UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E ELETRÔNICA

Lucas Lopes Trindade

DETECÇÃO DE OBSTÁCULOS PARA CARROS AUTÔNOMOS UTILIZANDO APRENDIZADO PROFUNDO

Florianópolis

2018

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

Trindade, Lucas Lopes
Detecção de obstáculos para carros autônomos utilizando aprendizado profundo / Lucas Lopes Trindade ; orientador, Héctor Pettenghi Roldán, 2018.
87 p.

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

Inclui referências.

1. Engenharia Elétrica. 2. Aprendizado profundo. 3. Carros autônomos. 4. Segmentação semântica. I. Roldán, Héctor Pettenghi. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Elétrica. III. Título.

Lucas Lopes Trindade

DETECÇÃO DE OBSTÁCULOS PARA CARROS AUTÔNOMOS UTILIZANDO APRENDIZADO PROFUNDO

Monografia submetida ao Curso de Graduação em Engenharia Elétrica para a obtenção do Grau de Bacharel em Engenharia Elétrica.
Orientador: Prof. Dr. Hector Pettenghi Roldan

Florianópolis

2018

Lucas Lopes Trindade

Detecção de obstáculos para carros autônomos utilizando aprendizado profundo

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

Florianópolis, 03 de dezembro de 2018.

Prof. Jean Vianei Leite, Dr.
Coordenador do Curso de Graduação em Engenharia Elétrica

Banca Examinadora:

Prof. Hector Pettenghi Roldan, Dr. Eng.
Orientador
Universidade Federal de Santa Catarina

Prof. Dr. Danilo Silva, Ph.D.
Universidade Federal de Santa Catarina

Eng João Guilherme Reiser de Melo, B.Sc.
Universidade Federal de Santa Catarina

RESUMO

Nos últimos anos, é crescente o desenvolvimento de tecnologias aplicadas a carros autônomos. Uma das características básicas destas tecnologias é a capacidade de identificação de outros objetos a sua volta. O presente estudo propõe a execução de uma metodologia de otimização que busca a implementação de um modelo de rede neural convolucional que tem como foco a segmentação semântica de veículos em imagens, visando aplicação em veículos autônomos. A otimização deste modelo é realizado através de um *trade-off* entre algumas das principais características que representam o desempenho de uma rede neural, como a sua acurácia, velocidade de inferência, complexidade de detecção e número de parâmetros treináveis. Para alcançar tais objetivos, foram utilizadas ferramentas de alto nível voltados para o aprendizado de máquina, como o Tensorflow e Keras, em combinação com um conjunto de dados disponibilizado para o público geral, concebido para a aplicação em desenvolvimento de modelos focados na detecção de objetos em meio ao ambiente de tráfego de veículos. Os resultados serão avaliados através de métricas e inferência de imagens em geral, analisando por fim a viabilidade das técnicas utilizadas para aplicação em automação de veículos.

Palavras-chave: Redes Neurais Convolucionais, Segmentação Semântica, Aprendizado Profundo, Veículos Autônomos.

ABSTRACT

In recent years, the development of technologies applied to autonomous cars has been increasing. One of the basic characteristics of these technologies is the ability to identify other objects around them. The present study aims at the execution of an optimization methodology that seeks the implementation of a convolutional neural network model focused on the semantic segmentation of vehicles in images, aiming application in autonomous vehicles. The optimization of this model is given by trade-off between some of the main characteristics that represent the performance of a neural network, such as its accuracy, inference speed, detection complexity and number of trainable parameters. To achieve these objectives, high-level machine-learning frameworks such as Tensorflow and Keras were used in combination with a dataset made available to the general public. These datasets were made for application in the development of models focused on the detection of objects in the environment of vehicle traffic. The results will be evaluated through metrics and inference of images in general, finally analyzing the feasibility of the techniques used for application in vehicle automation.

Keywords: Convolutional Neural Networks, Semantic Segmentation, Deep Learning, Autonomous Vehicles.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CNN	Convolutional Neural Network
RNA	Rede Neural Artificial
MLP	Multi Layer Perceptron
GPU	Graphics Processing Unit
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
FC	Fully Connected
CPU	Central Processing Unit
API	Application Programming Interface
FCN	Fully Connected Neural Network
IoU	Intersection Over Union
IA	Inteligência Artificial

SUMÁRIO

1 INTRODUÇÃO

Um dos grandes sonhos da civilização humana é a criação de máquinas capazes de atuar sem a necessidade do controle e supervisão de um ser humano, que possua capacidade de extrair, das informações que lhe são fornecidas, padrões que fogem da compreensão do homem.

Sendo assim, as redes neurais têm sido exaustivamente abordadas pela comunidade acadêmica nos últimos anos, e o aprendizado profundo se tornou uma poderosa ferramenta para reconhecimento de padrões nesta quantidade colossal de dados que possuímos nos dias de hoje, sendo aplicável em um grande número de áreas de conhecimento.

Por muitos anos, para o desenvolvimento de sistemas envolvendo aprendizagem de máquina, era preciso de uma cuidadosa engenharia e grande domínio sobre o assunto para desenvolver modelos para extração de informação a partir de dados(GOODFELLOW et al., 2016). Agora, após o desenvolvimento de formas de treinamento estocástico, juntamente com unidades de processamento poderosíssimas, como as GPUs-*Graphics Processing Unit*, e quantidades gigantescas de dados disponíveis na internet, tornaram as redes neurais um dos principais focos da comunidade acadêmica quando o assunto é aprendizado de máquina.

As redes neurais possuem um vasto campo de aplicação, como por exemplo, veículos autônomos(WU et al., 2016), detecção de doenças através de exames(RONNEBERGER et al., 2015), análise de mercado financeiro(HEATON et al., 2016), e muitos outros.

A detecção de objetos em imagens, uma das áreas de maior destaque envolvendo aprendizado profundo, sempre foi abordada por muitas técnicas diferentes, mas as redes neurais vem apresentando nos últimos anos resultados muito

superiores à qualquer outra.

Para desempenho distinto na área, em geral é necessário um vasto conjunto de dados e grande processamento computacional para lidar com estes dados aplicados em modelos complexos. Neste trabalho, será abordado justamente métodos otimizados de implementação e treinamento de redes neurais voltadas para a detecção de objetos, com foco em veículos autônomos.

A detecção de objetos pode ser realizada de forma simples, detectando a simples presença do objeto na imagem, indo até a segmentação à nível de pixel dos objetos contidos na imagem.

A abordagem da detecção de objetos será executada de acordo com o último método, chamado de segmentação semântica. Para alcançar tal objetivo, será utilizado um tipo específico de rede neural convolucional, a chamada *fully convolutional network* - FCN(LONG et al., 2015).

1.1 MOTIVAÇÃO

O desenvolvimento de veículos autônomos é uma realidade, onde empresas como Google, Tesla e Uber competem para atingir um alto nível de qualidade e confiança de seus veículos para que possam se fixar no mercado. Um dos grandes desafios, tanto para a automatização parcial de veículos, quanto para os veículos completamente autônomos, estes que ainda estão longe de ser adquiridos pelos consumidores comuns, é a detecção de objetos de interesse ao redor do veículo.

O comportamento e planejamento executado pela IA-Inteligência Artificial de veículos autônomos é definitivamente dependente dos objetos que o cercam, mostrando ações diferentes se existem a sua frente uma rodovia livre, um carro

em movimento, ou um pedestre atravessando a rua. O objeto mais comum em rodovias são outros veículos, como carros de passeio, caminhões, motos, etc.

Devido a esta frequência de aparições, a detecção deve possuir grande acurácia e velocidade de inferência. Por esta razão, a detecção destes objetos é sempre alvo de estudo quando novas técnicas são anunciadas, pois as consequências de uma má detecção são catastróficas, tanto para o usuário do veículo, quanto para as pessoas e bens materiais ao seu redor.

Portanto, durante o desenvolvimento de modelos de redes neurais, não é incomum a realização de *trade-off* entre acurácia, velocidade de execução e complexidade das detecções, por exemplo, tornando necessário uma constante análise dos modelos em diferentes condições de parâmetros e dados.

Desta maneira, é um desafio encontrar uma implementação simples e que ocupe pouco espaço na memória, algo limitado para desenvolvedores com *hardware* menos potentes, que apesar de apresentar bom tempo de inferência, também forneça boas detecções.

Para isto, será necessário dispensar características que elevam a complexidade da rede, como por exemplo a detecção de múltiplas classes de objetos. O modelo resultante poderá ser aplicado em inúmeras aplicações diferentes, e por usuários que não possuam grande poder computacional em mãos.

1.2 OBJETIVOS GERAIS

Implementação de um modelo simples e otimizado de detecção de veículos normalmente encontrados em rodovias, para possível aplicação em veículos autônomos, utilizando técnicas de segmentação semântica, que envolvem a aplica-

ção de aprendizado supervisionado em rede neurais convolucionais de tamanho reduzido e alta velocidade de execução.

1.3 OBJETIVOS ESPECÍFICOS

- Realizar o treinamento de modelos otimizados de redes neurais convolucionais utilizando diferentes tamanhos de *dataset*, aplicados em modelos com arquiteturas de complexidades diferentes.
- Analisar os resultados gerados pelos modelos implementados e averiguar o *trade-off* entre acurácia, velocidade de inferência, número de parâmetros e tempo de treinamento e inferência.
- Determinar viabilidade do modelo de CNN proposto para aplicações reais, em veículos autônomos.

2 FUNDAMENTAÇÃO TEÓRICA

Aprendizado de Máquina, ou *Machine Learning*, é uma área do conhecimento que surgiu de estudos relacionados à inteligência artificial e reconhecimento de padrões. Nesta área é contemplado o estudo e implementação de algoritmos que realizam aprendizado e predições baseadas em dados externos fornecidos, operando através da construção de modelos preditivos(GOODFELLOW et al., 2016).

Tais modelos recebem em sua entrada um conjunto de dados de treinamento, que podem ser qualquer estímulo externo, tanto físicos quanto virtuais. Assim, as predições não são realizadas a partir de instruções pré-determinadas de um algoritmo, e sim feitas com orientações em dados.

2.1 REDES NEURAIS ARTIFICIAIS

Redes neurais artificiais, ou RNA, são baseadas no conceito simplificado do funcionamento biológico do cérebro, onde um neurônio possui a função de processar e disseminar sinais elétricos(RUSSEL; NOVIG, 2003).

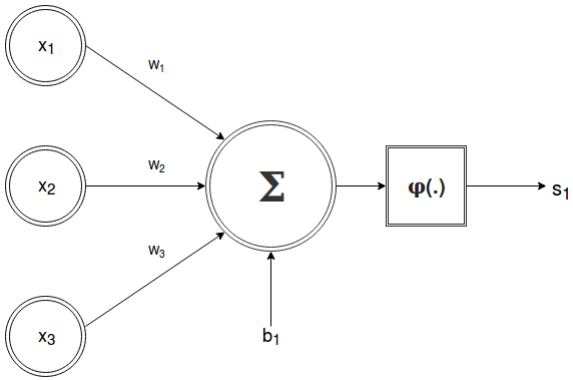
Tais conceitos foram vistos primeiramente em meados de 1940, (MCCULLOCH; PITTS, 1943), e expandidos posteriormente pelo cientista Frank Rosenblatt em 1957, onde foi proposto o que seria a arquitetura mais simples para uma RNA.

Esta RNA simples, opera a partir de entradas binárias, e realiza uma ponderação através de combinações lineares, e quando esta ponderação ultrapassa um limiar, é gerada uma saída binária. A área de estudos das redes neurais ganhou popularidade somente em 1986, com o desenvolvimento do algoritmo da retropropagação do erro.

Segundo Russel e Novig (2003), redes neurais são compostas por nós, estes que estão interligados através de vínculos orientados. Estes vínculos são representeados matematicamente por pesos numéricos(Wi), que determinam a intensidade da participação da informação que flui através de tais ligações.

A combinação linear destes vínculos passa por uma função de ativação que limita a intensidade dos dados de saída. Baseado nesses conceitos, temos a rede neural mais simples, também chamado de *perceptron*, representado de forma visual na Figura 1.

Figura 1 – Modelo de uma rede neural artificial.



Fonte: Elaborado pelo Autor.

$$s_i(t) = \varphi(\sum_{j=1}^{n} W_{ij}x_j + b_i) \tag{2.1}$$

Neste tipo de rede há somente uma camada de neurônios, que operam diretamente com os dados de entrada da rede, que estão organizados na forma vetorial $V = [x_1, x_2...x_i]$.

A expressão matemática que representa tal rede pode ser visualizada na Equação 1.

Tais modelos de rede neural permitem dois tipos de predições: a regressão linear e a regressão logística. Na regressão linear, temos a busca por uma função que irá representar a melhor curva para um conjunto de dados, minimizando os erros. Este modelo é muito utilizado quando é preciso estimar o valor final a partir de amostras.

A regressão logística permite estimar a probabilidade de ocorrer diferentes eventos, podendo ser interpretado como um modelo classificador dos dados de entrada(GOODFELLOW et al., 2016).

O valor de b_i, chamado de *bias*, adiciona um maior grau de liberdade à rede, tornando os neurônios mais sensíveis aos estímulos externos, ativando-se para sinais de entrada cada vez menores(NIELSEN, 2015).

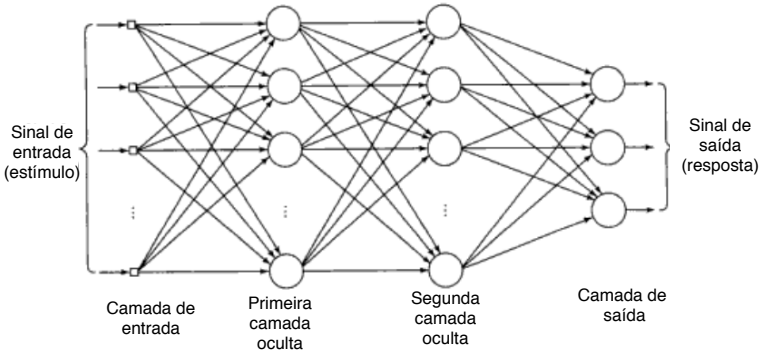
2.1.1 O MultiLayer Perceptron

O Multilayer Perceptron(MLP) é definido como uma rede de alimentação direta, ou *feed-forward*, pois segundo Goodfellow et al. (2016), a informação flui gradualmente através da rede, partindo das camadas de entrada, propagando-se em direção às camadas de saída, e como descrito por Russel e Novig (2003), pode ser representada matematicamente como uma função de suas entradas, sem nenhum outro parâmetro interno que não sejam seus pesos e *biases*.

As redes de alimentação direta são compostas de uma camada de entrada, uma ou várias camadas ocultas, e a camada de saída, podendo ser concebida em sua forma mais simples como uma rede de perceptrons, como pode ser visto na Figura 2.

Uma rede neural MLP é capaz de extrair características

Figura 2 – O Multilayer Perceptron.



Fonte: Haykin (2007).

de ordem mais elevada, conforme a rede se torne mais profunda e os nós mais numerosos(HAYKIN, 2007), apesar que indefinidamente aumentar a rede neural é inútil quando se há pouca informação para realizar o aprendizado, podendo até mesmo reduzir seu desempenho. Dizemos que uma camada é totalmente conectada quando cada neurônio, ou nó, é ligado por vínculos ponderados com todos os outros neurônios da camada anterior.

Apesar da popularização desta nomenclatura, certos autores não concordam em chamar uma rede de múltiplas camadas como uma rede de perceptrons(NIELSEN, 2015), pela razão de o perceptron originalmente concebido por Mcculloch e Pitts (1943) utiliza uma função de ativação do tipo Heaviside, ou seja, não diferenciável, enquanto que uma rede neural de múltiplas camadas está livre para utilizar outros tipos de ativações.

Além das redes de alimentação direta, existem as redes recorrentes, que utilizam suas saídas(RUSSEL; NOVIG, 2003)

para alimentar ciclicamente suas entradas, e diferentemente das redes de alimentação direta, podem apresentar memória de curto prazo, o que pode ser visto como um modelo mais próximo do funcionamento biológico do cérebro.

As redes de múltiplas camadas criaram uma nova área do conhecimento dentro do aprendizado de máquina, que vem evoluindo com força na última década, o chamado aprendizado profundo, ou *Deep Learning*.

O alto poder de processamento, principalmente com o desenvolvimento das *Graphics Processing Units* - GPU, além da quantidade colossal de informações disponíveis, estão entre os principais fatores que tornam o aprendizado profundo uma realidade.

Tais redes conseguem extrair características extremamente complexas conforme há um aprofundamento das camadas, onde as camadas mais próximas da entrada extraem conceitos mais simples da informação. Portanto, o aprendizado da hierarquia de interação entre características complexas e simples torna o aprendizado profundo tão atraente e com resultados tão significativos, sendo uma das áreas do conhecimento mais intensamente estudadas da década.

Um dos pontos chave que permitiu a popularização de redes neurais e a sua aplicação em problemas reais com alto grau de complexidade, foi a utilização de funções de ativação diferenciáveis, que possibilitou a implementação do algoritmo de retropropagação do erro durante o processo de treinamento.

Segundo Haykin (2007), o objetivo de uma função de ativação é a de limitação dos dados, ou seja, mapear os dados de entrada em dados de saída em outra escala. Adicionar uma não-linearidade à rede permite que a mesma respeite o teorema da aproximação universal de Cybenko, que torna uma rede neural capaz de representar qualquer função.

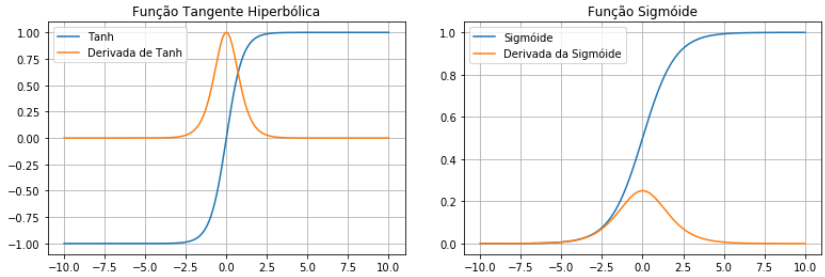
Entre as funções de ativação mais populares, temos: a

sigmóide, tangente hiperbólica, a *softmax* e a *rectified linear unit*(ReLU), que podem ser visualizadas nas Figuras 3 e 4.

Tabela 1 – As principais funções de ativação.

Sigmóide	Tanh	ReLU	Softmax
$\frac{1}{1+e^{-y_i}}$	$\frac{1-e^{-2y_i}}{1+e^{-2y_i}}$	$max(0, y_i)$	$\frac{e^{y+i}}{\sum_j^n e^{y_j}}$

Figura 3 – As funções Sigmóide e Tangente Hiperbólica.

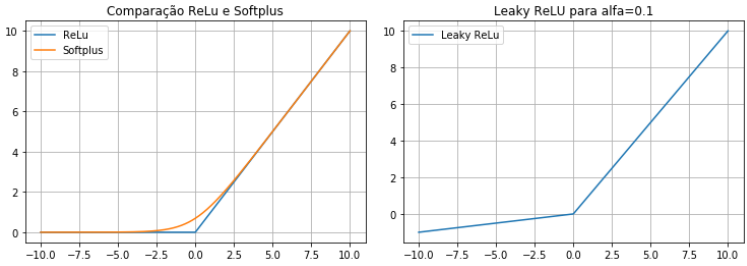


Fonte: Elaborado pelo Autor.

A função de ativação ReLU, popularizada por Krizhevsky et al. (2012), é uma das mais recentes funções de ativação apresentadas, e é caracterizada por ser uma simples função linear, porém limitada em zero para valores de entrada no intervalo $x_i \in (-\infty, 0)$. Ela é principalmente utilizada nas camadas ocultas das redes neurais, mostrando velocidade de convergência superior durante o cálculo do gradiente estocástico em relação às funções mais antigas, por ser linear e não saturar a saída.

A função ReLU apresenta problemas em determinadas situações, como por exemplo, caso ela receba valores de gradiente excessivamente altos durante a fase de treinamento, o

Figura 4 – As funções Softplus, ReLU e Leaky ReLU.



Fonte: Elaborado pelo Autor.

que pode resultar nos pesos se atualizarem de uma maneira que o neurônio nunca mais se ativará novamente, tornando-se irreversivelmente inativo(GOODFELLOW et al., 2016).

Para solucionar alguns dos problemas foram concebidas algumas variações, como por exemplo a *leaky* e a *parametric* ReLU, ambas lineares, ou a *softplus*.

A função *softplus*, apresentada na Equação 2.2, é do tipo não-linear, porém apresenta resultados bastante semelhantes com a ReLU, e possui a curiosa característica de sua derivada ser a função sigmoide. Por tais razões, não possui a performance computacional da ReLU, apesar de sutilmente corrigir o problema de neurônios inativos. A *leaky* ReLU, como pode ser vista na Equação 2.3, é similar à sua versão original, com a diferença de possuir um declive linear com inclinação fixa para o intervalo $(-\infty, 0)$.

$$\varphi(y_i) = ln(1 + e^{y_i}) \tag{2.2}$$

A *parametric* ReLU possui a propriedade de ter a inclinação do declive linear como um parâmetro a ser aprendido durante o treinamento(HE et al., 2015). Os benefícios de tais funções não foram complemente esclarecidos, apesar de so-

lucionarem de forma sutil a questão dos neurônios inativos.

$$\varphi(y_i) = max(\alpha y_i, y_i) \tag{2.3}$$

Em redes neurais de classificação, é amplamente utilizada a função *softmax* na camada de saída, por realizar uma avaliação probabilística dos resultados finais da rede. Diferentemente das outras funções, o valor de cada nó dependerá dos outros na mesma camada.

2.1.2 Treinamento de uma Rede Neural Artificial

Segundo Haykin (2007), aprender é o ato que gerará uma mudança de comportamento quando uma unidade de processamento é apresentada a um estímulo externo, devido a excitações vivenciadas em momentos passados. Uma rede neural possui a capacidade de aprender de acordo com os exemplos que lhes são fornecidos, caracterizando no que é definido em Barreto (2002) como um aprendizado conexionista, em que não é procurado o estabelecimento de regras que irão moldar a inteligência artificial, e sim determinar a intensidade dos vínculos dos neurônios, simbolizando uma aquisição de conhecimento.

A técnica atual mais popular para realizar o treinamento de uma rede neural, é através da retropropagação de erro, ou *backpropagation*, que busca a otimização do espaço de pesos W através da minimização da função de custo imposta através do cálculo do seu gradiente, método este que é uma abordagem analítica mais eficiente do algoritmo de aprendizagem Gradiente Descendente Estocástico, ou *Stochastic Gradient Descent* (SGD) (KARPATHY, 2016).

O aprendizado supervisionado é caraterizado pela utilização de estímulos externos já previamente rotulados com os resultados esperados, durante o processo de otimização da

rede neural artificial(BARRETO, 2002). O seu contraposto é o aprendizado não-supervisionado, em que ao invés de utilizar informações já rotuladas, temos a procura por características em comum entre todos os estímulos, e o correspondente agrupamento de dados similares(RUSSEL; NOVIG, 2003).

A função de custo(E) é a função que relaciona a saída fornecida pela rede(y_i), com a saída verdadeira(y^i), ou seja, uma forma de medir o desempenho da aprendizagem para um conjunto n de amostras(HAYKIN, 2007).

Apesar de ter sido inicialmente introduzida nos anos 70, a técnica de retropropagação de erro ganhou popularidade somente em 1986 através da publicação do artigo de *Rosenblatt*, e o centro do seu funcionamento é a realização do cálculo do gradiente da função de custo, em função dos parâmetros de cada neurônio da rede.

Esta é uma das razões para a necessidade de funções de ativação serem diferenciáveis, pois é realizada a derivação parcial de toda a rede neural. Tais operações dão a oportunidade de entender o quanto a variação dos pesos e *biases* afeta no comportamento da rede.

Entre as funções de custo mais empregadas em aprendizado supervisionado, temos a função de custo quadrático e a função entropia cruzada, na Equação 2.4, sendo esta última a mais aplicada em redes de regressão logística(KARPATHY, 2016).

$$E = -\sum_{i}^{n} y^{(i)} log(y_i) \tag{2.4}$$

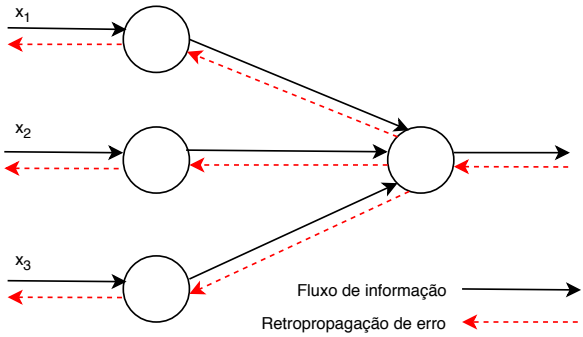
O treinamento de uma rede neural através do aprendizado supervisionado é realizado utilizando conjuntos de dados, que são um conjunto de dados que possuem os estímulos externos e seus respectivos resultados esperados(KARPATHY, 2016). Nos algoritmos computacionais, a alimentação da rede é feita utilizando subconjuntos destes dados, necessi-

dade causada pela limitação da memória computacional, chamados de *batches*, que possuem a desvantagem de tornar a convergência um processo mais ruidoso e instável. Uma época, ou *epoch*, corresponde o momento em que todos os *batches* de um conjunto de dados são alimentados na rede.

Para o cálculo do gradiente da função de custo, é aplicado o conceito de regra da cadeia, muito aplicada no cálculo diferencial, e criada por Gottfried Leibni.

O fluxo de informações durante a retropropagação do erro pode ser visualizada na Figura 5.

Figura 5 – Visualização da retropropagação do erro.



Fonte: Elaborado pelo Autor.

O gradiente é calculado camada por camada, no sentido oposto em relação ao fluxo de informações que flui pela rede. Após os cálculos, é necessário atualizar os pesos e *biases*.

$$W_i^{n+1} = W_i^n + \eta \frac{\partial E}{\partial W_i} \tag{2.5}$$

O método SGD pode gerar uma visualização gráfica como na Figura 6, em que temos a superfície formada pelo função de custo em relação ao espaço de pesos(W), assim

como a progressiva atualização deste pesos na busca pelo seu valor ótimo, localizado no ponto mínimo do vale desta superfície, que equivale ao valor mínimo da função de custo.

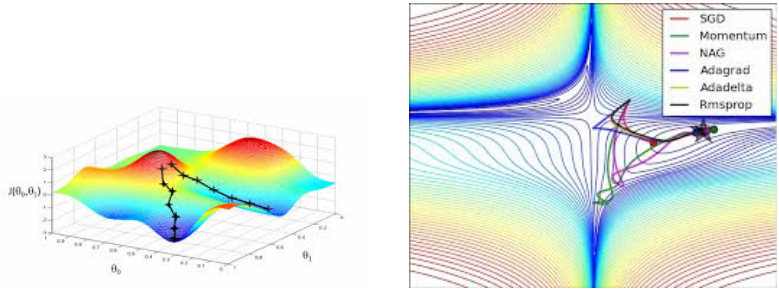
Um parâmetro importantíssimo para a retropropagação de erro é a taxa de aprendizagem(η), que pode ser visualizada na Equação 2.5. Ela determina o tamanho do passo que será dado para chegar no ponto mínimo do gradiente, e seu valor é fundamental na convergência da rede.

Caso tais passos sejam muito longos, torna-se impossível a convergência para um ponto mínimo, e se muito pequena, o processo de treinamento tomará muito tempo. Sua importância está também no comportamento de ativação dos neurônios, pois a utilização de valores elevados podem causar a "morte" dos nós quando acompanhados da função de ativação ReLU(GOODFELLOW et al., 2016).

Uma técnica interessante, apresentada em Zeiler (2012), é a do declínio gradual da taxa de aprendizagem, conforme a rede é treinada. Isto possibilita um início veloz na convergência, porém garantindo que se atinja o ponto mínimo global do gradiente.

Além da redução da taxa de aprendizagem, existem variações do método SGD que podem tornar o método de convergência mais veloz (KARPATHY, 2016). Com a introdução de um hyperparâmetro v, relativo a velocidade, temos o método *momentum*, que como o nome sugere, o problema é visto quase como do ponto de vista físico, em que dando uma velocidade inicial a um objeto ele atingirá o ponto mínimo do vale mais rapidamente. Outros métodos adaptativos abordam a redução da taxa de aprendizado com menor custo computacional e resultados superiores ao do SGD com *momentum*, como podem ser visualizadas na Figura 6.

Figura 6 – Visualização em (a) da função de custo em função do espaço de pesos. Em (b) temos a comparação entre os principais métodos de aprendizagem.



(a) (b)

Fonte: Karpathy (2016).

2.2 REDES NEURAIS CONVOLUCIONAIS

As redes neurais artificiais apresentadas até o momento apresentam certo limite de desempenho para determinadas aplicações, principalmente quando o estímulo externo é uma imagem. A RNA recebe em sua camada de entrada um vetor, sendo necessário portanto que as dimensões da informação sejam convertidas de sua forma original para que possa ser alimentada na rede(KARPATHY, 2016). Isto resulta em uma grave perda de informações espaciais, limitando o desempenho da rede.

Redes Neurais Convolucionais, ou do inglês *Convolutional Neural Network*(CNN) são, segundo Goodfellow et al. (2016), redes neurais cuja arquitetura é voltada especialmente para o processamento de informações cuja topologia é em grades, como os pixeis de uma imagem. Então, ao invés de utilizar a combinação linear para processar a informação,

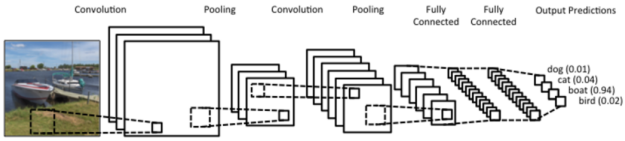
é aplicada a operação de convolução em pelo menos uma de suas camadas.

A operação de convolução, conforme Olah (2014), pode ser interpretada como a integral da multiplicação ponto a ponto de duas funções, conforme uma delas é invertida e transladada sobre a outra. Na Equação 2.6 temos a representação matemática da convolução de dois sinais contínuos.

$$s(t) = \int_{-\infty}^{\infty} x(a)w(t-a)da \tag{2.6}$$

Uma rede neural convolucional é constituída de uma sequência de camadas, possuindo três tipos principais: a camada de convolução, a camada de *pooling*, e a camada completamente conectada, exemplificadas na Figura 7.

Figura 7 – Exemplo de arquitetura de uma CNN.

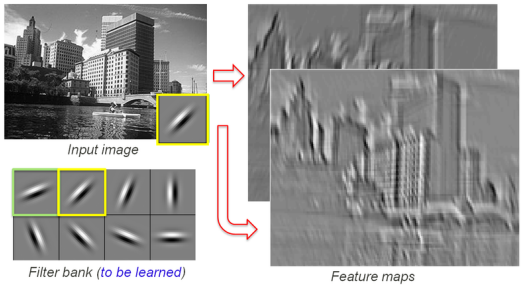


Fonte: Olah (2014).

A camada convolucional é caracterizada por realizar a convolução da entrada por uma estrutura chamada de filtro, ou *kernel*, como pode ser visto na Figura 9. Tal estrutura é basicamente uma matriz de pesos, estes que serão justamente os parâmetros alterados conforme ocorra a etapa de treinamento da rede, além de possuir dimensões consideravelmente reduzidas em relação a entrada(KARPATHY, 2016).

As camadas convolucionais se baseiam na reutilização da mesma matriz de pesos para operar com toda a entrada.

Figura 8 – Exemplo visual do *kernel* e o *feature map* resultante após a convolução.



Fonte: Olah (2014).

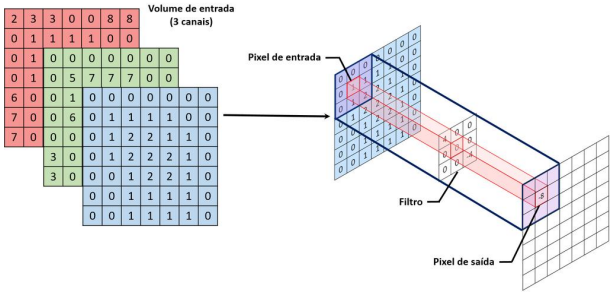
Este método promove portanto a redução na quantidade de parâmetros treináveis se comparado com as redes neurais já mencionadas, e resulta em uma grande economia de memória, além de aumento na velocidade de processamento computacional(GOODFELLOW et al., 2016).

O resultado da operação de convolução entre a entrada e o *kernel*, chamada de *feature map*, pode ser visualizada na Figura 8, e suas dimensões e comportamento durante a convolução com a entrada são dependentes de hiperparâmetros próprios da arquitetura.

Em primeiro lugar, o *stride* da camada convolucional especifica em quantos pixeis o *kernel* será transladado horizontalmente e verticalmente quando for deslocado sobre a entrada, quanto maior for o *stride*, menor volume terá o *feature map*.

Dependendo do parâmetro *stride*, assim como a resolução da imagem e do *kernel* e da entrada, não será possível transladar o kernel por um número inteiro de vezes. Isto pode causar perda de detecção de características nas bordas

Figura 9 – Deslocamento de *kernel* na camada de convolução.



Fonte: Karpathy (2016)

da imagem, ou então redução indesejada das dimensões do *feature map* resultante.

Desta forma, é preciso introduzir a técnica *padding*, cujo hiperparâmetro determina quantas linhas e colunas de zeros serão adicionadas nas bordas na matriz de entrada. Esta técnica permite inclusive manter iguais as dimensões originais da imagem de entrada e do *feature map*(KARPATHY, 2016).

O parâmetro profundidade, ou *depth*, determinará quantos *kernels* existirá em cada camada de convolução, resultando em uma maior extração de características e consequentemente mais camadas de *feature maps*.

A camada convolucional possui a função de extrair características pertinentes da entrada, podendo ser bordas, cores, texturas e outros (KARPATHY, 2016), e assim como nas redes neurais MLP, camadas de operação com pesos são seguidas de funções de ativação, que na grande maioria das arquiteturas será a ReLU e suas variações.

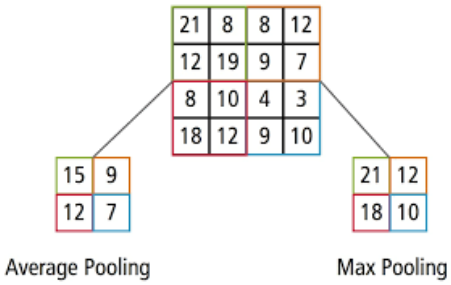
A camada de *pooling*, normalmente utilizada após a camada de convolução, possui a função de primariamente

reduzir as dimensões do *feature map*(altura e comprimento), implicando na redução do número de parâmetros nas próximas camadas(KARPATHY, 2016). Tal camada auxilia também, segundo Goodfellow et al. (2016), na invariância em pequenas translações na imagem.

Tal redução nas dimensões espaciais pode ser feita através da operação de *max pooling*, de longe a mais utilizada, ou da *average pooling*. Na *max pooling* é realizada a captura do pixel de maior valor em uma área retangular de tamanho pŕe-determinado, gerando um novo *feature map* conforme a janela é transladada.

Esta amostragem dos pixeis pode causar perda de informações, porém apresenta bons resultados práticos do ponto de vista de processamento computacional. Na *average pooling*, é aplicada a média quadrática da janela que se desloca pelo *feature map*, reduzindo os parâmetros de mesma maneira, porém de forma mais suave e com maior custo computacional. Ambas formas de amostragem abordadas podem ser visualizadas na Figura 10.

Figura 10 – Comparação entre técnicas de *pooling*, considerando uma janela 2x2 e *stride* 2.



Fonte: Karpathy (2016).

Em contraposição às camadas convolucionais, existem

as camadas de convoluções transpostas(LONG et al., 2015), também chamadas de de-convolução ou *fractionally strided convolution*. Tal camada mostra sua utilidade quando existe a necessidade realizar uma acréscimo na amostragem da sua entrada, técnica também chamada de *up-sampling*, ou seja, amostrar uma imagem de baixa resolução para uma maior, através de um aumento artificial da informação. Existem diversos métodos que podem alcançar tal objetivo, como a interpolação bilenar, porém elas não apresentam nenhum parâmetro treinável, limitando a capacidade de aprendizado e indo em direção oposta à ideia conexionista idealizada para estes tipos de redes neurais(XU et al., 2014).

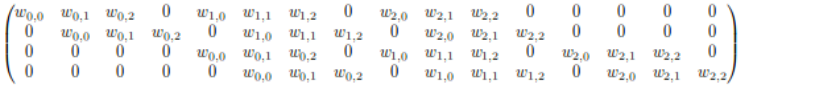
O propósito de tal camada é a de realizar uma operação no sentido oposto ao da camada convolucional, porém apresentando o mesmo nível de conectividade(DUMOULIN; VISIN, 2016). É dito como sendo o caminho oposto, pois o cálculo da convolução transposta é comparável com o cálculo realizado pelo algoritmo de otimização durante a etapa de atualização dos parâmetros, já que a informação flui no sentido contrário.

A convolução transposta opera de forma similar à convolução tradicional(DUMOULIN; VISIN, 2016), e pode ser melhor compreendida ao se observar a operação matemática envolvendo a chamada matriz de convolução, com a entrada. A matriz de convolução representa a operação de convolução na forma de uma multiplicação de matrizes.

Desta forma, o *kernel* deve ser convertido em vetor e adicionado a uma matriz esparsa, de acordo com a sua ordem de operação com a matriz de entrada. A matriz de convolução gerada a partir de uma operação de convolução de um *kernel* de tamanho 3x3, *stride* igual a 1 e sem *padding*, pode ser observada na Figura 11.

Tal matriz, quando multiplicada com a entrada, gera o *feature map* resultante da convolução tradicional. Por sua

Figura 11 – Matriz de convolução.



Fonte: Dumoulin e Visin (2016).

vez, se o mesmo for multiplicado pela transposta desta matriz de convolução, será obtida a matriz de entrada original na sua forma vetorial.

Isto permite a elevação da dimensão dos *feature maps*, algo muito útil para CNNs que utilizam o artifício de conexão entre camadas diferentes de convolução, a fim de realizar a transmissão de características de ordem mais elevadas. Para esta conexão acontecer, é preciso que ambas as camadas possuam as mesmas dimensões para completa comunicação, tornando o *up-sampling* necessário para os *feature maps* com dimensões reduzidas, porém mantendo o padrão de conectividade.

Após as etapas de convolução e *pooling* já citadas, temos, ao final de muitas CNNs, uma ou mais camadas completamente conectadas, chamadas também de *fully connected layer*(FC). Esta camada funciona como um rede MLP, em que cada nó é conectado através de pesos a todos os outros nós anteriores. Nesta camada de saída, é tomada como preferência a função softmax como função de ativação.

A camada FC permite a hierarquização das características propagadas através das camadas anteriores, expandindo o aprendizado da rede e o fluxo de informações, que na saída irá atribuir uma classe à imagem de entrada. Mesmo que muito utilizada, em diversas aplicações, a *FC* é dispensada, e as funções softmax ou sigmoide são aplicadas diretamente

no último *feature map* da rede neural convolucional(LONG et al., 2015).

2.3 TRABALHOS CORRELATOS

A busca de técnicas e algoritmos que tem como foco o desenvolvimento dos carros autônomos foi gradativamente se desenvolvendo nas últimas décadas, porém ganhou grande impulso com o desafio DARPA Grand Challenge em 2004. A área de detecção de obstáculos dinâmicos e estáticos é um dos pilares do desenvolvimento dos carros autônomos, que precisam realizar constantemente a detecção, reconhecimento e rastreamento de outros carros, pedestres, ciclistas e uma infinidade de outros obstáculos.

A detecção de obstáculos é realizada por sensores que captam informações do ambiente, utilizando tecnologias como radar, LIDAR, ultrassom, câmeras estéreo e GPS de alta precisão para localização. LIDAR e radares foram inicialmente os sensores mais utilizados, por permitirem um grau de visibilidade maior, além de coletar melhor dados espaciais, deixando as câmeras como suporte para tais sensores.

As imagens geradas pelas câmeras, estas que quase sempre são utilizadas em visão estéreo, até recentemente eram processadas utilizando técnicas como mapas de disparidade (BAHA, 2014), detecção de contornos(HUANG; LIU, 2016) ou *Histogram of Oriented Gradients* (HAN et al., 2006), porém tais métodos não apresentam resultados que possam tornar carros completamente autônomos para uso urbano uma realidade.

Foi somente com a desenvolvimento do aprendizado profundo e unidades de processamento mais potentes que as câmeras ganharam mais atenção na detecção de obstáculos.

Com a publicação da Alexnet(KRIZHEVSKY et al., 2012),

vencedora do ILSVRC(ImageNet Large Scale Visual Recognition Challenge) de 2012, houve uma verdadeira revolução na área das redes neurais convolucionais profundas, atingindo precisão de 17% no erro em top-5 com um conjunto de dados de 1,3 milhões de imagens e 1000 classes.

Tal publicação instigou o desenvolvimento de outras CNN's cada vez mais profundas, como a VGGNet(SIMONYAN; ZISSERMAN, 2014), GoogLeNet(SZEGEDY et al., 2015) ou a ResNet(HE et al., 2015), esta última que possui 152 camadas.

Um dos métodos mais utilizados de detecção e localização é a aplicação das chamadas *bounding boxes*, que nada mais são do que retângulos que limitam determinado objeto.

Inicialmente, umas das redes que mais se destacou neste ramo, a R-CNN(GIRSHICK et al., 2013) utiliza um algoritmo que propõe inúmeras sub-regiões na imagem, chamado de Procura Selecionada.

Esta arquitetura é extremamente lenta graças às redundâncias nos cálculos, porém tal desempenho foi melhorado em sua versão posterior, a Fast R-CNN(GIRSHICK, 2015), que propôs o método *Region of Interest Pooling*-RoiPool, técnica que aproveita cálculos já utilizados e evita redundância computacional, algo presente na arquitetura R-CNN.

Versões aperfeiçoadas foram propostas em Faster R-CNN(REN et al., 2015) e Mask R-CNN(HE et al., 2017), esta última que realiza a segmentação dos objetos, algo muito mais atraente para aplicação em detecção dos objetos para carros autônomos por fornecer melhor estimativa do posicionamento dos obstáculos.

Arquiteturas que realizam segmentação a nível de pixel, e possuem a característica de agrupar pixeis de mesma classe como uma única entidade, efetuam a chamada segmentação semântica.

Além de arquiteturas como a Mask R-CNN já citadas, temos a U-net(RONNEBERGER et al., 2015), SegNet(JÉGOU

et al., 2017a), DeepLab-CRF(CHEN et al., 2018), FCN(LONG et al., 2015), Dilated Conv.(YU; KOLTUN, 2015) e outras, que realizam a detecção de objetos através de segmentação semântica.

Tabela 2 – Alguns dos modelos de rede neural que realizam segmentação semântica.

Rede	Multi?	Dataset	Nº de imagens	Parâm. [M]	Tempo [ms]	IoU (média)
FCN-VGG16	**Sim**	Pascal VOC 2011	11 530	134	210	56,0
FCN-GoogLeNet	**Sim**	Pascal VOC 2011	11 530	6	59	42,5
SegNet	**Sim**	SUN RGB-D	10335	14,7	52,6	60,1
DeepLab-CRF	**Sim**	Pascal VOC 2012	11 530	65,1	125	71,6
Dilated Conv.	**Sim**	**Microsoft COCO**	**328 000**	-	-	71,3
Unet	Não	DIC-HeLa	20	31,3	-	77,3
Prop. Unet-8	Não	KITTI Vision 2012	7 480	**0,48**	**12,7**	89,1
Prop. Unet-16	Não	KITTI Vision 2012	7 480	1,96	20,4	**95,2**

Segue na Tabela 2 os resultados e outras informações úteis sobre alguns dos principais modelos desta área de estudo. Os conjuntos de dados Pascal VOC 2011 e 2012 possuem ambos 11 530 imagens para treinamento cada, com 20 classes diferentes, enquanto que a DIC-HeLa possui 20 imagens com somente 2 classes.

Tais conjuntos de dados influenciam no desempenho e no funcionamento geral da rede, pois como ser visto na coluna "Multi?", todas as redes citadas, exceto a U-net, são capares de realizar segmentação de múltiplas classes. Tais fatores tornam a comparação direta dos resultados algo complexo e muitas vezes inviável.

Estes modelos apresentados possuem graus de complexidades diferentes. Enquanto que as redes FCN-VGG16 e Unet possuem a principal característica de serem redes neurais completamente convolucionais do tipo *encode-decoder*, possuindo implementação simples, a rede DeepLab aplica o conceito de CRF-*Conditional Random Field*(KRÄHENBÜHL; KOLTUN, 2011), que utiliza uma segunda rede neural para gerar de forma eficiente os contornos dos objetos segmentados.

Segundo Wu et al. (2016), algoritmos de detecção de objetos para aplicação em carros autônomos devem ser velozes, para exibir o resultado do processamento em tempo real, ter alta acurácia, baixo consumo de energia, além de modelos de dimensões reduzidas.

Estes fatores podem necessitar possíveis *trade-offs*, pela razão de modelos com maior acurácia em geral serem maiores e mais lentos.

3 IMPLEMENTAÇÃO E TREINAMENTO DA CNN

Neste capítulo, será discutida toda a abordagem executada para alcançar os objetivos impostos no começo deste trabalho.

Primeiramente, será explanado o processo de manipulação do conjunto de dados, ou do inglês, *dataset*, para em seguida abordarmos a implementação do modelo de CNN responsável pela segmentação a nível de pixel, assim como os métodos e parâmetros estabelecidos para o seu treinamento.

3.1 MANIPULAÇÃO DO CONJUNTO DE DADOS

O desenvolvimento de modelos baseados em aprendizado supervisionado necessitam de um conjunto de dados para realizar o seu treinamento. Existem uma grande quantidade destes conjuntos de dados para o uso público pela internet, alguns deles já citados anteriormente, que podem conter dados sobre os mais diferentes assuntos, métodos de captura dos dados e sua quantidade, qualidade de aquisição, etc.

Portanto, é preciso utilizar um conjunto de dados que possua uma grande quantidade de imagens obtidas em um ambiente de tráfego de veículos, de preferência do ponto de vista do automóvel, em diferentes iluminações e tráfego.

Todas as imagens devem possuir rótulos, fornecendo informações sobre todos os objetos de interesse contidos na mesma, como seus tipos e suas localizações 2D, ou seja, as coordenadas dos pixeis máximos e mínimos em ambos os eixos.

Estas coordenadas apontam a localização do objeto na

imagem através de uma indicação retangular, chamado de *bounding box*.

Como fonte principal de dados para treinamento, foi utilizado o *dataset* KITTI Vision 2012 (GEIGER et al., 2012), fornecido pelo Karlsruhe Institute of Technology (KIT) e o Toyota Technological Institute at Chicago(TTI-C).

Este conjunto de dados foi coletado através da plataforma de direção autônoma *AnnieWAY*, que é um veículo da marca Volkswagen Passat equipado com duas câmeras de alta definição em modo estéreo, tanto monocromáticas quanto coloridas, Velodyne 3D Lidar, sistema de localização baseado em GPS/GLONASS, e um sensor radar. Esta plataforma trafegou pelas ruas e rodovias da cidade de Karlsruhe, na Alemanha, para realizar todas as aquisições que formam o conjunto de dados, durante diferentes momentos do dia.

O objetivo da confecção do *dataset* KITTI é a criação de um *benchmark* desafiador de visão computacional do mundo real, com foco em: visão estéreo, fluxo óptico, odometria visual, detecção de objetos 2D/3D e rastreamento 3D. Para este trabalho portanto foi utilizado o conjunto de dados gerado para o *benchmark* de detecção de objetos 2D, que possui todas as informações necessárias.

O conjunto de dados possui 7481 imagens com múltiplos objetos de interesse em cada, detalhadamente anotadas, totalizando 80256 objetos rotulados. As imagens estão no formato 'PNG' e são coloridas. As informações contidas em cada rótulo, incluem:

- *Type*: descreve o tipo de objeto, podendo ser, 'Car', 'Van', 'Truck', 'Pedestrian', 'Person Sitting', 'Cyclist', 'Tram','Misc' ou 'DontCare';

- *Truncated*: valor entre 0 e 1 que indica se o contorno do objeto sai do dos limites da imagem;

- *Occluded*: valor inteiro entre 0 e 3 que indica o nível do oclusão do objeto pelo cenário;

- *Alpha*: varia entre -π e π, indica o ângulo de observação do objeto em relação ao eixo da câmera;

- *Bbox:* coordenadas bidimensionais do bounding box que localiza o objeto, representada como as coordenadas dos pixeis nos quatro cantos(cima, baixo, lado esquerdo e direito);

- *Dimensions*: dimensões tridimensionais do objeto, em metros;

- *Location*: localização tridimensional do objeto em relação as coordenadas da câmera, em metros;

- *Rotation-y*: rotação da câmera no eixo y, variando entre -π e π;

Muitas destas informações são irrelevantes para o treinamento da CNN proposta, por isso foi realizada uma filtragem dos rótulos das imagens, que antes estavam armazenadas em arquivos de texto diferentes para cada imagem do conjunto de dados, em um único arquivo do tipo *comma-separated values*(CSV), contendo somente as informações necessárias.

A geração de tal arquivo foi realizado através de um algoritmo elaborado em Python3, e o resultado é uma lista contendo 6 colunas: nome da imagem, tipo do objeto rotulado e o *bounding box*, ou seja, quatro coordenadas nomeadas "xmin", "ymin", "xmax", ymax". Com tal arquivo em mãos, torna-se muito mais conveniente trabalhar com este conjunto de dados, por compactar todas as informações úteis em um único lugar.

3.1.1 Geração de máscaras

O modelo da CNN que será implementado e posteriormente discutido, possui em sua camada de saída uma matriz de mesma resolução que a imagem alimentada na entrada da rede. Por isso, tanto a imagem quanto seu rótulo, que será chamado de *máscara*, devem possuir as mesmas dimensões.

Por esta razão, é preciso converter os rótulos em máscaras, utilizando um breve algoritmo em Python 3. Esta máscara possui valor 0 para o que é considerado o plano de fundo da imagem, e 1 onde está localizado o objeto de interesse.

Tal ação torna necessário mudar o que eram antes diferentes tipos de objetos(carro, pedestre, etc), ou seja, uma escolha multi-variável, em uma escolha binária. Portando, é preciso realizar uma nova manipulação dos rótulos, tornando alguns dos tipos de objetos de interesse como plano de fundo, e os objetos que são o foco deste trabalho, como um único objeto de interesse.

Figura 12 – Imagens que alimentarão a CNN durante a fase de treinamento e posteriormente na avaliação.

(a) Imagem do dataset.

(b) Máscara gerada através dos rótulos.

Fonte: Elaborado pelo Autor.

Devido ao fato de que o foco deste trabalho é a detecção de veículos para aplicação em carros autônomos, os tipos

de objetos 'Pedestrian', 'Person Sitting', 'Cyclist','Misc' e 'DontCare' foram considerados planos de fundo, enquanto que 'Car', 'Van', 'Truck' e 'Tram' foram definidos como um único objeto de interesse.

Esta decisão, além de ocorrer graças a limitação do modelo utilizado, resulta em maior eficiência computacional e teoricamente em uma acurácia superior. O resultado da geração das máscaras podem ser visualizado na Figura 12.

3.2 FERRAMENTAS COMPUTACIONAIS UTILIZADAS

O desenvolvimento deste trabalho foi baseado em linguagens e bibliotecas abertas para o público em geral, e são amplamente utilizadas para aplicações gerais em aprendizado de máquina, assim como mais especificamente em aprendizado profundo, que é abordado neste trabalho.

Em primeiro lugar, a linguagem escolhida foi a Python 3, que é considerada a mais popular e conveniente para as aplicações que buscamos, e consequentemente a que possui o maior apoio da comunidade que pesquisa e desenvolve esta área.

Tal linguagem foi projetada com objetivo de priorizar o esforço do programador sobre o esforço computacional. Desta forma, possui foco na legibilidade do código à velocidade ou flexibilidade. Combina uma sintaxe clara e concisa com os recursos nativos e também de pacotes e bibliotecas desenvolvidas por outros, já que o seu método de desenvolvimento é aberto aos desenvolvedores que queiram contribuir com a linguagem.

O módulo chamado *Numpy*, amplamente utilizado no decorrer do trabalho, adiciona ao Python pacotes que oferecem suporte para trabalhar com matrizes multidimensionais, operações matemáticas complexas, e cálculos úteis no ramo

da engenharia. O módulo *Pandas* auxilia na manipulação e análise dos dados, nas manipulações de tabelas extensas e na estrutura dos dados, que estão sempre presentes na manipulação dos conjuntos de dados utilizados para aprendizado profundo. O *Matplotlib* é uma biblioteca de plotagem, e é utilizado como extensão para diversas outros pacotes, como o próprio *Numpy*.

Entre as bibliotecas focadas em aprendizado profundo, temos o *TensorFlow*, que é uma biblioteca de software de código aberto para computação numérica usando gráficos de fluxo de dados, utilizado para aprendizado de máquina e principalmente aprendizado profundo, e está sendo aplicada em larga escala em tarefas tanto em meio acadêmico quanto comercial. Este *framework* permite que se realize computação tanto em CPUs quanto GPUs, além de incluir também a ferramenta chamada *TensorBoard*.

O *TensorBoard* é uma ferramenta cujo objetivo é facilitar o entendimento, a depuração e a otimização dos programas que utilizam TensorFlow, algo extremamente importante quando é realizado o treinamento de redes neurais, por serem complexas e com tempo de treinamento demorado, e portanto é preciso uma visualização mais didática e simplificada possível. Ela opera lendo os *event files* gerados pelo TensorFlow, que contêm dados resumidos do treinamento da rede, que são criados ao requisitar a criação de *callbacks* do *TensorBoard*.

Por fim, *Keras* é uma API de aprendizado de máquina de alto nível, que também é escrita em Python e capaz de rodar utilizando o TensorFlow como *backend*. Esta API torna a implementação de modelos de aprendizado de máquina um processo consideravelmente mais simples, e permite que todas as fases de desenvolvimento destes modelos seja feita de forma direta e rápida, porém com a robustez e eficiência computacional do TensorFlow.

Para a otimização dos vários modelos que serão analisados neste trabalho, é preciso de certo poder computacional para que cada processo de treinamento não tome dias, portanto é preciso recorrer ao poder de computação paralela de uma GPU. Desta forma, foi utilizado a plataforma da Google chamada *Colaboratory*.

A *Colaboratory* é um serviço de computação em nuvem desenvolvido especialmente para desenvolvedores de inteligência artificial, que disponibiliza para o usuário um ambiente de desenvolvimento em Python que funciona diretamente no navegador. Possui já integrado em sua máquina virtual as principais bibliotecas e pacotes utilizados na área, e seu principal benefício é disponibilizar para o usuário uma GPU por tempo limitado. Segue na Tabela 3 as especificações do sistema:

Tabela 3 – Hardware disponibilizado pela plataforma *Colaboratory*.

CPU	GPU	Disco	RAM
Intel(R) Xeon(R) @2.3Ghz	Tesla K80 12GB GDDR5	33 GB	12.6 GB

3.3 ESPECIFICAÇÃO DO MODELO UTILIZADO

O modelo utilizado neste trabalho foi baseado em uma arquitetura definida por Ronneberger et al. (2015), chamada U-net. Este arquitetura foi concebida inicialmente para uso no meio da biomedicina, para detecção de padrões em imagens como raios-x, ressonância magnética, microscopia, e outros.

Para estes tipos de aplicações, tanto o treinamento quanto a detecção através de *bounding box* não gera resul-

tados ideais, pois as áreas de interesse não possuem forma clara e definida, o que pode dificultar a convergência da CNN durante a fase de treinamento, principalmente com poucos dados.

Esta arquitetura foi escolhida por apresentar especificação simples, que apesar de possuir uma implementação descomplicada utilizando os *frameworks* escolhidos para este trabalho, possui resultados robustos e confiáveis.

Como visto na Tabela 2, este modelo apresentou resultados ótimos para o conjunto de dados DeIC-HeLa, que possui somente 20 imagens da célula hela de forma rotulada, obtidas com um microscópio de contraste por interferência diferencial.

Desta maneira, apesar de sua versão original possuir proporções medianas com cerca de 30 milhões de parâmetros, e baixa velocidade de inferência, uma implementação reduzida de tal modelo deve manter sua propriedade de hierarquização de características eficiente.

Será analisado portando o comportamento desta arquitetura reduzida sob diferentes condições a fim de averiguar sua acurácia de detecção e velocidade de inferência.

A CNN implementada usando a arquitetura U-net realiza detecções de objetos na forma de segmentação semântica, em que pixeis que representam o objeto de interesse são detectados pela rede neural e agrupados, representado a forma exata do objeto. Tal modelo permite somente duas classes binárias, portando a imagem de saída que representa a segmentação da entrada também contém somente valores binários.

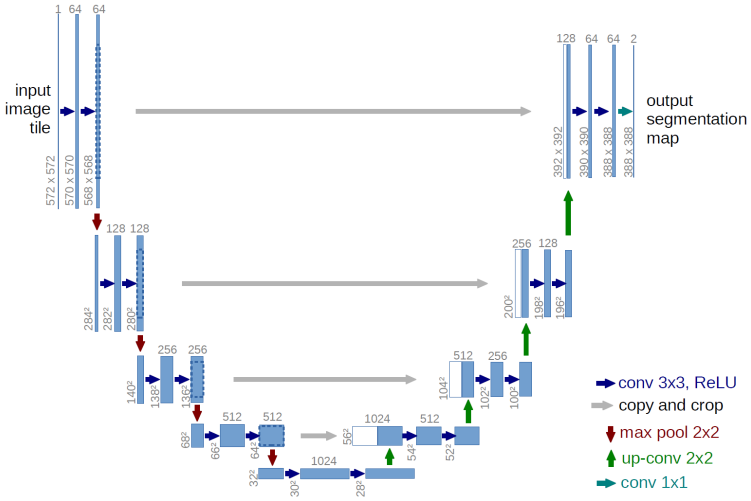
Tal característica representa uma limitação para aplicações que exigem detecção de múltiplas classes, porém há uma grande melhoria em acurácia.

Este modelo de CNN é chamada de *fully convolutional network*-FCN (LONG et al., 2015), ou seja, ela não possui

camadas totalmente conectadas ao final da rede, e sim consiste em um caminho de contração, no lado esquerdo, e um caminho de expansão, no lado direito, também chamado de *encoder-decoder*.

No modelo U-net original, o caminho de contratação segue a arquitetura típica de uma rede convolucional profunda. É composta pela aplicação seguida de duas convoluções com 64 filtros 3x3, cada uma seguida pela função de ativação do tipo ReLU, que foi popularmente introduzida por Krizhevsky et al. (2012), e por último a operação de *maxpooling* 2x2, para redução na resolução dos *feature maps*.

Figura 13 – A arquitetura original U-net. Por não usar a técnica de *padding* durante a amostragem, a saída possui resolução inferior à entrada.



Fonte: Ronneberger et al. (2015)

Este processo se repete duas vezes, onde após cada etapa de amostragem, é dobrado o número de filtros das camadas. A cada duas convoluções no caminho de expansão

é seguida pela convolução transposta, através de um filtro 2x2 que aumenta a resolução do *feature map* e divide pela metade o número de camadas dos filtros, para em seguida ser executada uma concatenação com o *feature map* do caminho de contração de mesma resolução. Esta concatenação é caracterizada pela união entre a quantidade de *feature maps* presentes nas camadas anteriores.

Por fim, após a concatenação final, que pode ser visualizada na Figura 13, o caminho é seguido pela convolução de dois filtros 3x3, cada uma delas seguida por uma função ReLU.

Na camada final, é executada uma convolução com filtro 1x1 e função de ativação sigmoide, que é usado para mapear as 64 camadas de *feature maps* em uma única, que possui resolução inferior se comparada com a imagem de entrada, devido a não existência de *padding*, algo corrigido na implementação da U-net deste trabalho.

Esta imagem de saída contêm os valores binários que representam a segmentação para cada pixel da imagem original de entrada. No total, a rede original é constituída de 23 camadas convolucionais. Na Tabea 4 se encontra a descrição detalhada das arquiteturas Unet-8 e Unet-16.

No modelo original da U-net, é utilizada como função de custo a função entropia cruzada binária, e para o treinamento estocástico da rede é executada a implementação do método Gradiente Estocástico Descendente com *momentum* igual a 0,99.

Na implementação do modelo deste trabalho, serão realizadas alterações, em que além da função de custo *Intersection Over Union*, será utilizado um método de treinamento que utiliza otimização estocástica chamado Adam(KINGMA; BA, 2014).

Este modelo foi implementado utilizando o API Keras, que permite uma maior simplificação de todo o processo. O

modelo U-net original é consideravelmente grande, com precisamente 31 378 945 parâmetros, tornando tanto o processo de treinamento quanto a execução extremamente demorado. Por isto, foram utilizados menos filtros nas camadas da rede.

Portando, foram concebidas duas implementações diferentes do modelo U-net, com o objetivo de comparar a diferença de desempenho de ambas. A principal e única diferença destes dois modelos é a quantidade de *kernels* presentes em suas camadas.

Desta forma, ao invés dos 64 *kernels* na primeira camada de convolução(que no lado de contração dobra a cada camada), foram implementados dois modelos com 8 e 16 *kernels* na primeira camada, chamados neste trabalho de Unet-8 e Unet-16, respectivamente.

Para manter a simetria da rede, o lado de expansão segue a mesma lógica, e portanto temos em sua saída a mesma quantidade de *kernels* que a entrada.

Ambos os modelos são idênticos, com a única diferença sendo a quantidade de *kernels* por camada. Assim, será possível observar a diferença que tal mudança acarretará nos resultados finais, e ambos os modelos serão analisados através de seus resultados de treinamento e predição.

3.4 TREINAMENTO DA CNN

Para o treinamento proposto, foram criados 6 subconjuntos de dados, cada um com quantidades variadas de imagens. São compostos por imagens com o tipo de formato RGB, ou seja, com 3 canais de cores.

Estes subconjuntos de dados são compostos por: 250, 500, 1000, 2000, 4000 e 7000 imagens cada. Desta maneira, será possível analisar a capacidade dos modelos Unet-8 e Unet-16 em aprender as características necessárias para um

Tabela 4 – Descrição da arquitetura das redes Unet-8 e Unet-16

id	Camada	Qtd. kernels		Resolução	Padding	Stride	Ativação
		Unet-8	Unet-16				
1	Convolução	8	16	3x3	SAME	1x1	ReLU
2	Convolução	8	16	3x3	SAME	1x1	ReLU
3	Maxpooling			2X2		2x2	
4	Convolução	16	32	3x3	SAME	1x1	ReLU
5	Convolução	16	32	3x3	SAME	1x1	ReLU
6	Maxpooling			2X2		2x2	
7	Convolução	32	64	3x3	SAME	1x1	ReLU
8	Convolução	32	64	3x3	SAME	1x1	ReLU
9	Maxpooling			2X2		2x2	
10	Convolução	64	128	3x3	SAME	1x1	ReLU
11	Convolução	64	128	3x3	SAME	1x1	ReLU
12	Maxpooling			2X2		2x2	
13	Convolução	128	256	3x3	SAME	1x1	ReLU
14	Convolução	128	256	3x3	SAME	1x1	ReLU
15	Convolução Transposta	64	128	2x2	SAME	2x2	ReLU
16	**Concatenação das camadas 15 e 11**						
17	Convolução	64	128	3x3	SAME	1x1	ReLU
18	Convolução	64	128	3x3	SAME	1x1	ReLU
19	Convolução Transposta	32	64	2x2	SAME	2x2	ReLU
20	**Concatenação das camadas 19 e 8**						
21	Convolução	32	64	3x3	SAME	1x1	ReLU
22	Convolução	32	64	3x3	SAME	1x1	ReLU
23	Convolução Transposta	16	32	2x2	SAME	2x2	ReLU
24	**Concatenação das camadas 23 e 5**						
25	Convolução	16	32	3x3	SAME	1x1	ReLU
26	Convolução	16	32	3x3	SAME	1x1	ReLU
27	Convolução Transposta	8	16	2x2	SAME	2x2	ReLU
28	**Concatenação das camadas 27 e 2**						
25	Convolução	8	16	3x3	SAME	1x1	ReLU
26	Convolução	8	16	3x3	SAME	1x1	ReLU
27	Convolução	1	1	1x1	SAME	1x1	sigmoide

bom desempenho, pois a capacidade de um modelo de rede neural em generalizar as informações está diretamente relacionada com a quantidade de informações que lhe são fornecidos durante o treinamento.

A capacidade de aprendizado das redes neurais também está diretamente relacionada com a sua arquitetura e a quantidade de parâmetros treináveis. Desta maneira, mesmo utilizando conjuntos de dados consideravelmente grandes, um modelo simples pode apresentar pouca melhoria significativa durante o treinamento.

Isto ocorre pois o modelo atingiu sua capacidade máxima de generalização das características apresentadas pelo conjunto de dados. Tal fenômeno é chamado de *underfitting*.

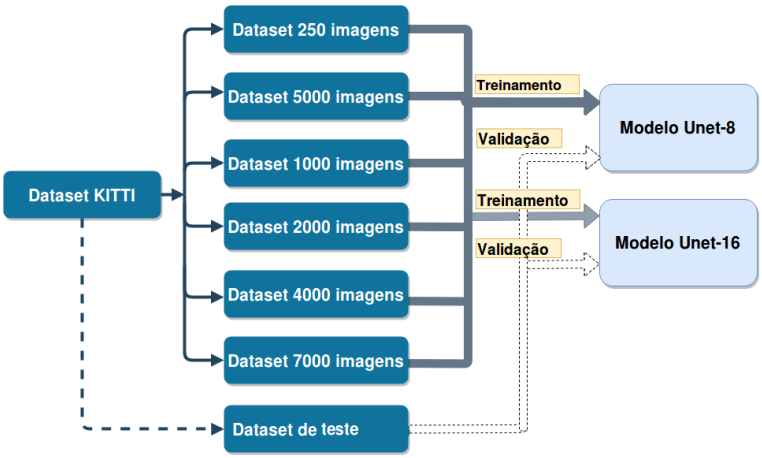
Por outro lado, existem modelos que possuem grande capacidade de aprendizado, com milhões de parâmetros treináveis e com eficiente hierarquização das características necessárias para generalização das informações que representam o rótulo que deseja-se detectar. Tais modelos, caso não sejam alimentados com a quantidade necessária de informações durante o treinamento, podem apresentar o fenômeno de *overfitting*.

Um modelo que apresenta *overfitting*, mostra resultados péssimos durante a fase de testes. Isto ocorre pois dada a grande capacidade de aprendizagem, e a pouca quantidade de informações disponíveis, o modelo realiza a "memorização" das informações durante o treinamento.

Desta forma, a função de custo apresenta valores baixíssimos, e grande acurácia caso o modelo seja testado com dados contidos no conjunto de dados de treinamento, porém resultados pobres quando lhe é apresentado alguma informação que nunca que foi vista. Com a memorização do conjunto de dados de treinamento, acaba-se perdendo a habilidade de generalização das informações que representam o objeto de interesse.

Os 6 subconjuntos de dados que serão utilizados para treinar os modelos deste trabalho possuem quantidades de imagens variadas, pois assim será possível analisar o comportamento da rede de acordo com a quantidade de dados apresentados, e consequentemente analisar a ocorrência tanto de *overfitting*, quanto *underfitting*.

Figura 14 – Fluxo de dados para cada treinamento.



Fonte: Elaborado pelo Autor.

O API Keras permite que lhe sejam fornecidos tanto o conjunto de dados de treinamento quanto o de teste, apresentando resultados simultâneos do treinamento e de teste do modelo.

Foi selecionado um conjunto de dados de teste fixo de 480 imagens, nunca antes alimentadas ao modelo durante o treinamento. Estas imagens permitirão que seja apresentada o verdadeiro desempenho do modelo. Foi determinado um tamanho fixo para teste para manter o mesmo critério de avaliação para todos os modelos otimizados.

A visão geral do precedimento de treinamento para cada conjunto de dados pode ser visualizado na Figura 14.

A função de custo utilizada no trabalho original (RONNEBERGER et al., 2015) é a chamada entropia cruzada binária. Esta é a versão direcionada da função entropia cruzada, que é utilizada quando há somente dois rótulos(objeto de interesse e plano de fundo), e apresenta leve vantagem computacional em relação a função de custo original por apresentar uma saída binária.

A expressão matemática para tal função de custo pode ser observada na equação 3.1, utilizando as convenções estabelecidas no capítulo 2.1.2.

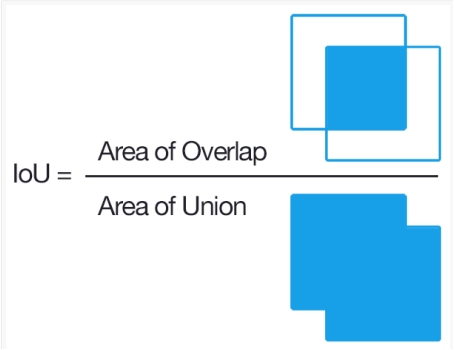
$$E = -\frac{1}{n}\sum_{i}^{n}[y^{(i)}log(y_i) + (1 - y^{(i)})log(1 - y_i)] \tag{3.1}$$

Este trabalho, todavia, utilizará outra função de custo, a chamada *Intersection Over Union* - IoU, também chamada de coeficiente de similaridade Jaccard.

Esta função é mais popularmente utilizada em sua versão em que se comporta como métrica, porém para a aplicação desejada, esta função de custo apresenta resultados mais satisfatórios que a entropia cruzada binária.

A função IoU é principalmente empregue quando é desejado calcular o grau de sobreposição e similaridade de dois *bounding boxes*(BEERS, 2018). Apesar de o objetivo deste trabalho ser a segmentação dos objetos de interesse, suas máscaras foram representadas a partir das formas retangulares de seus rótulos.

Figura 15 – Representação gráfica do como é realizado o cálculo da métrica *Interection Over Union*.



Fonte: Beers (2018).

Como pode ser observado na Figura 15 e na Equação 3.2, tal função é representada pela divisão da intersecção da detecção retangular, pela área de sua união.

$$E = 1 - \frac{|y^{(i)} \cap y_i|}{|y^{(i)} \cup y_i|} \tag{3.2}$$

O método de otimização do modelo que será utilizado é o *Adam*. Este método utiliza uma variação do gradiente estocástico para realizar a minimização da função de custo e a consequente otimização do modelo. O nome Adam é derivado de *adaptive moment estimation*, ou "estimativa de momento adaptativo". Segundo os autores (KINGMA; BA, 2014), é a combinação de outros dois métodos de otimização estocástica: o RMSprop e o AdaGrad.

A taxa de aprendizagem inicial foi definida como 0,001. Tal valor foi determinado através de observação prática, por garantir convergência e velocidade durante o treinamento. Outros hyperparâmetros importantes para o otimizador Adam,

como o β_1 e β_2 e o épsilon , receberam valores padrão fornecidos pelo API, que são 0,9, 0,999 e 10^{-8}, respectivamente.

Foram utilizados os valores padrão pois este modelo otimizador é bastante robusto em relação a estes parâmetros(KINGMA; BA, 2014) e tais valores são amplamente utilizados pela comunidade.

Umas das ferramentas utilizadas disponibilizadas pelo API Keras é a de redução da taxa de aprendizado durante o treinamento. Esta ferramenta permite a redução a taxa de aprendizado por um fator de 5, sempre que a função de custo mostra pouca alteração e encontra-se estagnada. Isto pode ocorrer tanto porque o modelo atingiu seu limite de aprendizado, ou então a atualização dos parâmetros está sendo afetada pela alta taxa de aprendizagem, impedindo o modelo atingir o valor de erro mínimo da função de custo.

As imagens originais do conjunto de dados KITTI possuem a resolução de 1242x375 pixeis. Apesar de resoluções maiores possuírem mais informações, o que é algo essencial para o treinamento de uma rede neural, é preciso realizar um *trade-off* com outro parâmetro muito importante para a aplicação destas redes, que é a duração do treinamento e a limitação da memória computacional disponível. Portanto, optou-se por realizar uma redução na dimensão das imagens, convertendo-as para a resolução de 560x176 pixeis.

Tal resolução foi determinada por, além de manter a essência da imagem, mantêm a proporção da mesma(≈ 3.2) evitando a deformação da imagem.

As imagens passaram por outro pré-processamento, a normalização. Esta normalização for realizada simplesmente dividindo cada valor presente na imagem por 255, que é o valor máximo possível para cada canal de pixel. Desta maneira, todos os pixeis passam a estar entre os valores 0 e 1.

Este pré-processamento garante melhor convergência

durante a fase de treinamento, graças a computação dos gradientes com valores menores, evitando a explosão de alguns gradientes durante o *backpropagation*, o que prejudica consideravelmente o resultado final da rede neural.

4 RESULTADOS

Nesta seção, primeiramente será realizada uma análise do treinamento referente ao modelo Unet-8, e logo em seguida sua versão mais complexa, a Unet-16, utilizando diferentes conjuntos de dados, conforme discutido na Seção 3.1.

O acompanhamento do progresso de otimização durante a fase de treinamento de redes neurais artificiais permite uma visualização do processo de aprendizagem do modelo, através do uso métricas pré-estabelecidas, além da função de custo que é calculada obrigatoriamente. Enquanto que uma função de custo preferencialmente converge para o valor mínimo possível, as métricas em geral convergem para um valor unitário, que indica o grau da acerto da rede neural durante o treinamento.

As métricas utilizadas neste trabalho, foram o cálculo de acurácia, e a métrica IoU(*Intersection Over Union*), aplicadas tanto sobre o conjunto de dados de treinamento, quanto o de teste.

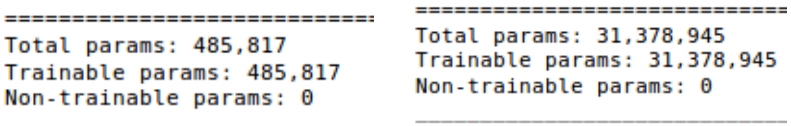
Utilizando a ferramenta *Tensorboard*, é possível visualizar o progresso de todos estes parâmetros variáveis e dependentes do andamento da otimização do modelo. Tais valores auxiliam a compreender e analisar o desempenho das CNNs que serão otimizadas neste trabalho.

4.1 TREINAMENTO DO MODELO UNET-8

A arquitetura reduzida da rede neural U-net, chamada neste trabalho de Unet-8, foi implementada de forma razoavelmente diferente da sua versão original, que é mais comumente utilizada na biomedicina(RONNEBERGER et al., 2015).

Além da redução da quantidade de *kernels* por camada,

Figura 16 – Quantidade de parâmetros treináveis presentes nas redes Unet-8(a) e U-net original(b).



(a) (b)

Fonte: Elaborado pelo Autor.

cujo impacto na quantidade de parâmetros treináveis pode ser observado na Figura 16, as duas arquiteturas diferem em suas funções de otimização e função de custo e inicialização de pesos. Tais modificações foram consideradas necessárias para o melhor desempenho da rede neural.

O treinamento das redes neurais neste trabalho foram projetadas para receberem lotes de 24 imagens por vez durante o treinamento, parâmetro também chamado de *batch*, e com épocas variáveis dependendo do tamanho do conjunto de dados.

Não foi estabelecido um valor único de épocas durante o treinamento pois observou-se durante o andamento deste trabalho que CNNs otimizadas com menos imagens necessitam de mais épocas para convergir para valor da função de custo mínimo.

Para isto, utilizou-se umas das funções de *callback* do API Keras, que é a de monitoramento das métricas ou funções de custo, e caso o parâmetro escolhido apresente estagnação, o treinamento é encerrado. Desta forma, como é desejado que o modelo atinja sua capacidade máxima de treinamento com o próprio conjunto de dados de treinamento, o parâmetro escolhido para monitoramento é a própria função

Tabela 5 – Hyperparâmetros e outras informações sobre o treinamento da Unet-8.

Hyperparâmetro ou Função	Valor
Método de otimização	Adam
Taxa de aprendizagem	0,001
β_1	0,99
β_2	0,999
Epsilon	0
Função de custo	IoU
batch	24
Patience	5
Métrica 1	Coef. IoU
Métrica 2	Acurácia
Resolução	560x176
Num. canais	3

de custo da CNN.

O parâmetro que determina o momento de parada é chamado de *patience*, e seu valor representa o número de épocas sem alteração significativa do valor monitorado(função de custo) que deve existir antes da parada forçada do treinamento.

Desta forma, ao deixar de monitorar os resultados do conjunto de dados de teste, o modelo pode apresentar *overfitting* caso não receba dados suficientes durante a otimização, algo que é desejado observar neste trabalho.

Os principais hyperparâmetros estão na Tabela 5, assim como outras informações pertinentes sobre o treinamento da CNN.

O treinamento foi realizado carregando todas as imagens diretamente na memória, criando uma única matriz compacta. Este processo levou aproximadamente 9 minutos para o conjunto de dados de 250 imagens, e 3 horas e 10

minutos para o de 7000 imagens.

Após o carregamento em si das imagens, inicia-se o treinamento da CNN de acordo com as especificações, que durou cerca de 9 minutos no caso mais rápido, e 1 hora e 7 minutos foi o treinamento com a maior duração. Estas informações podem ser observadas na Tabela 6.

Segue na Figura 17 os resultados de treinamento de acordo com o *Tensorboard*, para cada uma das métricas discutidas, tanto para etapa de treinamento quanto teste.

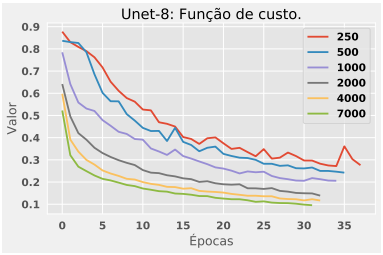
Fica claro, observando os gráficos da Figura 17, que os resultados durante o treinamento são superiores aos de validação, pela óbvia razão de os parâmetros da modelo serem otimizados em função do conjunto de dados de treinamento.

Através da análise dos resultados de treinamento é possível notar características adquiridas pelo modelo durante a otimização que podem muitas vezes causar um mal desempenho, ou então uma má abordagem do tema.

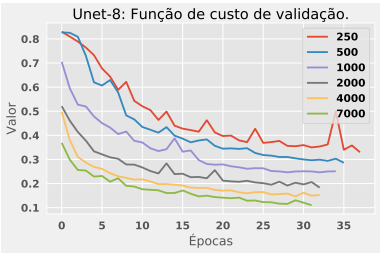
Graças à menor complexidade do modelo Unet-8, a rede apresentou somente um tênue *overfitting*, que pode ser observado com maior gravidade nos modelos treinados com os conjuntos de dados de 250 e 500 imagens, ao notar a estagnação do progresso das métricas de teste após certo ponto da otimização, e, consequentemente a diferença dos valores finais. Tal fenômeno não chegou a causar redução significativa do desempenho da rede neural, como pode ser observado através da predições de imagens logo após.

Observa-se que a métrica da acurácia mostrou valores significativamente altos logo após os momentos iniciais da otimização, tanto para treinamento quanto para teste. Esta característica, apesar de não ser muito útil nos momentos finais por mostrar pouquíssima diferença entre os valores, apresenta grande variação de resultado no início da convergência do modelo.

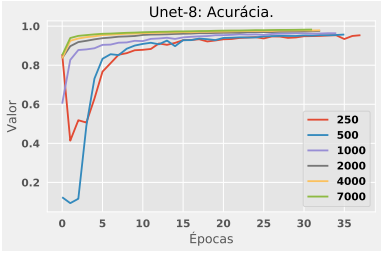
Figura 17 – Resultados referentes ao treinamento do modelo Unet-8 com os 6 *datasets* propostos.



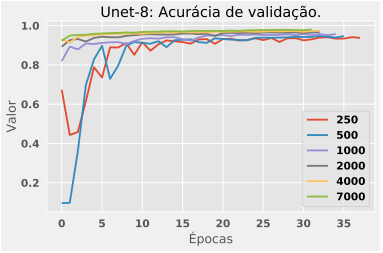
(a) Função de custo.



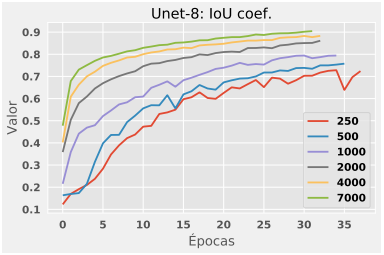
(b) Função de custo de teste.



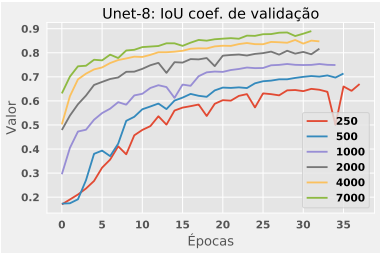
(c) Acurácia.



(d) Acurácia de teste.



(e) Coef. IoU.



(f) Coef IoU de teste.

Tabela 6 – Valores de duração do carregamento de imagens, duração do treinamento, e épocas durante a otimização do modelo Unet-8 antes da estagnação.

Qtd. imagens	Carregamento[s]	Treinamento[s]	Épocas
250	521	447	40
500	1210	557	36
1000	2312	753	33
2000	4478	1352	32
4000	7921	2374	31
7000	11501	4058	31

Figura 18 – Predições utilizando 2 imagens do *dataset* de treinamento com o modelo Unet-8 otimizado com 250 imagens em (a) e (c), e 7000 imagens em (b) e (d).

(a) (b)

(c) (d)

A partir da Tabela 6, nota-se que apesar da duração do processamento de cada época aumentar conforme aumente o número de imagens do conjunto de dados, existe uma clara redução da quantidade de épocas necessárias par atingir a convergência do modelo. Tal razão se deve ao fato de que

cada época fornece maior quantidade de dados para a CNN, tornando o aprendizado mais robusto graças à uma hierarquização de características mais completa se comparado com conjuntos de dados menores.

Desta forma, é de se esperar que para tais resultados de treinamento, os modelos gerados apresentem diferença visível a olho nu quando utilizados pra realizar predições com imagens de ambos os conjuntos de dados de treinamento e de teste, como pode ser visto a seguir.

O tempo de execução do modelo com arquitetura Unet-8, utilizando o hardware disponibilizado na Tabela 3, é de 0,0127 segundos.

Figura 19 – Predições utilizando 2 imagens do *dataset* de teste com o modelo Unet-8 otimizado com 250 imagens em (a) e (c), e 7000 imagens em (b) e (d).

(a) (b)

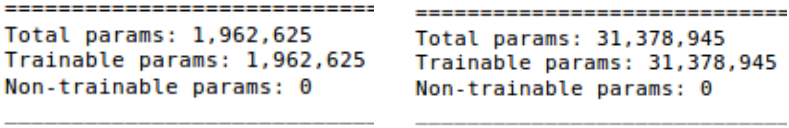
(c) (d)

4.2 TREINAMENTO DO MODELO UNET-16

A arquitetura estabelecida para o modelo Unet-16 possui obviamente maior potencial de aprendizado que a Unet-8.

Tal fator pode ser observado na Figura 20, em que apesar da rede possuir quase 16 vezes menos parâmetros treináveis que a sua arquitetura original, acaba contendo aproximadamente 4 vezes mais parâmetros que o modelo Unet-8.

Figura 20 – Quantidade de parâmetros treináveis presentes nas redes Unet-8(a) e U-net(b) original.



(a) (b)

Fonte: Elaborado pelo Autor.

Como mencionado na Seção 3.3, ambos os modelos possuem arquiteturas idênticas, com exceção da profundidade das suas camadas. Além isto, os seus hyperparâmetros também são bastante semelhantes, exceto o valor de *patience*, que determinará a parada do treinamento da rede neural. Este valor passou de 3 para 5, devido a característica da rede neural de convergir com mais épocas se comparado com a sua versão menos complexa, o que pode ser observado na Tabela 8.

Os resultados de treinamento mostraram superioridade da rede Unet-16 em relação a Unet-8, que torna-se mais clara conforme o número de imagens do conjunto de dados é incrementado.

Para o treinamento com somente 250 imagens, foi obtido, para a métrica IoU durante a fase de teste, o valor de 0,632 e 0,742, para Unet-8 e Unet-16, respectivamente. Enquanto que para 7000 imagens, a mesma métrica atingiu valores de 0.890 e 0,954.

Tabela 7 – Hyperparâmetros e outras informações sobre o treinamento da Unet-16.

Hyperparâmetro ou Função	Valor
Método de otimização	Adam
Taxa de aprendizagem	0,001
β_1	0,99
β_2	0,999
Epsilon	0
Função de custo	IoU
batch	24
Patience	7
Métrica 1	Coef. IoU
Métrica 2	Acurácia
Resolução	560x176
Num. canais	3

Por sua vez, observando os resultados da rede neural Unet-16, nota-se menor diferença entre seus valores de treinamento, enquanto que para teste, há grande diferença de resultados para o modelo otimizado com tamanhos diferentes de *dataset*. Tal característica não se mostrou de forma tão clara para a Unet-8.

Para a métrica IoU do treinamento com 250 imagens, foram obtidos os valores de 0,734 e 0,643, para as redes Unet-16 e Unet-8, respectivamente. Para o treinamento de 7000 imagens, os valores da mesma métrica citada foram 0,977 e 0.913, na mesma ordem.

A acurácia da fase de teste apresentou valores altos, que acabam não correspondendo aos resultados observados nas predições das Figuras 22 e 23. Para os dois extremos, 250 e 7000 imagens, foram obtidos os valores de 0,9862 e 0,9882 para a Unet-16 e Unet-8. Tais valores não represen-

tam fielmente a realidade, apesar de possuírem sua utilidade no início do treinamento.

Tabela 8 – Valores de duração do carregamento de imagens, duração do treinamento, e épocas durante a otimização do modelo Unet-16 antes da estagnação.

Qtd. imagens	Carregamento[s]	Treinamento[s]	Épocas
250	521	1313	55
500	1210	1335	53
1000	2312	2130	53
2000	4478	3681	52
4000	7921	6840	52
7000	11501	12666	51

Percebe-se que quanto maior a quantidade de imagens, menor o ruído nas métricas e na função de custo durante o treinamento.

Esta oscilação momentânea dos resultados se deve ao fato de que a rede neural ainda não atingiu grau de aprendizado elevado. Desta forma, quando há dados com características novas apresentadas durante o treinamento, a rede apresenta alto valor de perda.

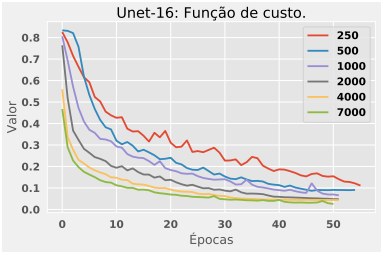
Tal fenômeno, em casos mais sérios, pode impedir os modelos de convergirem de forma eficiente, porém mesmo no caso de somente 250 imagens, as redes neurais não só convergiram para um valor mínimo aceitável, mas mostraram algumas predições plausíveis, apesar da falta de precisão do contorno dos objetos de interesse.

Os resultados presentes na Figura 21 mostram que as arquiteturas Unet-16 e Unet-8 são robustas, por convergirem para valores baixos de perda mesmo com pouquíssimas imagens, algo que muitos modelos não atingem.

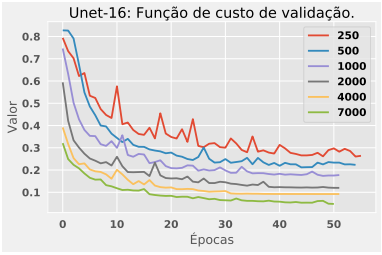
Em consequência de ser 4 vezes maior que a Unet-8,

este modelo realiza cada predição em 0,0204 segundos em média, ao utilizar o hardware disponibilizado pelo *Colaboratory*, observado na Tabela 3.

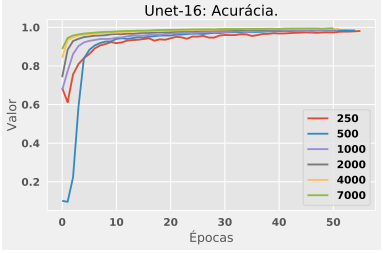
Figura 21 – Resultados referentes ao treinamento do modelo Unet-16 com os 6 *datasets* propostos.



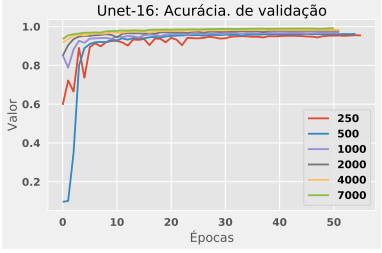
(a) Função de custo.



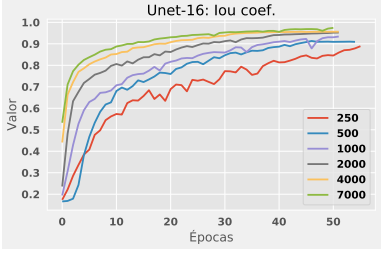
(b) Função de custo de teste.



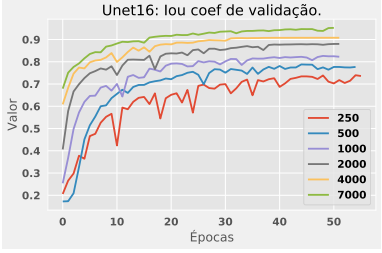
(c) Acurácia.



(d) Acurácia de teste.



(e) Coef. IoU.



(f) Coef. IoU de teste.

Fonte: Elaborado pelo Autor.

Figura 22 – Predições utilizando 2 imagens do *dataset* de treinamento com o modelo Unet-16 otimizado com 250 imagens em (a) e (c), e 7000 imagens em (b) e (d).

(a) (b)

(c) (d)

Fonte: Elaborado pelo Autor.

Figura 23 – Predições utilizando 2 imagens do *dataset* de teste com o modelo Unet-16 otimizado com 250 imagens em (a) e (c), e 7000 imagens em (b) e (d).

(a) (b)

(c) (d)

Fonte: Elaborado pelo Autor.

4.3 ANÁLISE DOS RESULTADOS

É possível observar, nas imagens geradas pelas predições, que a segmentação dos objetos de interesse mostra pouca acurácia quanto aos seus contornos extatos, apresentando muitas vezes uma segmentação visivelmente retangular. Esta característica está visivelmente ligada ao fato de a máscara que representa o rótulo da imagem é gerada a partir de *bouding boxes* concebidos pela equipe que criou o *dataset* KITTI 2012(GEIGER et al., 2012).

Graças a este artifício empregado, a CNN possui a característica manter a forma geométrica dos rótulos ao invés do objeto de interesse, apesar de claramente ter aprendido com sucesso a hierarquização de características que definem o objeto, e consequentemente realizar a detecção e localização dos mesmos.

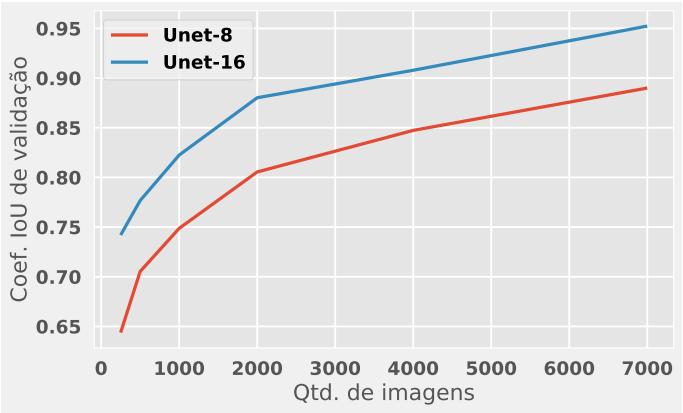
Apesar de empregada no meio acadêmico(YU et al., 2016), o treinamento de CNNs utilizando tal técnica de geração de máscaras, utilizando rótulos representados por *bounding boxes*, mostrou resultados que não representam em muitos casos a segmentação verdadeira do objeto do interesse, algo importante para a aplicação em veículos autônomos.

Esta característica foi exacerbada pela função de custo IoU, que apesar de efetiva, condiciona ainda mais a rede neural durante o treinamento a manter a forma retangular de suas predições.

Para que ocorra a segmentação do contorno verdadeiro, é preciso portanto a otimização da CNN utilizando um *dataset* com máscaras que representam este contorno, e não a partir de *bounding boxes*. Contudo, os modelos treinados mostraram ótimo desempenho para a detecção e localização dos objetos de interesse, atingindo a acurácia de 0,991 com o melhor modelo, o Unet-16 treinado com 7000 imagens.

Os resultados apresentados neste trabalho confirmam

Figura 24 – Melhores resultados da métrica IoU, comparando ambos os modelos.



Fonte: Elaborado pelo Autor.

os conceitos populares abordados por todos os autores da área. O modelo U-net com mais *kernels* possui uma maior capacidade de reter informações, e consequentemente isto reflete nos resultados.

Por outro lado, o gráfico de crescimento do desempenho das CNNs mostram que ambos os modelos possuem a capacidade de melhoria dos seus resultados. Isto pode ser concluído graças a característica assintótica da curva mostrada na Figura 24. Tal fato mostra que ambos os modelos não apresentam *underfitting*, tornando claro que os modelos necessitam de mais dados.

Como mencionado, as redes neurais implementadas e otimizadas neste trabalho resultaram em uma imprecisa detecção de contornos, porém ótima detecção e localização dos objetos de interesse quando visto por um lado não mais voltado para aplicação em veículos autônomos.

Tabela 9 – Alguns dos modelos de rede neural que realizam segmentação semântica.

Rede	Multi?	Dataset	Nº de imagens	Parâm. [M]	Tempo [ms]	IoU (média)
FCN-VGG16	**Sim**	Pascal VOC 2011	11 530	134	210	56,0
FCN-GoogLeNet	**Sim**	Pascal VOC 2011	11 530	6	59	42,5
SegNet	**Sim**	SUN RGB-D	10335	14,7	52,6	60,1
DeepLab-CRF	**Sim**	Pascal VOC 2012	11 530	65,1	125	71,6
Dilated Conv.	**Sim**	**Microsoft COCO**	**328 000**	-	-	71,3
Unet	Não	DIC-HeLa	20	31,3	-	77,3
Prop. Unet-8	Não	KITTI Vision 2012	7 480	**0,48**	**12,7**	89,1
Prop. Unet-16	Não	KITTI Vision 2012	7 480	1,96	20,4	**95,2**

Ao observar a Tabela 9, nota-se a princípio que os modelos otimizados neste trabalho atingiram resultados excelentes se comparado com os outros modelos populares da área, mesmo possuindo o objetivo de serem simples e velozes.

Porém, deve-se esclarecer que os modelos abordados na tabela possuem foco em detecção de múltiplas classes, enquanto que a Unet-8 e Unet-16 possuem resultados para detecção de classes binárias.

Esta técnica torna o modelo muito mais eficiente e preciso na detecção dos objetos de interesse, apesar de perder a habilidade de detecção de múltiplas classes.

As redes apresentaram tempo médio de execução para predições de 0,0127(Unet-8) e 0,0204(Unet-16) segundos, correspondendo aproximadamente à 75 e 50 quadros por segundo. Tais resultados mostram que é possível não só atin-

gir, mas ultrapassar os 24 quadros por segundo, que era a meta de execução em tempo real.

Desta forma, com o hardware adequado, tais redes possuem velocidade de execução compatível com aplicações que necessitam de detecções em tempo real.

Por outro lado, graças a falha de detecção precisa de contornos e de objetos à longa distância, as redes neurais convolucionais propostas nestes trabalho necessitam de melhorias, porém evitando ao máximo perder eficiência computacional.

5 CONCLUSÃO

A análise dos modelos propostos neste trabalho apresentaram resultados que podem ser úteis para os usuários de aprendizado profundo com foco em segmentação semântica, que não possuem grande poder computacional para treinar modelos complexos, assim como poucos dados de treinamento.

5.1 CONSIDERAÇÕES FINAIS

Ao início do trabalho, através da fundamentação teórica, mostrou-se como toda a área de conhecimento que envolve o aprendizado profundo vem caminhando a passos largos, mostrando técnicas, modelos, conceitos e práticas que mudam a cada mês que passa. Estas mudanças tornam muito do que já foi desenvolvido obsoleto, assim como resgatam antigos conceitos abandonados ao abordá-los de uma maneira diferente.

Os métodos de detecção de objetos que envolvem segmentação semântica apresentam modelos mais lentos e com menor acurácia do que aqueles que realizam classificação, ou detecção e localização através de *bounding boxes*. Portanto, a comunidade acadêmica procura avidamente aperfeiçoar tal área, que é vital para todo o setor de automatização automobilística.

No capítulo em que é apresentado a rede neural convolucional proposta neste trabalho, foi explicada a decisão de redução da complexidade do modelo original da U-net, e a implementação de duas arquiteturas similares, chamadas de Unet-8 e a Unet-16, ao diminuir a quantidade de filtros em cada camada convolucional. Desta forma, reduziu-se em

dezenas de vezes a capacidade de aprendizado da rede. O objetivo desta decisão é a de encontrar um modelo com a robustez da U-net, porém que ocupe menos espaço na memória e com maior velocidade de inferência.

A incrementação na quantidade de imagens no conjunto de dados utilizado apresentou uma clara melhoria dos resultados. O maior conjunto de dados, com 7000 imagens, ainda não é suficiente para alcançar o máximo aprendizado oferecido pelas redes, principalmente da Unet-16.

Apesar disto, maiores quantidades de dados acarretariam em uma maior utilização de memória computacional disponível, dificultando a utilização destes métodos por usuários comuns.

Por outro lado, ambos os modelos apresentaram robustez quando foram otimizados com poucas imagens, algo que era esperado, pois tal rede foi originalmente implementada para ser aplicada na biomedicina, esta que nem sempre possui grandes conjuntos de dados rotulados disponíveis, dependendo da aplicação.

A velocidade de execução é umas das principais características da rede proposta, onde mesmo o modelo Unet-16, que possui quase 4 vezes mais parâmetros e com resultados superiores, apresentou velocidade de execução próxima dos 50 quadros por segundo.

Pelos resultados obtidos, a rede neural proposta não possui desempenho suficiente para aplicação em veículos autônomos, tanto por sua falta de acurácia na detecção de contornos e de objetos à distância, quanto pela característica de combinar todos os veículos em uma única classe, algo que reduz o poder de tomada de decisão da IA do automóvel.

Estes resultados comprovam os conceitos básicos que envolvem a área de aprendizado profundo, que é a de melhoria expressiva de desempenho quando existe uma incrementação na quantidade de dados em parceria com utilização de

modelos complexos. Para que seja possível tal parceria, é preciso essencialmente grande poder computacional, principalmente na forma de GPUs de última geração.

Todos os códigos utilizados neste trabalho se encontram na plataforma de hospedagem de código-fonte Github[1].

5.2 SUGESTÕES PARA TRABALHOS FUTUROS

- A primeira indicação é, em parceria com um maior poder computacional, realizar novos treinamentos utilizando *datasets* consideravelmente maiores, com dezenas de milhares de imagens.

- Utilizar outras arquiteturas que também são classificadas como FCN, como a SegNet(JÉGOU et al., 2017a), ou DenseNet(JÉGOU et al., 2017b), que apresentam resultados superiores à U-net, porém com implementação mais complexa.

- Utilizar conjuntos de dados voltados especialmente para a segmentação semântica, a fim de reduzir a falta de acurácia da detecção de bordas, além de reduzir o ruído na hora da otimização dos modelos.

- Realizar pré-processamento das imagens do *dataset*, como por exemplo *data augmentation*, em que imagens extras são geradas através da originais, ao inverter, rotacionar, ou mudar o formato de cores das mesmas. Isto garante invariância em inúmeros parâmetros, como por exemplo translação, iluminação e oclusão do objeto de interesse.

- Adicionar camadas de regularização, como a L2, *dro-*

[1] https://github.com/LucasLopesTrindade/TCC

pout, ou *batch normalization*, que tornarão a rede neural menos influenciada pelo *overfitting*.

- Incluir métodos de cálculo de distância dos objetos de interesse em relação à câmera, utilizando possivelmente mapas de disparidade.

REFERÊNCIAS

BAHA, N. Real-time obstacle detection approach using stereoscopic images. *International Journal of Information Engineering and Electronic Business*, MECS Publisher, v. 6, n. 1, p. 42–48, feb 2014.

BARRETO, J. M. *Introdução às Redes Neurais Artificiais*. www.inf.ufsc.br/ barreto/tutoriais/Survey.pdf: Departamento de Informática e Estatística, 2002.

BEERS, F. van. Using intersection over union loss to improve binary image segmentation. 2018.

CHEN, L.-C.; PAPANDREOU, G.; KOKKINOS, I.; MURPHY, K.; YUILLE, A. L. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 40, n. 4, p. 834–848, 2018.

DUMOULIN, V.; VISIN, F. A guide to convolution arithmetic for deep learning. 2016.

GEIGER, A.; LENZ, P.; URTASUN, R. Are we ready for autonomous driving? the kitti vision benchmark suite. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2012.

GIRSHICK, R. Fast r-cnn. abr. 2015.

GIRSHICK, R.; DONAHUE, J.; DARRELL, T.; MALIK, J. Rich feature hierarchies for accurate object detection and semantic segmentation. nov. 2013.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.

HAN, F.; SHAN, Y.; CEKANDER, R.; SAWHNEY, H. S.; KUMAR, R. A two-stage approach to people and vehicle detection with hog-based svm. In: *Performance Metrics for Intelligent Systems 2006 Workshop*. [S.l.: s.n.], 2006. p. 133–140.

HAYKIN, S. *Redes neurais: princípios e prática*. [S.l.]: Bookman Editora, 2007.

HE, K.; GKIOXARI, G.; DOLLÁR, P.; GIRSHICK, R. B. Mask r-cnn. *CoRR*, abs/1703.06870, 2017.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. 2015.

HEATON, J.; POLSON, N.; WITTE, J. H. Deep learning in finance. *arXiv preprint arXiv:1602.06561*, 2016.

HUANG, Y.; LIU, S. Multi-class obstacle detection and classification using stereovision and improved active contour models. *IET Intelligent Transport Systems*, v. 10, n. 3, p. 197–205, 2016. ISSN 1751-956X.

JÉGOU, S.; DROZDZAL, M.; VAZQUEZ, D.; ROMERO, A.; BENGIO, Y. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In: IEEE. *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*. [S.l.], 2017. p. 1175–1183.

JÉGOU, S.; DROZDZAL, M.; VAZQUEZ, D.; ROMERO, A.; BENGIO, Y. The one hundred layers tiramisu: Fully

convolutional densenets for semantic segmentation. In: IEEE. *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*. [S.l.], 2017. p. 1175–1183.

KARPATHY, A. *Stanford University CS231n: Convolutional Neural Networks for Visual Recognition*. 2016. <http://cs231n.github.io/>.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

KRÄHENBÜHL, P.; KOLTUN, V. Efficient inference in fully connected crfs with gaussian edge potentials. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2011. p. 109–117.

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

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.

MCCULLOCH, W.; PITTS, W. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, v. 5, p. 127–147, 1943.

NIELSEN, M. A. *Neural Networks and Deep Learning*. Determination Press, 2015. <http://neuralnetworksanddeeplearning.com/>.

OLAH, C. Understanding convolutions. 2014. <http://colah.github.io/>.

REN, S.; HE, K.; GIRSHICK, R.; SUN, J. Faster r-cnn: Towards real-time object detection with region proposal networks. jun. 2015.

RONNEBERGER, O.; FISCHER, P.; BROX, T. U-net: Convolutional networks for biomedical image segmentation. 2015.

RUSSEL, S.; NOVIG, P. *Artificial Inteligence: A Modern Approach. 2a. edição*. [S.l.]: Prentice Hall, 2003.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. set. 2014.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. Going deeper with convolutions. In: *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2015. p. 1–9. ISSN 1063-6919.

WU, B.; IANDOLA, F.; JIN, P. H.; KEUTZER, K. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. 2016.

XU, L.; REN, J. S.; LIU, C.; JIA, J. Deep convolutional neural network for image deconvolution. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 2014. p. 1790–1798.

YU, F.; KOLTUN, V. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

YU, J.; JIANG, Y.; WANG, Z.; CAO, Z.; HUANG, T. Unitbox: An advanced object detection network. In: ACM. *Proceedings of the 2016 ACM on Multimedia Conference*. [S.l.], 2016. p. 516–520.

ZEILER, M. D. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.