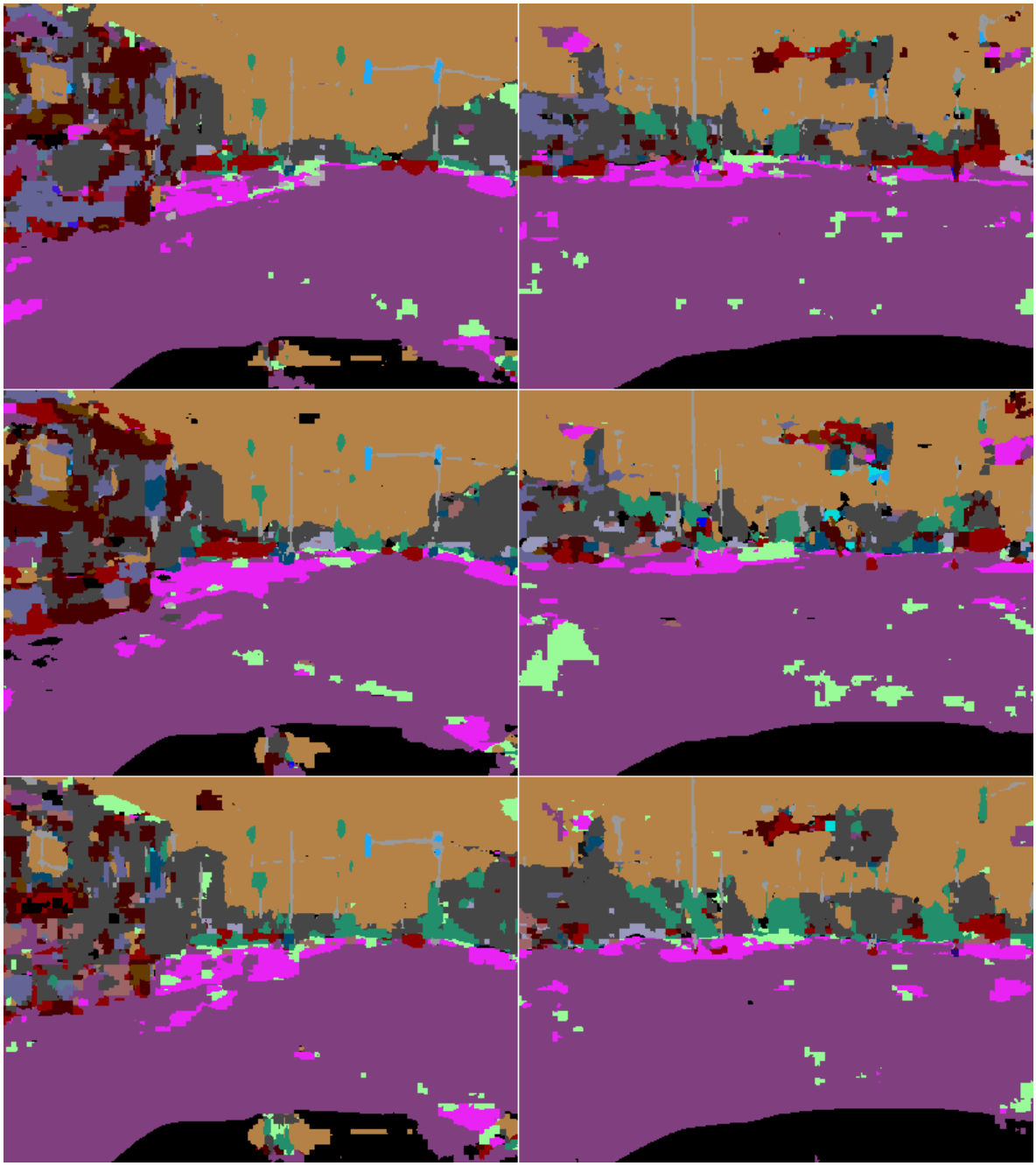

Figura 4.13 - Imagens com parâmetros de validação abaixo da média, utilizando a segunda arquitetura de rede. Da esquerda pra direita, de cima pra baixo: a) GPA: 0.699 e IoU médio: 0.156; b) GPA: 0.825 e IoU médio: 0.148; c) GPA: 0.685 e IoU médio: 0.153; d) GPA: 0.806 e IoU médio: 0.148; e) GPA: 0.678 e IoU médio: 0.147; f) GPA: 0.834 e IoU médio: 0.156. As imagens (a) e (b) são saídas da rede treinada com o algoritmo Adam, (c) e (d) do Adamax e (e) e (f) do Nadam.

CAPÍTULO 5

Conclusão

O presente trabalho apresentou a implementação de uma rede neural convolucional para classificar e segmentar imagens de cenas exteriores. O treinamento utilizou cenas sintéticas de alto detalhamento. O primeiro passo foi a escolha da técnica utilizada. Neste caso as redes convolucionais foram eleitas, devido suas características naturais para lidar com processamento de imagens junto ao aprendizado de máquina.

Com a técnica escolhida, o conjunto de treinamento *Playing for data* foi selecionado e uma parcela dele foi adaptada, visando uma aplicação futura do robô LEIA 1. A adaptação do *dataset* consistiu no recorte e redimensionamento das imagens (para $300x400$), bem como a transformação de RGB para tons de cinza.

Para a implementação das duas arquiteturas de rede validadas, a linguagem de programação Python foi eleita, juntamente com as APIs Theano para o cálculo otimizado em GPU das operações matriciais, e Keras para a prototipagem das arquiteturas de rede.

Após o desenvolvimento, foi necessário a validação das arquiteturas de rede, para permitir a análise de desempenho das mesmas. Para tanto, 2.500 foram utilizadas e foram calculadas estatísticas para os parâmetros GPA (*global pixel accuracy*) e IoU (*intersection over union*) médio. O GPA foi empregado para verificar a capacidade de classificação das redes, e o IoU para avaliar a presença e distribuição das classes presentes na imagem. Além disso, três algoritmos de otimização diferentes foram implementados e comparados (Adam, Adamax e Nadam). Esses três métodos são da mesma família, sendo o Adamax e o Nadam variações do Adam.

Os resultados obtidos para as duas redes, considerando os três algoritmos otimizadores demonstraram a viabilidade do uso de ambas arquiteturas, sendo sua escolha altamente dependente do problema. O melhor resultado de valor médio de GPA e IoU médio foi observado na arquitetura 2, com o otimizador de treinamento Adamax (média do GPA: 0.805 e média do IoU médio: 0.217). Em relação à primeira rede, o ganho de GPA médio com o algoritmo Adamax foi de 0.021. Esta diferença representa, em média, 2.520 *pixels* a mais corretamente classificados, por imagem, pela arquitetura 2.

Vale ressaltar que na segunda arquitetura foi observado um maior esforço computacional requerido, tanto ao longo do treinamento quanto da utilização da rede. Em relação ao treinamento, observou-se um aumento de cerca de 50% no tempo requerido. O aumento mais significativo em relação ao uso está na quantidade de memória requerida pela segunda rede.

Dessa forma, conclui-se que a escolha, ou mesmo adaptação da arquitetura a ser utilizada depende da aplicação a qual a rede será submetida. Para aplicações que exigem um menor custo computacional, em detrimento de uma maior acurácia e menor pixelização, a primeira arquitetura pode ser utilizada. Caso uma melhor segmentação seja um fator determinante, ou haja poder computacional suficiente disponível, a arquitetura 2 torna-se mais recomendada.

Como trabalhos futuros recomenda-se:

- aplicação de algoritmos de pós-processamento na imagem, visando reduzir a supersegmentação da mesma;
- implementação da arquitetura 2 no robô LEIA 1;
- utilização de uma câmera estéreo de maior resolução no robô;
- testes com outras arquiteturas de rede, pra melhor compreensão da relação da acurácia da rede com o número de camadas e de parâmetros ajustáveis da mesma.

APÊNDICE A

Pseudocódigo

Neste apêndice são apresentados os pseudocódigos referentes à implementação, treinamento e validação das redes convolucionais utilizadas neste trabalho. O primeiro passo é construir os modelos da rede. O algoritmo 15 implementa a primeira arquitetura de rede, e o algoritmo 17 a segunda arquitetura.

Algoritmo 3: Construção da primeira arquitetura de rede

1 entrada $\leftarrow CamadaEntrada(formato : (300, 400), canais : 1)$
2 camada1 $\leftarrow Convolucional(entrada : entrada, filtros : 32, janela : (5, 5), ativacao : ReLU)$
3 camada2 $\leftarrow Convolucional(entrada : camada1, filtros : 32, janela : (5, 5), ativacao : ReLU)$
4 camada3 $\leftarrow Pooling(entrada : camada2, estrategia : max, janela : (2, 2), passo : (2, 2))$
5 camada4 $\leftarrow Convolucional(entrada : camada3, filtros : 64, janela : (5, 5), ativacao : ReLU)$
6 camada5 $\leftarrow Convolucional(entrada : camada4, filtros : 64, janela : (5, 5), ativacao : ReLU)$
7 camada6 $\leftarrow Pooling(entrada : camada5, estrategia : max, janela : (2, 2), passo : (2, 2))$
8 camada7 $\leftarrow Convolucional(entrada : camada6, filtros : 128, janela : (3, 3), ativacao : ReLU)$
9 camada8 $\leftarrow Upsampling(entrada : camada7, fator : (2, 2))$
10 camada9 $\leftarrow Dropout(entrada : camada8, taxa : 0.5)$
11 camada10 $\leftarrow Convolucional(entrada : camada9, filtros : 64, janela : (1, 1), ativacao : ReLU)$
12 camada11 $\leftarrow Upsampling(entrada : camada10, fator : (2, 2))$
13 camada12 $\leftarrow Dropout(entrada : camada11, taxa : 0.5)$
14 camada13 $\leftarrow Convolucional(entrada : camada12, filtros : 35, janela : (1, 1), ativacao : softmax)$
15 **retorne** *Modelo(entrada, camada13)*

Algoritmo 4: Construção da segunda arquitetura de rede

1 entrada $\leftarrow CamadaEntrada(formato : (300, 400), canais : 1)$
2 camada1 $\leftarrow Convolucional(entrada : entrada, filtros : 64, janela : (5, 5), ativacao : ReLU)$
3 camada2 $\leftarrow Convolucional(entrada : camada1, filtros : 64, janela : (5, 5), ativacao : ReLU)$
4 camada3 $\leftarrow Pooling(entrada : camada2, estrategia : max, janela : (2, 2), passo : (2, 2))$
5 camada4 $\leftarrow Dropout(entrada : camada3, taxa : 0.25)$
6 camada5 $\leftarrow Convolucional(entrada : camada4, filtros : 128, janela : (5, 5), ativacao : ReLU)$
7 camada6 $\leftarrow Convolucional(entrada : camada5, filtros : 128, janela : (5, 5), ativacao : ReLU)$
8 camada7 $\leftarrow Convolucional(entrada : camada6, filtros : 128, janela : (5, 5), ativacao : ReLU)$
9 camada8 $\leftarrow Pooling(entrada : camada7, estrategia : max, janela : (2, 2), passo : (2, 2))$
10 camada9 $\leftarrow Dropout(entrada : camada8, taxa : 0.25)$
11 camada10 $\leftarrow Convolucional(entrada : camada9, filtros : 256, janela : (1, 1), ativacao : ReLU)$
12 camada11 $\leftarrow Convolucional(entrada : camada10, filtros : 256, janela : (1, 1), ativacao : ReLU)$
13 camada12 $\leftarrow Upsampling(entrada : camada11, fator : (2, 2))$
14 camada13 $\leftarrow Convolucional(entrada : camada12, filtros : 64, janela : (1, 1), ativacao : ReLU)$
15 camada14 $\leftarrow Upsampling(entrada : camada13, fator : (2, 2))$
16 camada15 $\leftarrow ConvTransposta(entrada : camada14, filtros : 35, janela : (5, 5), dilatacao : (8, 8), ativacao : softmax)$
17 **retorne** *Modelo(entrada, camada15)*

Após gerar o modelo, é necessário treiná-lo. O algoritmo 11 ilustra os passos do processo de atualização de pesos e validação.

Algoritmo 5: Rotinas de treinamento

Entradas: *modelo*: arquitetura de rede
t_x : imagens de treinamento
t_y : rótulos de treinamento
v_x : imagens de validação
v_y : rótulos de validação
e : épocas de treinamento
b : número de amostras por atualização de gradiente
opti: otimizador escolhido

```
1  pesos ← InicializarPesos(modelo)
2  para i ← 0 até e faça
3      t_x ← Embaralhar(t_x)
4      lotes ← CriarLotes(dados : t_x, rotulos : t_y, amostras_por_lote : b)
5      para cada lote em lotes faça
6          saida ← Propagar(modelo : modelo, entrada : lote)
7          AtualizarPesos(modelo : modelo, entradas : lote, saida_rede :
            saida, otimizador : opti)
8      fim
9      Validar(modelo : modelo, entradas : v_x, saidas_esperadas : v_y)
10 fim
11 retorne modelo
```

O algoritmo 5 apresenta o fluxo de execução do sistema, interligando os métodos descritos anteriormente.

Algoritmo 6: Fluxo de execução

```
1 dataset ← CarregarDataset()
2 dataset ← AdaptarDataset(dataset)
3 modelo ← ConstruirModelo() (Usando o algoritmo 15 ou 17)
4 Treinamento(modelo, dataset) (Como descrito no algoritmo 11)
5 CalcularEstatisticas()
```

REFERÊNCIAS BIBLIOGRÁFICAS

APPLE. **Performing Convolution Operations**. 2016. https://developer.apple.com/library/content/documentation/Performance/Conceptual/vImage/ConvolutionOperations/ConvolutionOperations.html. Disponível em: 14/02/2018. 13, 57

BADRINARAYANAN, A. K. V.; CIPOLLA, R. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. **arXiv:1505.07293**, 2016. 20

BEAR B. W. CONNORS, B. M. F.; PARADISO, M. A. **Neurociências: Desvendando o Sistema Nervoso**. [s.n.], 2002. ISBN 9788582714331. Disponível em: <https://books.google.com.br/books?id=bywtDwAAQBAJ>. 23

BERGSTRA, J.; BREULEUX, O.; BASTIEN, F.; LAMBLIN, P.; PASCANU, R.; DESJARDINS, G.; TURIAN, J.; WARDE-FARLEY, D.; BENGIO, Y. Theano: A cpu and gpu math compiler in python. In: **Proc. 9th Python in Science Conf**. [S.l.: s.n.], 2010. p. 1–7. 68, 70

BRAGA, A. P. **Redes neurais artificiais: teoria e aplicações**. LTC Editora, 2007. ISBN 9788521615644. Disponível em: <https://books.google.com.br/books?id=R-p1GwAACAAJ>. 65

BRETT., L. **Machine learning with R**. [S.l.]: Packt Publishing Ltd, 2013. 77

BROWNLEE, J. **Clever Algorithms: Nature-inspired Programming Recipes**. Lulu.com, 2011. ISBN 9781446785065. Disponível em: <https://books.google.com.br/books?id=SESWXQphCUkC>. 50

CHOLLET, F. **Deep learning with Python**. [S.l.]: Manning Publications, 2017. 69, 71

DUCHI JOHN, H. E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. The Journal of Machine Learning Research, 12:2121–2159, 2011. 80

FERNEDA, E. Redes neurais e sua aplicação em sistemas de recuperação de informação. **Ciência da Informação**, scielo, v. 35, p. 25 – 30, 04 2006. ISSN 0100-1965. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0100-19652006000100003&nrm=iso>. 13, 33

FERREIRA, A. A. Comparação de arquiteturas de redes neurais para sistemas de reconheceimento de padrões em narizes artificiais. Universidade Federal de Pernambuco, 2004. 71

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. http://www.deeplearningbook.org. 51

HAYKIN, S. S. **Redes Neurais**. 2ed. ed. [S.l.]: Bookman Companhia ED, 2001. 13, 31, 32, 34, 37, 38, 39, 40, 41, 42, 43, 44, 46

HE X. ZHANG, S. R. K.; SUN, J. Deep residual learning for image recognition. **arXiv:1512.03385v1**, 2015. 21

HOLLAND, O. Grey walter: the pioneer of real artificial life. In: MIT PRESS, CAMBRIDGE. **Proceedings of the 5th international workshop on artificial life**. [S.l.], 1997. p. 34–44. 19

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. **CoRR**, abs/1412.6980, 2014. Disponível em: <http://arxiv.org/abs/1412.6980>. 80, 81, 82, 83

LECUN, Y. B. Y.; HINTON, G. Deep learning. **Nature**, p. 436– 444, 2015. 57

LONG, J.; SHELHAMER, E.; DARRELL, T. Fully convolutional networks for semantic segmentation. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2015. p. 3431–3440. 13, 21, 24, 59

MASOOD, S.; SHARIF, M.; MASOOD, A.; YASMIN, M.; RAZA, M. A survey on medical image segmentation. **Current Medical Imaging Reviews**, Bentham Science Publishers, v. 11, n. 1, p. 3–14, 2015. 60

MINSKY, M.; PAPERT, S. Perceptrons: An introduction to computational geometry. **M.I.T. Press**, 1969. 31

NEWELL, A. Physical symbol systems. **Cognitive Science**, p. 4:135–183, 1980. 30

PENG X. ZHANG, G. Y. G. L. C.; SUN, J. Large kernel matters – improve semantic segmentation by global convolutional network. **arXiv:1703.02719v1**, 2017. 21

PERERVENKO, V. **Redes Neurais de Terceira Geração: Redes Profundas**. 2015. https://www.mql5.com/pt/articles/1103. Disponível em: 19/02/2018. 13, 53

RICHTER, S. R.; VINEET, V.; ROTH, S.; KOLTUN, V. Playing for data: Ground truth from computer games. In: LEIBE, B.; MATAS, J.; SEBE, N.; WELLING, M. (Ed.). **European Conference on Computer Vision (ECCV)**. [S.l.]: Springer International Publishing, 2016. (LNCS, v. 9906), p. 102–118. 13, 65, 66, 68

RODRIGUES, C. A. d. S. P.; VINHAL, C.; CRUZ, G. da. Fully convolutional networks for segmenting images from an embedded camera. In: **2017 IEEE Latin American Conference on Computational Intelligence (LA-CCI)**. [S.l.: s.n.], 2017. p. 1–6. 73

ROSSUM, G. **Python Reference Manual**. Amsterdam, The Netherlands, The Netherlands, 1995. 69

RUDER, S. An overview of gradient descent optimization algorithms. **CoRR**, abs/1609.04747, 2016. Disponível em: <http://arxiv.org/abs/1609.04747>. 80, 82

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 3rd. ed. [S.l.]: Prentice Hall, 2009. 19, 29

SENTHILKUMAR, M.; RAMASAMY, V.; SHEEN, S.; VEERAMANI, C.; BONATO, A.; BATTEN, L. **Computational Intelligence, Cyber Security and Computational Models: Proceedings of ICC3 2015**. Springer Singapore, 2015. (Advances in Intelligent Systems and Computing). ISBN 9789811002519. Disponível em: <https://books.google.com.br/books?id=vGQ-CwAAQBAJ>. 53

SERMANET K. KAVUKCUOGLU, S. C. P.; LECUN, Y. Pedestrian detection with unsupervised multi-stage feature learning. **International Conference on Computer Vision and Pattern Recognition (CVPR'13)**, 2013. 54

SERMANET, P.; LECUN, Y. Traffic sign recognition with multi-scale convolutional networks. **In Neural Networks (IJCNN), The 2011 International Joint Conference**, p. 2809–2813, 2011. 54

SHUAI Z. ZUO, B. W. B.; WANG, G. Dag-recurrent neural networks for scene labeling. **CVPR 2016**, 2016. 22

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **ICLR 2015**, 2017. 21

SOARES, F. **Introdução À Ciência Da Computação Com Jogos**. Elsevier Brasil, 2012. ISBN 9788535252545. Disponível em: <https://books.google.com.br/books?id=KMkcZ7tR0fkC>. 19

SZEGEDY W. LIU, Y. J. P. S. S. R. D. A. D. E. V. V. A. R. C. Going deeper with convolutions. **CVPR 2015**, 2015. 22

SZELISKI, R. **Computer Vision: Algorithms and Applications**. Springer, 2010. Disponível em: <http://szeliski.org/Book>. 20, 25, 59, 62

TIELEMAN, T.; HINTON, G. Coursera: Neural networks for machine learning. Lecture 6.5 - RMSProp - Technical report, 1980. 81

ZAITOUN, N. M.; AQEL, M. J. Survey on image segmentation techniques. **Procedia Computer Science**, Elsevier, v. 65, p. 797–806, 2015. 60