Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Estimação do ângulo de direção por Vídeo para Veículos Autônomos utilizando Redes Neurais Convolucionais Multicanais

Arthur Emídio Teixeira Ferreira

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Flávio de Barros Vidal

Brasília
2017

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Estimação do ângulo de direção por Vídeo para Veículos Autônomos utilizando Redes Neurais Convolucionais Multicanais

Arthur Emídio Teixeira Ferreira

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Flávio de Barros Vidal (Orientador)
CIC/UnB

Prof.ª Dr.ª Carla Cavalcante Koike
CIC/UnB

Prof. Dr. Teófilo Emidio de Campos
CIC/UnB

Prof. Dr. Rodrigo Bonifácio de Almeida
Coordenador do Bacharelado em Ciência da Computação

Brasília, 12 de julho de 2017

Dedicatória

À minha família e ao leitor.

Agradecimentos

À minha mãe, **Lucia Helena Cavalcanti Teixeira**, que me ensinou muito sobre a vida, que me encoraja e faz inúmeros sacrifícios para que eu realize todos os meus objetivos. Muito obrigado por ser uma ótima mãe e amiga. Todo o amor que recebi nesses 23 anos foram essenciais para eu ter chegado até aqui.

Ao meu pai, **Paulo Emídio Ferreira**, pelos conselhos, apoio, ensinamentos e pela educação que você e minha mãe me proporcionaram. Muito obrigado pela motivação para eu sempre dar o melhor de mim. Você é um grande pai!

À minha irmã, **Fernanda Helena Teixeira Ferreira**, por ter sido a melhor irmã que eu poderia ter pedido há 18 anos atrás. A sua companhia, amizade e bom humor foram muito importantes para toda essa caminhada.

À minha avó, **Helena Cavalcanti Teixeira**, por ser uma pessoa que sempre me apoiou muito em todas os períodos da minha vida. Muito obrigado pelo delicioso *mousse* de maracujá, por acreditar em mim e por todo o amor que recebo quando nos vemos.

À minha melhor amiga e namorada, **Mariana de Abreu Cavalcante dos Santos**, por todo o amor, pelas conversas, e pelos planos para o futuro. Muito obrigado por me ouvir e por sempre me dar esperanças e força para atingir os meus objetivos.

À toda minha família e amigos, por terem me proporcionado uma grande base, pelo companheirismo, e pela inspiração.

À CAPES e à *Haverford College*, por terem me proporcionado uma graduação sanduíche de excelente qualidade. *A special thanks to professor* ***Sorelle Friedler****, for being the person responsible for introducing me to computer vision.*

Ao meu orientador e amigo **Flávio de Barros Vidal**, pelas disciplinas em que fui seu aluno, pelos PIBICs, por todas as ideias trocadas em reuniões e *e-mails*, e por me dar um enorme apoio acadêmico e profissional, me abrindo diversas oportunidades pelas quais eu sou muito grato.

À todos professores e funcionários do Departamento de Ciência da Computação e do Departamento de Matemática da Universidade de Brasília por terem me proporcionado uma graduação pela qual tenho muito orgulho.

Resumo

A tecnologia de navegação em veículos autônomos consiste em uma aplicação de inteligência artificial ainda em aberto, e tem sido significantemente explorada pela indústria automobilística e tecnológica nos últimos anos devido ao potencial impacto que tal inovação trará ao cotidiano humano. A partir dessa motivação, este trabalho propõe uma metodologia baseada em redes neurais convolucionais (CNN) multicanais capazes de prever o ângulo de direção de um veículo, tendo como única entrada imagens capturadas por uma câmera acoplada na região frontal do veículo. São propostas cinco arquiteturas de redes convolucionais: de 1 canal, 2 canais e 3 canais na etapa de convolução. A partir dos testes realizados utilizando uma base de imagens públicas são apresentadas comparações quantitativas entre os modelos propostos.

Palavras-chave: redes neurais convolucionais, multicanal, visão computacional, aprendizado de máquina, veículos autônomos

Abstract

The navigation technology in autonomous vehicles is an artificial intelligence application which remains unsolved and has been significantly explored by the automotive and technological industries in the last years due to the potential impact that such innovation will bring in the near future. Based on this motivation, this work proposes a methodology grounded on multichannel convolutional neural networks (CNN) capable of predicting the steering angle of an autonomous vehicle, having as only input images captured by a camera attached to the vehicle's frontal area. It is proposed five convolutional network architectures: 1-channel, 2-channel and 3-channel in the convolution step. Based on the performed tests using a public image dataset, it is presented a quantitative comparison between the proposed models.

Keywords: convolutional neural networks, multichannel, computer vision, machine learning, autonomous vehicles

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

α	Taxa de aprendizado de um modelo
λ	Hiperparâmetro de regularização
$\mathcal{H}$	Domínio da função de hipóteses de aprendizado
$\nabla_\theta f$	Gradiente da função f avaliado em θ
Ψ_i	Valor de saída do neurônio artificial i
σ	Função sigmóide
τ	Função de ativação de uma rede neural artificial
θ	Parâmetros de um modelo
ε	Função de ativação ELU
a_i	Valor de saída do neurônio artificial i
F	Altura e largura de um *kernel* de convolução
H	Comprimento de uma imagem
$h_\theta(x)$	Hipótese dada uma entrada x, parametrizado por θ
I	Imagem de entrada
in_i	Combinação linear das entradas de um neurônio artificial i
K	*kernel* de convolução
L	Função de custo de um modelo genérico de aprendizado de máquina
M	Mapa de ativação de uma camada de convolução
P	Tamanho de *padding* de um *kernel* de convolução

R Função de penalização de uma função de custo

S Tamanho de passo (*stride*) de um *kernel* de convolução

W Largura de uma imagem

x Vetor de entrada

$X^{(test)}$ Conjunto de dados de teste

$X^{(train)}$ Conjunto de dados de treinamento

$y^{(train)}$ Vetor de valores "alvo" de um conjunto de dados de treinamento

J Função de custo de um modelo de regressão linear

m Número de exemplos em um conjunto de dados

n Número de características de um modelo

Lista de Abreviaturas e Siglas

ALVINN *Autonomous Land Vehicle In a Neural Network.*

CNN *Convolutional Neural Network.*

CONV Camada de Convolução.

CPU *Central Processing Unit.*

ELU *Exponential Linear Unit.*

FC Camada Totalmente Conectada.

GAN *Generative Adversarial Networks.*

GPS *Global Positioning System.*

GPU *Graphics Processing Unit.*

HDF5 *Hierarchical Data Format version 5.*

IMU *Inertial Measurement Unit.*

NRMSE *Normalized Root Mean Squared Error.*

OMS Organização Mundial da Saúde.

POOL Camada de *Pooling*.

ReLU Rectified Linear Unit.

RGB *Red Green Blue.*

RMSE *Root Mean Squared Error.*

RNA Rede Neural Artificial.

RNN *Recurrent Neural Network*.

STIP *Space-Time Interest Points*.

TPU *Tensor Processing Unit*.

Capítulo 1

Introdução

1.1 Contextualização

Em 1885, o inventor Karl Benz construiu o primeiro veículo equipado com um motor de combustão interna a entrar em produção, sendo um marco significante para o início da indústria automobilística. Desde então, os carros se tornaram elementos cada vez mais importantes para a era contemporânea, tendo causado um impacto relevante no cotidiano humano e na economia mundial. Por exemplo, é reportado que um total de 94.976.569 veículos foram produzidos em 2016, sendo o Brasil responsável pela produção de 2.156.356 [3].

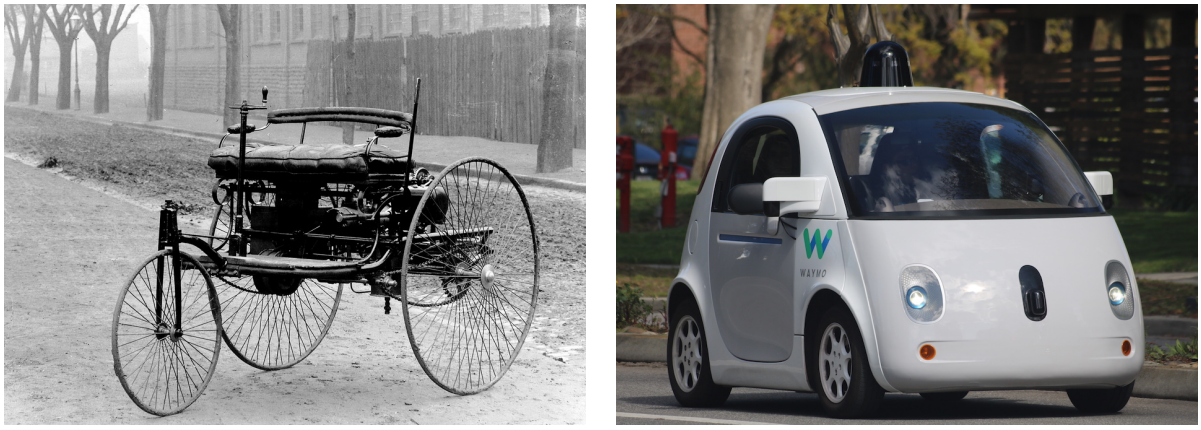
Ao longo de mais de um século de indústria automobilística, os automóveis têm evoluído tecnologicamente de maneira expressiva. No entanto, uma característica tem se mantido a mesma: o controle do veículo é responsabilidade de um agente humano.

Um automóvel autônomo é definido como um veículo capaz de compreender o ambiente ao seu redor e de se auto-navegar [4]. A ideia de navegação autônoma não é nova, dado que sistemas de piloto automático têm sido implementados em aviões e navios desde a primeira metade do século XX. No entanto, até a escrita deste manuscrito, carros completamente autônomos ainda não são uma realidade para a população em geral. Isso se deve ao fato de que a navegação em regiões urbanas apresenta uma série de desafios ainda em aberto na área de inteligência artificial e visão de máquina, como: detecção de objetos (e.g. demais veículos, pedestres, ciclistas, animais), estimação de ações, e decisão em situações críticas.

Os primeiros avanços significativos no desenvolvimento de um veículo autônomo ocorreram aproximadamente durante a década de 1980, quando pesquisadores da Universidade de Carnegie Mellon (Pensilvânia, EUA) criaram uma série de veículos autônomos chamados *Navlab* [5]. Em 1995, um desses carros realizou uma viagem de 5.000 quilômetros em estradas dos Estados Unidos, navegando de forma autônoma por 98,2% do tempo.

Recentemente, o desenvolvimento de veículos autônomos também tem ganho mais atenção da indústria. Por exemplo, o veículo vencedor na competição de corrida de carros autônomos titulada "*DARPA Grand Challenge*" em 2005 foi construído por uma equipe da Universidade de Stanford (Califórnia, EUA) em cooperação com um laboratório de pesquisa da empresa automobilística Volkswagen [6].

Na década de 2010, a indústria tem focado no desenvolvimento de carros autônomos para o consumidor final. Isso se deve principalmente aos avanços na tecnologia de sensores (e.g. LIDAR) [7], de processamento (e.g. GPU) [8], e também à pesquisa ativa na área de aprendizado de máquina (e.g. redes neurais artificiais profundas) [9].



(a) *Benz Patent-Motorwagen*: O veículo criado por Karl Benz em 1885.

(b) Um carro autônomo de testes produzido pela Waymo[1] em 2017. [2]

Figura 1.1: Um século de inovações na indústria automobilística.

1.2 Definição do problema

De acordo com a Organização Mundial da Saúde (OMS) [10], é estimado que 1,25 milhões de pessoas morreram em 2013 em decorrência de acidentes automobilísticos e estima-se que 20 a 50 milhões de pessoas sofreram danos físicos não fatais. Além disso, de acordo com um estudo feito pelo governo estadounidense [11], 94% dos acidentes de trânsito fatais nos Estados Unidos envolvem o erro humano. Com base nisso, é possível que o uso de veículos autônomos desenvolvidos sobre um rigoroso protocolo de controle de qualidade tenham o potencial de reduzir o número de acidentes de forma significante. Por exemplo, o tempo de reação e decisão de um sistema autônomo em situações críticas pode ser consideravelmente menor do que o de um ser humano. Além disso, automóveis podem se comunicar entre si de forma a evitar conflitos em ações tomadas.

[1] http://waymo.com

[2] Adaptado de commoms.wikimedia.org sob licença CC BY-SA 4.0.

De acordo com um estudo conduzido pelo Laboratório Nacional de Energia Renovável dos Estados Unidos [12], é possível que o advento de veículos autônomos reduza o consumo de combustíveis em até 90%. Iniciativas de compartilhamento de carros (*ridesharing*) poderiam reduzir o congestionamento nas ruas, potencialmente decaindo a emissão de gases poluentes na atmosfera. Adicionalmente, automóveis autônomos poderiam realizar rotas mais eficientes sob o ponto de vista energético.

Portanto, é justificável o argumento de que veículos autônomos podem trazer benefícios à sociedade. Por este motivo, diversas empresas automobilísticas e de tecnologia têm explorado este desafio nos últimos anos. Por exemplo, a companhia Tesla[3] possui um sistema embarcado em seus carros capaz de controlar velocidade, freio, aceleração e direção do veículo, sob a atenção completa do motorista.

Com base nisso, o problema abordado neste trabalho consiste no desenvolvimento de um sistema capaz de controlar a direção de um automóvel terrestre de maneira precisa.

1.3 Objetivos do projeto

A visão é um sentido essencial para motoristas tomarem decisões de direção de um automóvel. De forma a suprir essa necessidade, veículos autônomos são equipados com um conjunto de sensores que permitem a extração de informações sobre os objetos ao seu redor, como é o caso de câmeras que ficam instaladas em diferentes partes do carro, permitindo que o sistema de navegação interprete os *pixels* de entrada e produza os comandos de controle apropriados.

Nos últimos anos, com o advento do *aprendizado profundo* na área de inteligência artificial, as redes neurais convolucionais (CNN) têm se apresentado como uma interessante abordagem capaz de extrair informações de imagens digitais. Além disso, com os recentes avanços nas arquiteturas de unidade de processamento gráfico (GPU), o treinamento de redes neurais profundas se tornou factível.

Com isso, o objetivo geral deste trabalho consiste na **construção de modelos baseados em redes neurais convolucionais multicanais que sejam capazes de estimar o ângulo de direção de um veículo dado um conjunto de imagens da sua vista frontal**.

A partir do objetivo geral, são definidos os objetivos específicos do trabalho: *i)* apresentar uma arquitetura de rede neural convolucional capaz de estimar o ângulo de direção do veículo, *ii)* apresentar variações da arquitetura original com múltiplos canais de entrada, *iii)* avaliar a eficácia do método proposto, considerando as suas variações.

[3] http://tesla.com

1.4 Apresentação do manuscrito

No Capítulo 2 é apresentada uma revisão sobre os fundamentos de aprendizado de máquina, sendo introduzidos os diferentes tipos de aprendizado, algoritmos, e conceitos teóricos que são essenciais para a compreensão deste trabalho.

Já no Capítulo 3 é introduzido o conceito de redes neurais artificiais e como elas são capazes de "aprender" a partir de exemplos. No mesmo capítulo, é abordada a teoria acerca das redes neurais convolucionais (CNN) e a sua aplicabilidade no contexto de extração de características em imagens.

Para o Capítulo 4 é descrita a metodologia proposta para estimação do controle de direção de um veículo utilizando redes neurais convolucionais com múltiplos canais de entrada. No capítulo 5 são descritos os testes realizados e as respectivas análises qualitativas e quantitativas. Por fim, o capítulo 6 apresenta uma conclusão sobre a importância do trabalho e a discussão sobre possíveis trabalhos futuros.

Capítulo 2

Aprendizado de Máquina

Este capítulo tem como objetivo a apresentação de tópicos teóricos fundamentais da área de aprendizado de máquina para uma compreensão efetiva da metodologia e dos resultados apresentados neste trabalho.

2.1 Definição

O Aprendizado de Máquina, também referenciado como *Machine Learning*, é uma disciplina que estuda algoritmos capazes de aprender a partir de dados, sendo uma área multidisciplinar que se relaciona com inteligência artificial, probabilidade e estatística, teoria da complexidade computacional, teoria de controle, teoria da informação, filosofia, psicologia, neurobiologia, e outras áreas [13].

De acordo com [13], considera-se que um programa de computador aprende a partir da experiência E com respeito a uma classe de tarefas T e uma medida de performance P, se a sua performance em tarefas de T, medido por P, aperfeiçoa com a experiência E.

De forma a deixar essa definição mais clara, observemos um exemplo de um programa que tem por objetivo estimar as condições meteorológicas de uma cidade. Uma possibilidade de atribuição para as variáveis E, P, T seria:

- Experiência (E): uma base de dados históricos meteorológicos da região, obtidos em diferentes épocas de cada ano.
- Performance (P): a proporção de exemplos estimados com sucesso (i.e. acurácia).
- Tarefa (T): estimar se a região estará ensolarada, chuvosa, ou nevosa.

2.1.1 Problemas para *Machine Learning*

Como visto no exemplo da seção anterior, o aprendizado de máquina pode ser utilizado para resolver diversos problemas práticos. De forma geral, tarefas que se enquadram nos tipos de problemas abaixo podem ser potencialmente exploradas através de *Machine Learning*, de acordo com apresentado em [14]:

1. Classificação: classificar a entrada dentre as k categorias existentes. Em tarefas de classificação, um algoritmo de aprendizado de máquina pode exercer o papel da função $f : \mathbb{R}^n \longrightarrow \{1, ..., k\}$. Neste caso, a saída da função é a categoria estimada do dado de entrada. Outra função que pode ser produzida por um algoritmo de classificação é uma de distribuição de probabilidade para as classes existentes. Um exemplo de problema de classificação é o de reconhecimento de objetos dada uma imagem de entrada, como é apresentado em [15], que faz uso de redes neurais convolucionais.

2. Regressão: é um tipo de problema no qual o algoritmo de aprendizado de máquina deve dar como saída um valor numérico, produzindo uma função $f : \mathbb{R}^n \longrightarrow \mathbb{R}$. Portanto, a saída de um problema de regressão é contínua, diferente dos problemas de classificação. Um exemplo de tarefa de regressão é a de estimar o valor de um imóvel dada a sua área e localização.

3. Transcrição: consiste na geração de uma descrição em formato textual sobre o dado de entrada. Um exemplo para esse problema é o de descrever o conteúdo de uma imagem com frases em inglês, como proposto em [16].

4. Tradução Automática: dada uma entrada estruturada em uma certa linguagem, um algoritmo de aprendizado de máquina que resolve o problema de tradução automática deve traduzir este dado para outra linguagem. Um exemplo comum é o de tradução textual de idiomas, como é o caso do sistema de tradução proposto em [17], que utiliza redes neurais recorrentes (*RNNs*).

5. Detecção de Anomalias: consiste em classificar em um conjunto de dados aqueles que são mais atípicos. O trabalho apresentado em [18] usa algoritmos de aprendizado de máquina para detectar anomalias mecânicas que são difíceis de serem estimadas, com base em dados extraídos por sensores.

6. Síntese de Dados: o objetivo é produzir novos exemplos similares aos dados de entrada. Isso pode ser útil, por exemplo, para a geração de novos dados de treinamento para demais algoritmos de aprendizado de máquina. O trabalho [19] usa *Generative Adversarial Networks* (*GANs*) para sintetizar imagens que se assemelham com os dados de treinamento.

Elencado os principais problemas a serem resolvidos a partir do uso de Aprendizado de Máquinas, nas Seções seguintes serão apresentados os principais problemas que foram empregados na metodologia proposta e/ou que fornecem subsídios teóricos sobre o método adotado.

2.2 Paradigmas de aprendizado

Os principais paradigmas de aprendizado, de acordo com [14], são:

2.2.1 Aprendizado supervisionado

No contexto de aprendizado supervisionado, o agente recebe como entrada uma base de dados tal que cada exemplo descrito por um vetor x possui um valor ou um rótulo associado y. A partir disso, um algoritmo de aprendizado supervisionado deve ser capaz de estimar y dado um x, estimando, por exemplo, a distribuição de probabilidade $p(y \mid x)$.

Problemas de regressão e classificação são geralmente explorados através do aprendizado supervisionado. Considere, por exemplo, o problema para classificar se o animal presente em uma imagem consiste em um cachorro ou um gato. Neste caso, o algoritmo de aprendizado supervisionado recebe um conjunto de imagens denominado por X e um conjunto de rótulos denominado y, tal que a i-ésima imagem $X^{(i)}$ possui um rótulo associado $y^{(i)} \in$ `{Cachorro, Gato}`.

Regressão Linear e Redes Neurais Artificiais são exemplos de técnicas que podem ser aplicadas no contexto de aprendizado supervisionado, e serão vistas com detalhes nas seções 2.3 e 3.1, respectivamente.

2.2.2 Aprendizado não-supervisionado

No aprendizado não-supervisionado, o agente recebe bases de dados sem rótulo ou valor associado. Com isso, um algoritmo de aprendizado não-supervisionado necessita encontrar alguma estrutura na base de dados de entrada, sem receber *feedback* durante o processo de aprendizado. Portanto, um agente observa cada exemplo x da base de dados e tenta aprender a distribuição de probabilidade $p(x)$.

Um exemplo de aprendizado não-supervisionado é o de clusterização de dados de acordo com um conjunto de características. Um exemplo de algoritmo utilizado para resolver tal problema é o *k-means* [20], que é capaz de dividir os exemplos de entrada em k conjuntos disjuntos.

2.2.3 Aprendizado semi-supervisionado

Existem problemas que podem ser resolvidos com uma união entre as ideias por trás do aprendizado supervisionado e não-supervisionado. Por exemplo, podem existir situações nas quais a base de dados contém rótulos em somente uma fração das entradas. Também é possível que os dados de entrada possuam ruídos que impossibilitem um aprendizado puramente supervisionado.

2.2.4 Aprendizado por reforço

O aprendizado por reforço se caracteriza pela interação do agente com o ambiente. Para cada decisão realizada, o ambiente responde com uma recompensa ou uma punição. A partir desse sistema de *feedback*, o agente deve convergir para uma configuração tal que a ocorrência de recompensas seja maximizada. Um exemplo de utilização dessa abordagem de aprendizado é o uso de redes neurais profundas para a criação de um agente capaz de aprender o complexo jogo de tabuleiro *Go* [21].

2.3 Regressão Linear

2.3.1 Definição

De acordo com [14], a Regressão linear é um método capaz de aprender uma função linear para estimar um escalar y dado um vetor $x \in \mathbb{R}^n$ de entrada. No ponto de vista de aprendizado de máquina, a regressão linear apresenta conceitos importantes para a compreensão de técnicas mais avançadas.

Um modelo de regressão linear requer um processo de treinamento prévio para que os parâmetros $\theta \in \mathbb{R}^n$ da função linear sejam ajustados a partir de um determinado conjunto de dados $X^{(train)} \in \mathbb{R}^{n \times m}$, que contém m exemplos de n dimensões cada. Além disso, a base de dados tem para todo exemplo i um valor "alvo" $y_i^{(train)} \in \mathbb{R}$.

Com os parâmetros θ aprendidos, é esperado que o modelo seja capaz de estimar novas entradas não presentes no conjunto de treinamento. A partir disso, a hipótese de um modelo de regressão linear $h_\theta(x)$ para uma entrada $x \in \mathbb{R}^n$ é descrita por:

$$h_\theta(x) = \theta^\top x \tag{2.1}$$

Como exemplo, considere uma base de dados univariável, tal que a única variável corresponde ao número de pessoas que habitam uma determinada cidade. Além disso, considere que também é dado para cada exemplo o lucro de uma específica rede de restaurantes na determinada cidade. A partir desse conjunto de dados, é possível que um

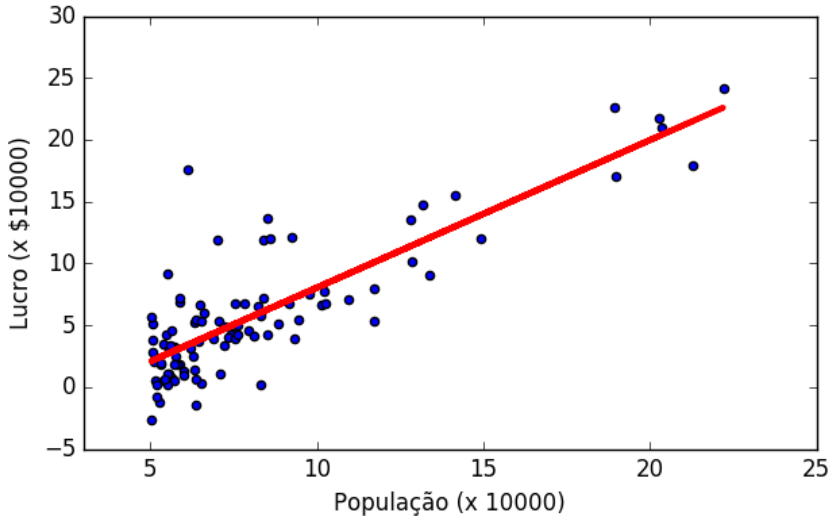


Figura 2.1: Gráfico de dispersão e a reta ajustada pelo algoritmo de regressão linear.

modelo de regressão linear seja capaz de estimar o lucro y que a rede de restaurantes teria se começasse a operar em uma cidade de x habitantes. A Figura 2.1 apresenta os resultados de um modelo treinado sobre uma base de dados fictícia, onde cada ponto azul corresponde a um exemplo de treinamento e a reta vermelha representa a função ajustada.

2.3.2 Função de Custo

Funções de custo são comumente utilizadas para medir a performance de um modelo de aprendizado de máquina sobre uma determinada base de dados. No contexto de regressão linear, uma função de custo $J : \mathbb{R}^n \to \mathbb{R}$ pode ser baseada no erro quadrático médio das predições geradas pelo modelo e o valor verdadeiro presente no conjunto de entrada:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(X^{(i)}) - y^{(i)})^2 \tag{2.2}$$

Tal que $X \in \mathbb{R}^{n \times m}$ corresponde a uma base de dados com m exemplos de n dimensões, $y \in \mathbb{R}^m$ ao vetor de valores "alvo" para cada exemplo do conjunto, e θ aos parâmetros aprendidos pelo modelo.

A partir disso, o objetivo de otimização do algoritmo de regressão linear é encontrar um conjunto de parâmetros ótimo θ^* que minimize a função de custo J:

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^n} J(\theta) \tag{2.3}$$

Um algoritmo iterativo conhecido como método da descida de gradiente é capaz de determinar um θ^* ótimo encontrando o mínimo local da função de custo J. No contexto de

regressão linear, a aplicação do método da descida de gradiente corresponde ao processo de treinamento.

2.4 Método da Descida de Gradiente

2.4.1 Definição

Seguindo o apresentado por [14], o método da descida de gradiente é um algoritmo de otimização utilizado para minimizar uma função diferenciável $f(x)$ baseado em seu gradiente $\nabla_x f$, que é definido como um vetor de derivadas parciais tal que cada elemento corresponde a uma dimensão do domínio da função. Por exemplo, suponha uma função de custo $J(\theta)$ tal que $\theta \in \mathbb{R}^n$, $\nabla_\theta J$ é definido por:

$$\nabla_\theta J = \left(\frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, ..., \frac{\partial J}{\partial \theta_n} \right) \tag{2.4}$$

O gradiente de uma função $J(\theta)$ corresponde à direção que os parâmetros devem mudar para $J(\theta)$ sofrer um pequeno acréscimo. Como o objetivo é de minimização, usamos a direção negativa dada por $\nabla_\theta J$. Portanto, atualizamos os parâmetros θ usando:

$$\theta = \theta - \alpha \nabla_\theta J(\theta) \tag{2.5}$$

Tal que α corresponde à taxa de aprendizado, sendo um escalar positivo que determina o tamanho de cada "passo" realizado pelo algoritmo. Quanto menor a taxa de aprendizado, mais iterações serão necessárias para atingir o mínimo local. No entanto, um α muito alto pode causar divergência. Portanto, um α pode ser escolhido usando busca binária e selecionando aquele valor que possui melhor performance em tempo de treinamento e que não tem problemas de convergência. Uma outra estratégia é o uso de um α dinâmico, utilizando "passos" mais altos nas primeiras iterações (i.e. mais longe do ponto ótimo), e o diminuindo com o passar do tempo.

As iterações do método da descida de gradiente podem ser visualizadas no exemplo fictício apresentado na Figura 2.2, que consiste em um gráfico de superfície de uma função $J : \mathbb{R}^2 :\rightarrow \mathbb{R}$. Cada seta representa a transição dos parâmetros θ após cada iteração do método da descida de gradiente. O ponto amarelo se refere ao custo com os pesos iniciais. A cada iteração do método, os parâmetros se ajustam de forma a diminuir o valor da função. O método é interrompido quando o mínimo local, representado pelo ponto verde, é atingido.

O método da descida de gradiente pode ser lido em pseudocódigo em Algoritmo 1.

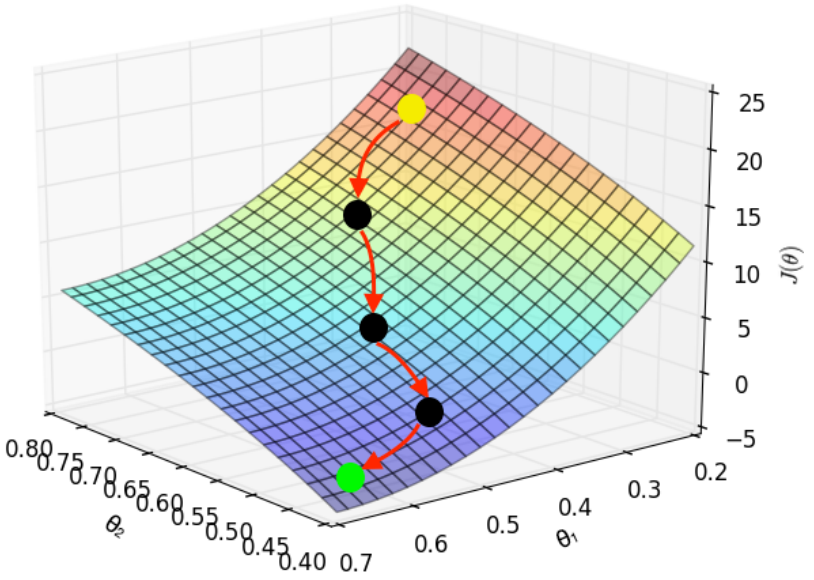


Figura 2.2: Visualização do método da descida de gradiente para encontrar o mínimo local de uma função de custo $J(\theta_1, \theta_2)$.

Algoritmo 1 Método da Descida de Gradiente

procedimento GRADIENT_DESCENT
    Entrada: um conjuto de parâmetros iniciais θ e uma taxa de aprendizado α.
    repita
        $\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$ (atualização simultânea para $j = 1, 2, ..., n$).
    até convergir à um mínimo local.

2.4.2 Variações do Método da Descida de Gradiente

Considere a função de otimização $J(\theta)$ apresentada para o algoritmo de regressão linear na Equação 2.2. A derivada parcial de $J(\theta)$ em respeito a $\theta_j \in \theta$ é dada pela Equação 2.6:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) \cdot x_j^{(i)} \tag{2.6}$$

Como é visto em Algoritmo 1, $\frac{\partial}{\partial \theta_j} J(\theta)$ precisa ser computado em cada iteração do método da descida de gradiente clássico. Com base nisso, observa-se em 2.6 a presença de um somatório que passa por todos os m exemplos do conjunto de dados de entrada. Em outras palavras, o método da descida de gradiente clássico requer o processamento de toda a base de dados de treinamento para a realização de um único ajuste aos parâmetros θ. Portanto, essa metodologia pode não ser praticável em extensos conjuntos de dados, pois é possível que o método demore muito tempo para encontrar o mínimo local da função a ser otimizada.

A partir dessa ideia, introduzimos duas variações do método da descida de gradiente que possibilitam o treinamento usando grandes conjunto de dados.

Método da Descida de Gradiente Estocástico

O método da descida de gradiente estocástico utiliza somente 1 exemplo em cada iteração do algoritmo, fazendo com que os parâmetros sejam ajustados a cada novo exemplo. Isso traz benefícios em relação ao método da descida de gradiente clássico pois a convergência ao mínimo local será potencialmente mais rápida.

No entanto, como os parâmetros são atualizados a cada novo exemplo, podem ocorrer atualizações de parâmetros que não contribuem para a convergência ao mínimo local. Isso se deve ao fato de que podem existir exemplos (e.g. *outliers*) que introduzem ruído ao sistema.

O Algoritmo 2 apresenta o método da descida de gradiente estocástico em formato de pseudocódigo usando a função de custo descrita pela Equação 2.2 e seu gradiente presente na Equação 2.6.

Método da Descida de Gradiente em Mini Lotes

A ideia do método da descida de gradiente em mini lotes é utilizar b exemplos em cada iteração do algoritmo, tal que $1 < b < m$. Nessa variação os parâmetros são atualizados com mais frequência do que no método da descida de gradiente clássico, potencialmente reduzindo o tempo para convergência ao mínimo local. Além disso, o problema que ocorre

Algoritmo 2 Método da Descida de Gradiente Estocástico

1: **procedimento** STOCHASTIC_GRADIENT_DESCENT
2: **Entrada:** um conjunto de parâmetros iniciais θ e uma taxa de aprendizado α.
3: Reordene aleatoriamente os exemplos de treinamento.
4: **repita**
5: **para** i $\leftarrow$ 1 até m **faça**
6: $\theta_j \leftarrow \theta_j - \alpha(h_\theta(x_i) - y_i) \cdot x_j^{(i)}$ (atualização simultânea para $j = 1, 2, ..., n$).
7: **fim para**
8: **até** convergir a um mínimo local.

no método da descida de gradiente estocástico de um exemplo contribuir com ruído na atualização dos parâmetros tende a diminuir, já que não é considerado somente 1 exemplo em cada iteração do algoritmo.

Adicionalmente, é possível que a computação de cada iteração seja paralelizada entre até b núcleos de processamento, já que tais cálculos são independentes. Um ponto negativo é que b se torna mais um hiperparâmetro a ser considerado, e que pode precisar ser ajustado para diferentes tipos de treinamento.

2.5 Sobre-ajuste e sub-ajuste

De acordo com [14], o que separa *machine learning* de um problema de otimização é a necessidade de que técnicas de aprendizado de máquina sejam capazes de treinar modelos que generalizam para exemplos que nunca foram processados anteriormente. Ou seja, não é suficiente que um modelo apresente boa performance na base de dados utilizada na etapa de treinamento. Sejam $X^{(train)}$ e $X^{(test)}$ os conjuntos de dados de treinamento e de teste, respectivamente, tal que:

$$X^{(train)} \cap X^{(test)} = \emptyset \tag{2.7}$$

Além disso, é considerado em [14] como *erro de treinamento* e *erro de teste* o valor dado por uma função de custo $J(\theta)$ sobre $X^{(train)}$ e $X^{(test)}$, respectivamente. Portanto, para efeitos de generalização de um modelo de aprendizado, é importante que ambos os erros de treinamento e de teste sejam decaídos e estejam próximos entre si.

Caso o erro de treinamento sofra decréscimo e a sua diferença com o erro de teste continue alta, então identificamos um problema chamado sobre-ajuste. Em outras palavras, sobre-ajuste é causado por uma hipótese $h_\theta(x)$ que possui bons resultados para o conjunto de dados de treinamento mas não é capaz de generalizar para exemplos não vistos anteriormente. Isso pode ser consequência da utilização de uma pequena base de dados de treinamento ou pelo uso excessivo de características.

Caso um modelo de aprendizado não seja capaz de tornar o erro de treinamento baixo o suficiente, observamos a ocorrência do problema denominado sub-ajuste. Ou seja, o modelo não foi capaz de capturar a relação entre os dados de treinamento $X^{(train)}$ com os seus respectivos valores "alvo" $y^{(train)}$. Isso é geralmente causado por um modelo e/ou base de dados de treinamento muito simples (e.g. uso de poucas características). Uma opção para minimizar a ocorrência de sub-ajuste consiste na utilização de características polinomiais para que o modelo seja capaz de aprender uma hipótese mais complexa. Por exemplo, na regressão linear podemos tentar realizar o ajuste de pontos usando um polinômio de maior ordem.

É possível visualizar nas Figuras 2.3 e 2.5 a ocorrência de sub-ajuste e sobre-ajuste em um problema de regressão linear de uma única característica x.

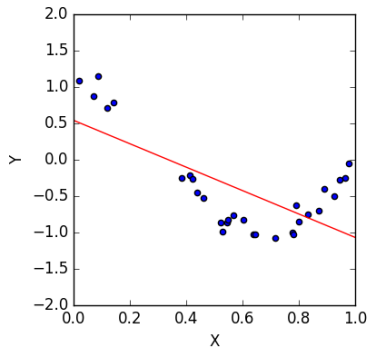


Figura 2.3: Sub-ajuste (função de 1º grau)

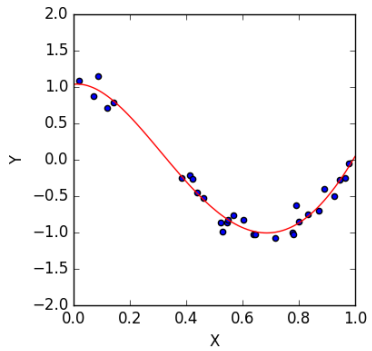


Figura 2.4: Ajuste aceitável (função de 4º grau)

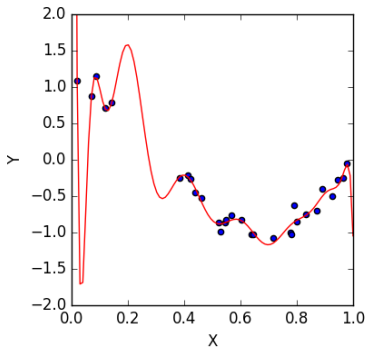


Figura 2.5: Sobre-ajuste (função de 15º grau)

No 1º exemplo (Figura 2.3), é observado que uma função de 1º grau não é capaz de ajustar os dados de treinamento de maneira aceitável, pois os pontos não estão distribuídos de forma linear. Portanto, o erro de treinamento será alto, indicando a ocorrência de sub-ajuste.

No 2º exemplo (Figura 2.5), foi utilizado um polinômio de 15º grau, sendo uma função exageradamente complexa para ajustar os dados de treinamento, o que pode levar a um alto erro de teste. Por exemplo, é provável que o decréscimo abrupto em y observado no começo do gráfico não contribua para exemplos de teste tal que $x \approx 0,05$.

Enquanto isso, a Figura 2.4 mostra a curva de ajuste usando um polinômio de 4º grau que apresentou um bom ajuste para os dados de treinamento e que pode ser capaz de generalizar para novos exemplos, potencialmente tendo um erro de teste inferior comparado aos outros dois modelos.

Regularização

Seja h uma função de hipótese que mapeia uma entrada x para um valor alvo y. De acordo com [22], o objetivo de otimização de um algoritmo de aprendizado de máquina é encontrar uma função h^* tal que:

$$h^* = \arg\min_{h \in \mathcal{H}} L(h) \tag{2.8}$$

Tal que $L(h)$ corresponde a uma função de custo, como é o caso da função de erro quadrático médio apresentada na Equação 2.2.

Regularização corresponde ao uso de uma função $R : \mathcal{H} \to \mathbb{R}$ para penalização da função de custo. Com isso, o novo objetivo de otimização é dado por:

$$h^* = \arg\min_{h \in \mathcal{H}} L(h) + \lambda R(h) \tag{2.9}$$

Onde λ corresponde a um escalar positivo denotado por *hiperparâmetro de regularização*. A regularização é considerada como uma medida de complexidade da função de otimização. Portanto, soluções complexas tendem a ser penalizadas, minimizando a possível ocorrência de sobre-ajuste. Vale ressaltar que a escolha do λ é de grande importância, pois um valor muito alto pode causar sub-ajuste, já que a função de custo L irá perder relevância para o objetivo de otimização.

Na regressão linear, uma função de regularização comumente utilizada é a soma dos quadrados dos pesos. Isso evita que certos pesos tenham uma influência desproporcional no cálculo da hipótese. Com isso, o objetivo de otimização de regressão linear pode ser reescrito como:

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^n} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 + \sum_{j=1}^{n} \theta_j^2 \tag{2.10}$$

Capítulo 3

Redes Neurais Artificiais

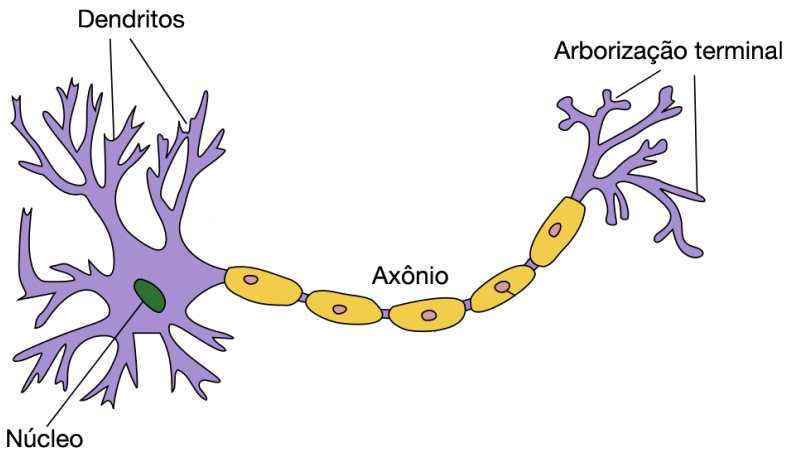
Modelos lineares de aprendizado de máquina geralmente não funcionam bem com hipóteses não-lineares, que são cruciais para a solução de problemas do mundo real. Por exemplo, suponha que queremos utilizar o algoritmo de regressão linear para aprender uma hipótese $h_\theta(x)$ não-linear. Para obter resultados satisfatórios, teríamos que adicionar características não-lineares, como $x_1x_2, x_1x_3, ...x_1x_n$. Isso implica em um aumento considerável no número de características, tornando impraticável aplicar modelos lineares para a solução de problemas que envolvam muitas dimensões (e.g. classificação de imagens e vídeos). Para estas e outras situações que a não-linearidade do problema a se classificar seja uma questão preponderante, faz-se o uso de Redes Neurais Artificiais (RNAs). A seguir, maiores detalhes sobre estas redes são apresentados.

3.1 Redes Neurais Artificiais Clássicas

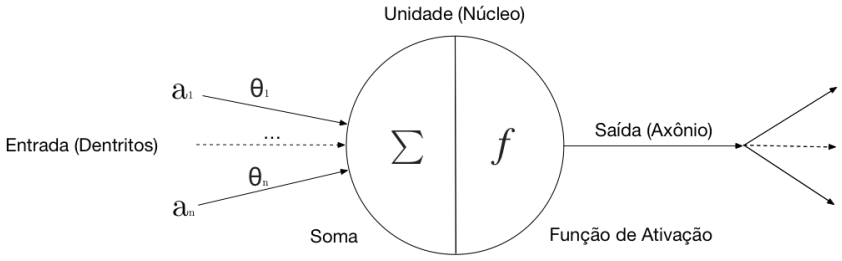
Uma rede neural artificial (RNA), de acordo com [22] é um modelo de aprendizado capaz de representar funções não-lineares. Essa seção tem como objetivo explicar redes neurais artificiais alimentadas para frente, que são representadas por um grafo acíclico dirigido (DAG). Esse tipo de RNA possui uma grande relevância na área de aprendizado profundo, como é o caso de redes neurais convolucionais, que serão discutidas na seção 3.2.1.

De acordo com a neurociência [23], neurônios são células do sistema nervoso essenciais para o processo de aprendizado em animais. Neurônios recebem impulsos elétricos provenientes de outros neurônios através de estruturas conhecidas como dendritos. Cada neurônio possui um membro chamado axônio, que é responsável por emitir sinais para demais neurônios. Um modelo simplificado de um neurônio pode ser visto na Figura 3.1a.

3.1.1 Neurônio



(a) Representação gráfica de um neurônio biológico. [1]



(b) Representação de um neurônio de uma RNA.

Figura 3.1: Comparação entre um neurônio biológico e um neurônio artificial.

Com base nessa estrutura biológica, um modelo inicial de neurônio artificial foi proposto em 1943 por McCulloch e Pitts [24]. Um neurônio artificial (também conhecido como unidade) pode ser representado por um vértice que contém um conjunto de arestas de entrada. Cada aresta j de um vértice i possui um peso denotado por $\theta_j^{(i)}$, que é multiplicado pelo valor de saída a_j de um outro neurônio. Com isso, a unidade i computa a combinação linear de suas n entradas:

$$in_i = \sum_{j=0}^{n} \theta_j^{(i)} a_j \tag{3.1}$$

Em seguida, a unidade precisa decidir o seu nível de ativação a partir da combinação linear in_i. Em outras palavras, determinar o grau de "excitação" do neurônio. Para isso, é computado $f(in_i)$, tal que f é a função de ativação da unidade, que será discutida em mais detalhes na seção 3.1.2. Finalmente, o valor de saída da unidade i é dado por:

[1] Adaptado de commons.wikimedia.org sob licença CC BY-SA 3.0.

$$a_i = f\left(\sum_{j=0}^{n} \theta_j^{(i)} a_j\right) \tag{3.2}$$

A saída da unidade é então propagada como entrada em demais neurônios. A Figura 3.1b apresenta uma representação gráfica de um neurônio artificial.

3.1.2 Função de Ativação

Uma função de ativação é definida por $\tau : \mathbb{R} \to \mathbb{R}$. De maneira geral, funções não-lineares são utilizadas de forma com que uma rede neural artificial seja capaz de representar modelos não-lineares. As funções comumente utilizadas na literatura são:

Sigmóide

A função sigmóide é dada por:

$$\sigma(x) = \frac{1}{1 + \mathrm{e}^{-x}} \tag{3.3}$$

Como pode ser visto no gráfico da Figura 3.2a, a função tem como saída um número real dado pelo intervalo $[0, 1]$. A sigmóide é geralmente utilizada para regressão logística, como é o caso de problemas de classificação. Por exemplo, considere uma RNA capaz de classificar se uma dada imagem contém uma face humana, e que todos os neurônios da rede usem a função sigmóide como função de ativação. Com isso, o resultado da unidade de saída é a probabilidade de uma face estar contida na imagem.

Tangente Hiperbólica

A função de tangente hiperbólica, visualizada pelo gráfico da Figura 3.2b, tem como saída um número real que pertence ao intervalo $[-1, 1]$. É uma função que se relaciona com a sigmóide pela seguinte igualdade:

$$\tanh(x) = 2\sigma(2x) - 1 \tag{3.4}$$

Função Linear Retificada (*ReLU*)

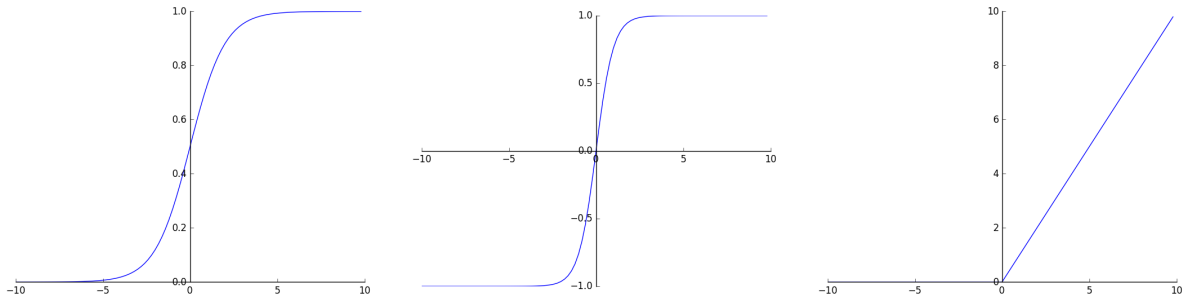
A função de ativação linear retificada aplica um limiar sobre a sua entrada, sendo definida por:

$$\Psi(x) = \max(0, x) \tag{3.5}$$

Apesar de Ψ ser uma função não-linear, sua computação é bastante eficiente pois não faz uso de expoentes, como é o caso de σ e tanh. Além disso, seu gradiente é simplesmente definido por:

$$\nabla\Psi(x) = \begin{cases} 0 & x < 0 \\ 1 & x > 0 \end{cases} \tag{3.6}$$

Com isso, métodos de otimização baseados no gradiente podem ser otimizados se fizerem o uso da função linear retificada. Por esta razão, é uma função que tem sido frequentemente utilizada em redes neurais profundas.



(a) Sigmóide (b) Tangente Hiperbólica (c) Função Linear Retificada

Figura 3.2: Exemplos de função de ativação.

3.1.3 Redes neurais artificiais de única camada

Uma rede neural artificial de única camada (também chamada de rede *perceptron*) consiste em uma RNA na qual a entrada está conectada diretamente com neurônios de saída. Uma rede *perceptron* pode ser utilizada para aprender funções lineares. Considere o exemplo apresentado em [25] baseado na função lógica AND, que opera sobre 2 operandos x_1 e x_2 e é definida pela Tabela 3.1.

x_1	x_2	x_1 AND x_2
0	0	0
0	1	0
1	0	0
1	1	1

Tabela 3.1: Tabela verdade da função lógica AND.

Com isso, podemos modelar uma RNA capaz de aprender a função AND através da seguinte hipótese:

$$h_\theta(x_1, x_2) = \sigma(\theta_1 x_1 + \theta_2 x_2 + b) \tag{3.7}$$

Tal que θ_1, θ_2 se referem aos parâmetros da rede que serão aprendidos, e σ é a função de ativação sigmóide. Além disso, b é um escalar referido como *bias*, e que também é aprendido na etapa de treinamento. O *bias* fornece maior flexibilidade ao modelo de aprendizado por ser um termo adicional que não depende de outro neurônio ou da entrada.

Modelando uma rede na qual o neurônio de saída produza $h_\theta(x_1, x_2)$, inicializando com valores aleatórios, e treinando a RNA usando técnicas que serão descritas na seção 3.1.5, podemos chegar em um resultado tal que $\theta_1 = \theta_2 = 20, b = -30$, conforme mostrado na Figura 3.3. Testando a hipótese $h_\theta(x_1, x_2)$ sobre os parâmetros obtidos, observamos que uma rede *perceptron* é capaz de aprender uma função linear:

$$\begin{aligned}
h_\theta(0,0) &= \sigma(20 \cdot 0 + 20 \cdot 0 - 30) = \sigma(0 + 0 - 30) = \sigma(-30) = 0 \\
h_\theta(0,1) &= \sigma(20 \cdot 0 + 20 \cdot 1 - 30) = \sigma(0 + 20 - 30) = \sigma(-10) = 0 \\
h_\theta(1,0) &= \sigma(20 \cdot 1 + 20 \cdot 0 - 30) = \sigma(20 + 0 - 30) = \sigma(-10) = 0 \\
h_\theta(1,1) &= \sigma(20 \cdot 1 + 20 \cdot 1 - 30) = \sigma(20 + 20 - 30) = \sigma(10) = 1
\end{aligned}$$

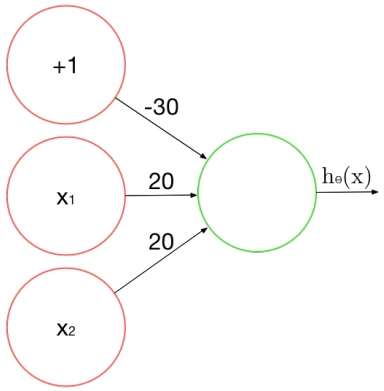


Figura 3.3: Rede neural artificial de única camada que produz a função lógica AND.

3.1.4 Redes neurais artificiais de múltiplas camadas

Uma limitação das redes neurais artificiais de única camada é a incapacidade de aprender funções não-lineares, como é o caso da função lógica XOR. No entanto, modelos não-lineares podem ser construídos usando redes neurais de múltiplas camadas.

Considere a rede neural apresentada na Figura 3.4, que possui 4 camadas. A 1ª camada contém as entradas da rede. A 2ª e a 3ª são denominadas camadas internas da rede. Por fim, a 4ª é a camada de saída da RNA, que produz a hipótese em termos dos parâmetros

atuais da rede e o dado de entrada. O número de camadas internas e a quantidade de unidades em cada camada são parâmetros arbitrários. Teoricamente, é esperado que um maior número de camadas e neurônios permitam que a rede neural artificial seja capaz de extrair mais informações dos dados de entrada, aprendendo funções mais complexas que se ajustam melhor aos pontos de treinamento, podendo inclusive chegar em situações de sobre-ajuste. No entanto, técnicas de regularização podem ser aplicadas para evitar a ocorrência de tal problema.

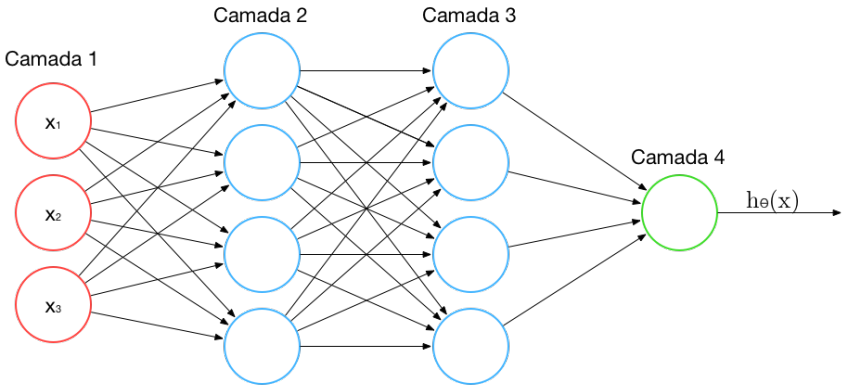


Figura 3.4: Rede neural artificial de mútiplas camadas completamente conectada.

O modelo gerado por uma rede neural artificial de múltiplas camadas pode ser interpretado como uma aplicação de funções compostas não-lineares, dado que cada neurônio produz um valor determinado por uma função de ativação não-linear. Com isso, uma RNA de múltiplas camadas é capaz de resolver problemas de regressão não-linear. Considere novamente a função lógica XOR, que pode ser escrita em termos de funções lineares AND, OR, NOT:

$$x_1 \oplus x_2 = (x_1 \vee x_2) \wedge \neg(x_1 \wedge x_2) \tag{3.8}$$

A partir disso, podemos modelar uma RNA de múltiplas camadas contendo redes *perceptrons* encadeadas que computam as funções lógicas AND, OR, e NOT. Logo obtemos uma rede neural artificial capaz de gerar uma hipótese que aproxima a função não-linear XOR.

3.1.5 Aprendizado em Redes Neurais Artificiais

O aprendizado em redes neurais artificiais é composto por duas etapas principais: a propagação e a retropropagação, de acordo com [22].

A propagação de uma RNA, como apresentado por [22], consiste na computação das funções de cada unidade seguindo a ordem topológica da rede, ou seja, começando a partir

da camada de entrada e finalizando na camada de saída. A última camada produz uma hipótese $h_\theta(x)$, baseada nos parâmetros atuais θ e no dado de treinamento x, que possui um valor alvo y. Com isso, é possível definir um erro de estimação $E(\theta)$:

$$E(\theta) = |y - h_\theta(x)|^2 \tag{3.9}$$

O objetivo da retropropagação é computar a contribuição de cada parâmetro da rede para o erro $E(\theta)$. Essa informação é obtida a partir do gradiente $\nabla_\theta E(\theta)$. Para calcular o gradiente da função, o algoritmo de retropropagação propaga o erro da unidade de saída para os neurônios internos usando o sentido inverso da ordem topológica da rede.

Com o gradiente $\nabla_\theta E(\theta)$ computado, cada parâmetro da RNA pode ser alterado usando o método do gradiente estocástico. A equação de atualização de um parâmetro θ_i da rede é dada por:

$$\theta_i = \theta_i - \alpha \frac{\partial E(\theta)}{\partial \theta_i} \tag{3.10}$$

O algoritmo de aprendizado em redes neurais artificiais baseado na retropropagação pode ser visto em formato de pseudocódigo em Algoritmo 3 [22].

Vale ressaltar que existem outros métodos de otimização estocástica que podem apresentar melhores resultados de convergência em redes neurais artificiais. O trabalho [26] apresenta uma série de comparações de performance entre diferentes tipos de algoritmos, como ADAGRAD, ADADELTA, RPROP, e RMSprop.

Algoritmo 3 Iteração de aprendizado em uma rede neural artificial.

procedimento LEARNING_ITERATION

Entrada: Um vetor x de entrada associado à um valor alvo de saída y.

Entrada: Uma rede neural artificial totalmente conectada de L camadas, tal que $\theta_{i,j}^{(l)}$ denota o valor do peso da conexão entre a unidade i da camada $l-1$ para a unidade j da camada l. Cada neurônio da RNA utiliza uma mesma função de ativação g, de derivada g'.

 # Propagação para frente:

para cada unidade i da 1ª camada **faça**

$a_i^{(1)} \leftarrow x_i$

fim para

para $l \leftarrow 2$ até L **faça**

para cada unidade j na camada l **faça**

$in_j^{(l)} \leftarrow \sum_i \theta_{i,j}^{(l)} a_i^{(l-1)}$

$a_j^{(l)} \leftarrow g(in_j^{(l)})$

fim para

fim para

 # Retropropagação:

para cada unidade j na última camada L **faça**

$\delta_j^{(L)} \leftarrow g'(in_j^{(L)}) \cdot (y_j - a_j^{(L)})$

fim para

para $l \leftarrow L - 1$ até 2 **faça**

para cada unidade j na camada l **faça**

$\delta_j^{(l)} \leftarrow g'(in_j^{(l)}) \sum_i \theta_{j,i}^{(l+1)} \cdot \delta_i^{(l+1)}$

fim para

fim para

 # Atualização dos pesos a partir do método do gradiente estocástico:

para todos pesos $\theta_{i,j}^{(l)}$ tal que $l = 2, 3, ..., L$ **faça**

$\theta_{i,j}^{(l)} \leftarrow \theta_{i,j}^{(l)} + \alpha \cdot a_i^{(l)} \cdot \delta_j^{(l)}$

fim para

3.2 Redes Neurais Profundas

Como vimos na seção anterior, redes neurais artificiais que possuem mais unidades e camadas são capazes de representar funções não-lineares mais complexas. No entanto, o aumento de camadas em uma RNA causa um crescimento considerável da carga de processamento para a realização do treinamento da rede, devido ao grande número de parâmetros que devem ser ajustados em cada iteração do algoritmo de aprendizado.

Uma forma de computar o algoritmo de aprendizado de RNAs (como o descrito em Algoritmo 3) é através de multiplicação de matrizes. Com base nisso, unidades de processamento gráfico (GPU) têm sido adotadas para treinar redes neurais artificiais devido à notável diferença de performance que GPUs possuem em relação à CPUs para a execução de operações matriciais. Além disso, trabalhos recentes propõem uma arquitetura de processamento computacional dedicada para a execução de algoritmos de aprendizado de máquina, referenciada como *Tensor Processing Unit* (TPU) [27].

Com tal recente avanço tecnológico, o treinamento de RNAs de múltiplas camadas se tornou praticável, surgindo o termo "rede neural profunda". Apresentamos na próxima seção um tipo de rede neural comumente associada com o conceito de redes neurais profundas - redes neurais convolucionais especificamente - seguindo as ideias apresentadas no trabalho de [14].

3.2.1 Redes Neurais Convolucionais

Redes neurais convolucionais (CNN), de acordo com [14], são redes neurais artificiais especializadas para processar dados de entrada que possuem alguma espécie de topologia espacial, como é o caso de imagens, vídeos, áudio, e texto.

Redes neurais convolucionais foram introduzidas em 1989 por LeCun et al. [28] através de um trabalho desenvolvido para reconhecer dígitos escritos à mão em uma imagem de entrada, sendo uma aplicação de bastante relevância para o serviço de correio norte-americano da época.

Após o trabalho inicial de Yann LeCun [28], redes neurais convolucionais ganharam mais atenção em 2012, com o desenvolvimento de uma CNN titulada *AlexNet* [15], capaz de atingir resultados melhores que o estado da arte em uma competição de classificação de imagens (*ImageNet Large Scale Visual Recognition Challenge*).

Redes neurais artificiais clássicas completamente conectadas são capazes de receber os *pixels* de uma imagem como entrada e aprender características relevantes sobre o seu conteúdo. No entanto, o custo computacional para fazer isso é muito elevado. Considere uma imagem de dimensões 300x300 com 3 canais. Um único neurônio da 2ª camada que esteja conectado com todas as unidades de entrada possui $300 \cdot 300 \cdot 3 = 270.000$

parâmetros a serem aprendidos através da retropropagação. Como uma única unidade e uma camada não são suficientes para aprender características sobre uma imagem, esse número aumenta consideravelmente, sendo impraticável treinar uma rede que receba uma entrada com alta dimensionalidade. Além disso, redes neurais artificiais clássicas não consideram a estrutura espacial da entrada (i.e. a RNA pode não considerar a proximidade entre *pixels*).

Uma rede neural é considerada convolucional quando a mesma possui pelo menos uma camada de convolução. Essa camada recebe uma entrada multidimensional (também chamada por tensor) e a aplica uma série de convoluções usando um conjunto de filtros. Além de camadas de convolução, CNNs também são geralmente compostas por outros tipos de camadas. Antes de definir cada uma, definimos o que é uma operação de convolução.

Operação de Convolução

Sejam as funções $p : \mathbb{R} \to \mathbb{R}$ e $q : \mathbb{R} \to \mathbb{R}$. A convolução é definida como uma operação linear $*$ que computa a superposição de duas funções em função de um deslocamento τ:

$$(p * q)(t) = \int_{-\infty}^{\infty} p(\tau)q(t - \tau)d\tau \tag{3.11}$$

A equação 3.11 descreve a definição de uma convolução contínua. Para o contexto de redes neurais artificiais, na qual a entrada é discreta (e.g. uma imagem digital), descrevemos a definição da convolução discreta:

$$(p * q)(t) = \sum_{\tau=-\infty}^{\infty} p(\tau)q(t - \tau) \tag{3.12}$$

Convolução em imagens

No contexto do domínio espacial, a filtragem de imagens pode ser realizada através de convoluções. Um filtro de imagem é um procedimento que recebe como entrada a imagem original I e uma matriz bidimensional K (também chamada de *kernel*) de dimensões $m \times n$, e produz uma imagem de saída I'. Usando a equação 3.12 como referência, a convolução pode ser descrita por:

$$I'(i, j) = (I * K)(i, j) = \sum_{m} \sum_{n} I(m, n)K(i - m, j - n) \tag{3.13}$$

Em outras palavras, a operação de convolução em imagens pode ser compreendida pela ação de movimentar o *kernel* sobre todas as posições da imagem, computando o produto escalar entre K e $I(i, ..., i + m, j, ...j + n)$ (i.e. a parte da imagem coberta pelo *kernel*).

Como exemplo, considere o filtro Laplaciano, que pode ser descrito pelo *kernel* $K_{\text{Laplacian}}$:

$$K_{\text{Laplacian}} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{3.14}$$

Um exemplo de aplicação do filtro Laplaciano sobre uma imagem pode ser visto na Figura 3.5. Como é possível notar, o filtro foi capaz de destacar as bordas presentes na imagem de entrada.

(a) Imagem original. (b) Imagem filtrada.

Figura 3.5: Aplicação do filtro Laplaciano sobre uma imagem.

Além do *kernel* utilizado, a saída do processo de convolução é determinada por duas variáveis: uso de *padding* e o tamanho do passo.

O *padding* é caracterizado pela inserção simétrica de novos *pixels* ao redor da imagem de entrada. No contexto de filtros, é comum a inserção de *pixels* de valor 0 (*0-padding*). Isso é realizado para que tenhamos um controle sobre o tamanho da saída e que as bordas da imagem original sejam corretamente consideradas no processo de convolução.

O tamanho do passo (também chamado de *stride*) consiste no número de *pixels* que o *kernel* irá se mover sobre a imagem de entrada para realizar cada produto escalar. Ou seja, se o *stride* utilizado for 1, o *kernel* não irá "saltar" *pixels* durante a sua passagem sobre a imagem, percorrendo-a *pixel* a *pixel*. Por outro lado, caso maiores tamanhos sejam utilizados, menos produtos escalares serão computados, e com isso, o tamanho da saída será menor.

Considere que o tamanho da imagem de entrada I seja $W \times H$, que o tamanho do *kernel* K seja $F \times F$, que o tamanho do *padding* seja P, e que o tamanho do passo seja P. A partir disso, a largura W' e a altura H' da imagem filtrada são dadas por:

$$W' = \frac{(W - F + 2P)}{S} + 1 \tag{3.15}$$

$$H' = \frac{(H - F + 2P)}{S} + 1 \tag{3.16}$$

Camadas de uma Rede Neural Convolucional

Descrevemos abaixo três tipos de camadas comumente encontradas em arquiteturas de redes neurais convolucionais.

Camada de Convolução (CONV)

A camada de convolução é o componente principal de uma CNN. O seu papel durante a etapa de propagação é aplicar a convolução de K filtros sobre a entrada, gerando K saídas chamadas *mapas de ativação*. Cada filtro é aplicado sobre a profundidade da entrada. Por exemplo, se a entrada consiste em uma imagem de 3 canais de cor (e.g. RGB), o filtro também precisa ter 3 de profundidade, de forma a gerar um único mapa de ativação bidimensional que considere todos os canais da entrada. Observe na Figura 3.6 os resultados produzidos por uma camada de convolução durante o estágio de propagação.

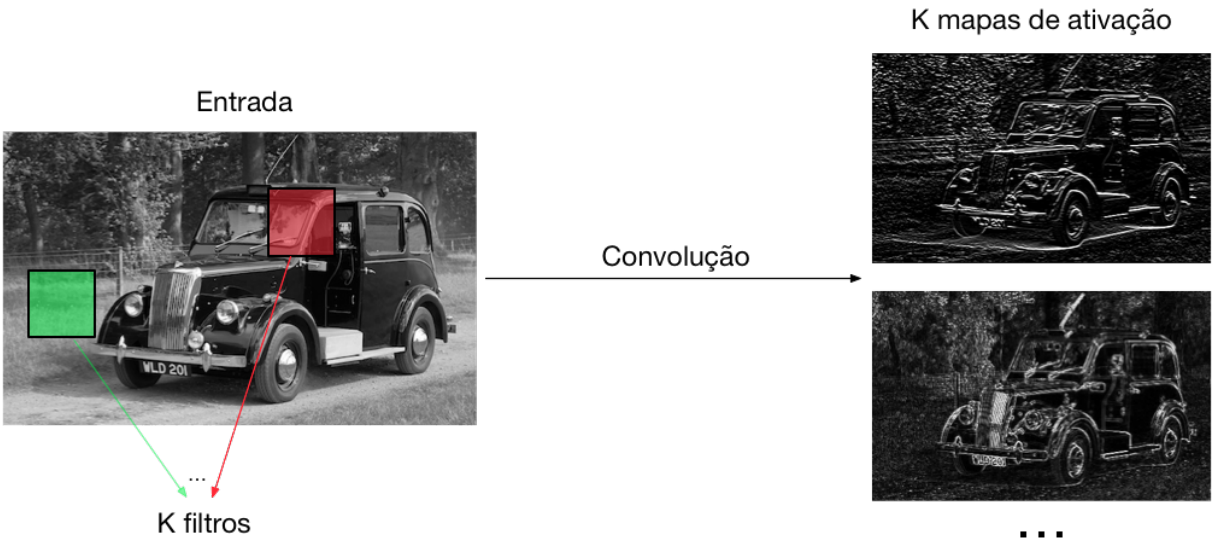


Figura 3.6: Uma camada de convolução recebendo uma imagem como entrada e produzindo K mapas de ativação.

O papel da retropropagação em camadas convolucionais é de alterar os valores dos *kernels* dos filtros para que os mesmos sejam ativados quando a entrada conter certas características observadas em múltiplos exemplos do conjunto de treinamento. As primeiras camadas convolucionais de uma CNN tendem a ser ativadas para características de baixo nível (e.g. bordas com uma determinada orientação) e as camadas mais profundas são responsáveis por extrair aspectos de alto nível (e.g. a mão de uma pessoa).

Como já foi descrito, redes neurais artificiais clássicas são inviáveis para treinamento em larga escala sobre grandes entradas, como é o caso de imagens digitais. Um dos fatores principais que tornam CNNs apropriadas para esse tipo de processamento é a consideração de que se um filtro é capaz de detectar um certo tipo de característica na posição (x_1, y_1) da entrada, então também será capaz de fazer o mesmo em outra posição (x_2, y_2). A partir dessa ideia, é possível que os parâmetros da camada sejam compartilhados. Isso tem como consequência uma redução expressiva no número de parâmetros a serem aprendidos através da retropropagação.

Por exemplo, considere uma imagem de 3 canais, de dimensão total $200 \times 200 \times 3$. Considere uma camada CONV que possui 20 filtros de tamanho $5 \times 5 \times 3$ e que recebe a imagem como entrada. Devido ao compartilhamento de parâmetros, cada filtro possui $5 \cdot 5 \cdot 3 = 75$ parâmetros a serem aprendidos, totalizando $75 \cdot 20 = 1.500$ parâmetros (sem contabilizar *bias*) na camada CONV. Por outro lado, caso uma camada totalmente conectada fosse utilizada, o total de parâmetros da camada para produzir os 20 mapas de ativação seria $(200 \cdot 200 \cdot 20) \cdot (200 \cdot 200 \cdot 3) = 96.000.000.000$, um valor de significante maior magnitude presente em uma única camada.

Uma maneira interessante de observar o que uma rede neural convolucional está aprendendo é extrair em formato de imagem os mapas de ativação produzidos durante o processo de propagação pelas camadas convolucionais. A Figura 3.7 mostra um subconjunto dos mapas de ativação da 1ª camada CONV presente na CNN *AlexNet* [15], que foi proposta para realizar classificação de objetos em imagens. É possível notar que diferentes filtros destacam diferentes regiões de cada imagem, sendo informações propagadas para as demais camadas, que extrairão informações de mais alto nível com base nos mapas de ativação.

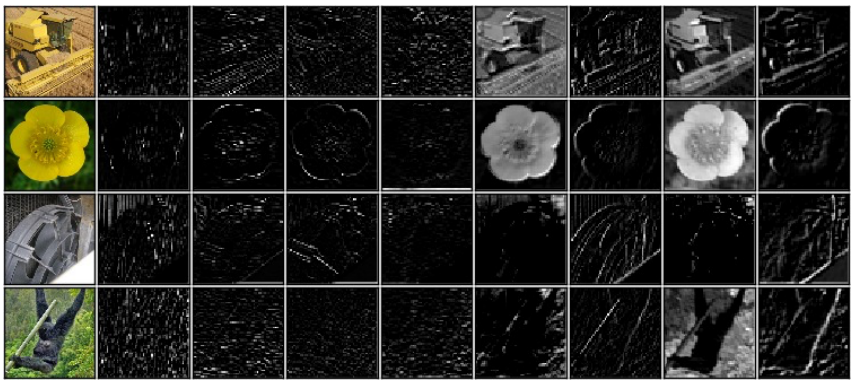


Figura 3.7: Mapas de ativação de 8 filtros da 1ª camada de convolução da *AlexNet* [2].

[2]Adaptado de [15]

Camada de *pooling* (POOL)

A camada de *pooling* é responsável por controlar o sobre-ajuste durante o treinamento da rede neural convolucional. É uma camada que geralmente recebe como entrada a saída de uma camada de convolução. A operação da camada POOL consiste em realizar uma subamostragem de cada mapa de ativação (i.e. canal) da entrada. A saída da camada de *pooling* é o mesmo número de mapas de ativação recebidos na entrada, mas subamostrados por um mesmo fator.

A operação de *pooling* pode ser descrita como a convolução da entrada por um filtro de tamanho pré-definido que passa sobre a entrada com um certo *stride*, computando uma função que produz um escalar dada a região em que a janela do filtro se encontra. Uma função comumente utilizada é a `max` (*max pooling*).

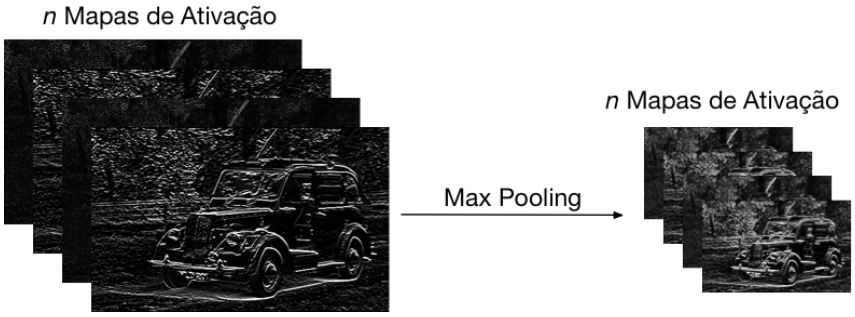


Figura 3.8: Uma camada de *max pooling* recebendo n mapas de ativação como entrada.

Como exemplo, considere um mapa de ativação M de tamanho 4×4:

$$M = \begin{bmatrix} 54 & 67 & 165 & 32 \\ 5 & 45 & 59 & 1 \\ 42 & 22 & 211 & 93 \\ 55 & 34 & 11 & 57 \end{bmatrix} \tag{3.17}$$

Aplicando-se a operação de *max pooling* usando um filtro de tamanho 2×2 e *stride* 2, a saída é dada por M':

$$M' = \begin{bmatrix} 67 & 165 \\ 55 & 211 \end{bmatrix} \tag{3.18}$$

Como é possível notar, M' foi reduzido por um fator 2, tornando a entrada mais simples para as próximas camadas. Além disso, a operação de *pooling* faz com que a rede neural convolucional se torne invariante a pequenas transformações na imagem. Por outro lado, existe a perda de informação, que até um certo nível pode ser benéfica para evitar o sobre-ajuste. Para não haver excesso, [29] recomenda o uso de um *kernel* de *pooling* de

dimensões 2×2 e *stride* 2. Por outro lado, caso o fator de subamostragem do *pooling* for muito alto, o modelo pode começar a sofrer com sub-ajuste.

Camada Totalmente Conectada (FC - *fully connected*)

Uma camada totalmente conectada é equivalente à camada presente em uma rede neural artificial clássica, na qual todas as suas unidades estão conectadas com a camada anterior. Esse tipo de camada é geralmente utilizada no fim de uma CNN, onde a dimensionalidade do dado é geralmente menor que a entrada original. A camada totalmente conectada é usada como uma forma de aprender funções não-lineares com base na combinação das características extraídas pelas camadas anteriores. Portanto, é geralmente responsabilidade da camada totalmente conectada dar a saída final de uma CNN, a partir de alguma função de ativação.

3.2.2 Redes Neurais Convolucionais Multicanais

Redes neurais convolucionais clássicas possuem um único canal de entrada. Por exemplo, uma imagem é dada para a rede, que é repassada linearmente entre as camadas, até a última camada produzir uma saída final.

Uma CNN de N canais é caracterizada por receber N entradas que são processadas por camadas paralelas até um determinado ponto, onde os fluxos se unem, para que então um processamento serial seja prosseguido. Em trabalhos recentes, é comum que o ponto de concatenação dos dados esteja presente antes da primeira camada totalmente conectada da rede, ou seja, o processamento paralelo se concentra entre as camadas de convolução.

O problema de reconhecimento de ações em vídeos é um tema que tem sido explorado através de CNNs multicanais. O motivo por trás disso é que uma CNN convencional não seria capaz de extrair a informação temporal presente no vídeo de entrada, já que a mesma recebe como entrada os quadros do vídeo de forma independente. O trabalho de Karpathy et al. [1] propõe uma metodologia baseada em CNNs de múltiplos canais capazes de gerar rótulos da ação predominante em um vídeo. Em uma arquitetura proposta, o autor propõe o uso de uma rede neural convolucional de 2 canais tal que os canais recebem dois quadros do vídeo de entrada separados em aproximadamente um terço de segundo.

Outra utilidade no uso de múltiplos canais em CNNs é ainda proposta no trabalho de Karpathy et al. [1], e consiste na redução da dimensionalidade da entrada da rede. O treinamento de redes neurais profundas em conjuntos de larga escala (e.g. milhares de vídeos) é um processo que pode demorar semanas para ser concluído nas melhores GPUs disponíveis atualmente. A partir disso, o autor propõe uma arquitetura de multiresolução, com dois canais titulados *fóvea* e *contexto*.

Considere uma imagem de entrada original de tamanho W x H. O canal de fóvea recebe a região central da imagem de resolução $\frac{W}{2}$x$\frac{H}{2}$. O canal de contexto recebe a imagem original subamostrada em 50%. A ideia por trás da metodologia é que os canais possam evitar que a rede perca performance nos resultados, apesar de receber um conjunto de entrada de dimensionalidade reduzida em 50% em relação à base de dados original. A Figura 3.9 mostra a arquitetura proposta pelo trabalho.

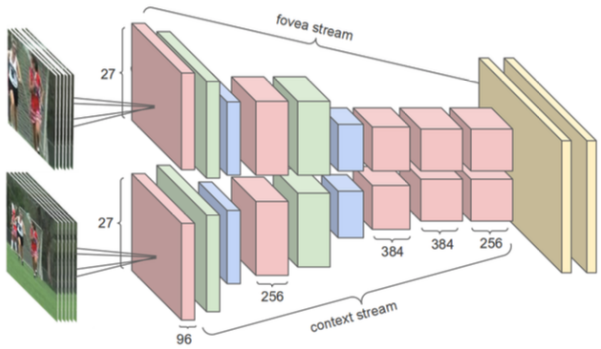


Figura 3.9: Arquitetura multicanal proposta por Karpathy et al. [1] para redução de dimensionalidade da entrada da rede. As camadas vermelhas correspondem às camadas de convolução, as verdes são operações de normalização, as azuis são camadas de *pooling*, e as amarelas são camadas totalmente conectadas.[3]

De acordo com os resultados obtidos por Karpathy et al. [1], a arquitetura com os canais de fóvea e contexto obteve uma acurácia de classificação de 60,0%, um valor melhor que o obtido através da arquitetura de único canal, que apresentou acurácia de 59,3%.

[3]Imagem extraída de [1].

Capítulo 4

Veículos Autônomos

Veículos autônomos, de acordo com [4], consistem em automóveis capazes de compreender o ambiente ao seu redor e de se auto-guiarem. De acordo com um documento publicado pela organização *Society of Automotive Engineers* (SAE International) [30], a tecnologia de direção autônoma é classificada em seis níveis:

- Nível 0 (sem automação): não há sistema autônomo no veículo, toda a responsabilidade de direção é do motorista.
- Nível 1 (assistência ao motorista): o controle do veículo é compartilhado entre o motorista e o sistema autônomo. Neste caso, algumas funções específicas são automatizadas, como ocorre na tecnologia *cruise control*, na qual o veículo é capaz de manter uma velocidade pré-determinada.
- Nível 2 (automação parcial): o controle de aceleração, freio, e direção é feito através do sistema autônomo. No entanto, é necessário que o motorista mantenha atenção total e esteja preparado para intervir a qualquer momento.
- Nível 3 (automação condicional): o sistema não requer que o motorista tenha total atenção. No entanto, é importante que o motorista esteja preparado para tomar controle do veículo caso o sistema o notifique.
- Nível 4 (alto nível de automação): o sistema não requer atenção do motorista, sendo capaz de lidar com situações em que o motorista não atendeu a notificação de intervenção ao controle do veículo.
- Nível 5 (automação completa): o sistema é completamente autônomo, sendo capaz de dirigir o veículo em qualquer tipo de situação. Sendo assim, não é necessária a presença de um humano no automóvel.

Veículos autônomos necessitam de um conjunto de periféricos que permitam uma interface entre o meio físico e o sistema de automação. A partir da fusão dos dados

obtidos através de diferentes sensores, é esperado que o sistema autônomo seja capaz de reconhecer o meio ao seu redor, uma condição essencial para que o veículo possa ser controlado de maneira eficiente. Alguns sensores comumente utilizados em veículos autônomos são:

1. GPS (*Global Positioning System*): localização e mapeamento são tarefas de grande importância no contexto de veículos autônomos para navegação e determinação de rotas. Com isso, um dispositivo GPS pode fornecer a geolocalização do automóvel com precisão de aproximadamente 5 metros.
2. IMU (*Inertial Measurement Unit*): um IMU consiste em um periférico eletrônico capaz de coletar a velocidade angular e a aceleração linear do veículo. Como o GPS pode ser impreciso e estar indisponível em certas situações, o IMU pode ser utilizado para determinar a posição e orientação do carro autônomo.
3. Radar: ondas de rádio são utilizadas em veículos autônomos para mapear objetos (e.g. carros, pedestres) ao redor do veículo. Radares também apresentam bons resultados em situações de difícil visibilidade, como chuva, neblina, neve, etc.
4. Lidar: é uma tecnologia que determina a distância entre objetos através da emissão de *lasers*, permitindo que seja realizado um mapeamento tridimensional dos arredores do veículo. O Lidar geralmente produz resultados de maior resolução que um radar. No entanto, o seu custo de produção atual é consideravelmente elevado.
5. Câmera: veículos autônomos usam uma série de câmeras em diferentes regiões do veículo para que o sistema possa interpretar os arredores do carro a partir de imagens que contêm informações de grande importância, como cor e textura. O gargalo do uso de câmeras em veículos autônomos é o processamento intensivo das imagens (e.g. os quadros capturados podem ser usados como entrada em redes neurais artificiais profundas).

4.1 Veículos Autônomos e Aprendizado de Máquina

Algoritmos de aprendizado de máquina são comumente utilizados para processar os dados obtidos a partir dos sensores instalados no veículo autônomo. Isso se deve ao fato de que redes neurais artificiais profundas apresentam bons resultados para uma série de subproblemas, como: i) detecção de pista [31], ii) detecção de pedestres [32], iii) detecção de veículos a partir dos dados produzidos por Lidar [33].

Existe um grande desafio em aberto sobre a utilização de técnicas de aprendizado de máquina para resolver problemas que requerem uma taxa de erro extremamente baixa.

Além disso, de acordo com [34] técnicas de aprendizado de máquina são sujeitas a exemplos adversários, que são entradas propositalmente modificadas para que o modelo produza uma resposta incorreta. Esta condição pode ser encontrada no exemplo da Figura 4.1, originalmente proposto por [35]. Após a inserção de um pequeno ruído na imagem original, a rede neural convolucional classificou o animal de maneira errônea e com um nível de confiança bastante elevado. Portanto, é importante que sistemas de veículos autônomos sejam capazes de tratar problemas como esse, de forma a não comprometer a segurança dos usuários. No momento, isso é um tópico de pesquisa ativa e de bastante relevância para a evolução da inteligência artificial em produtos finais.

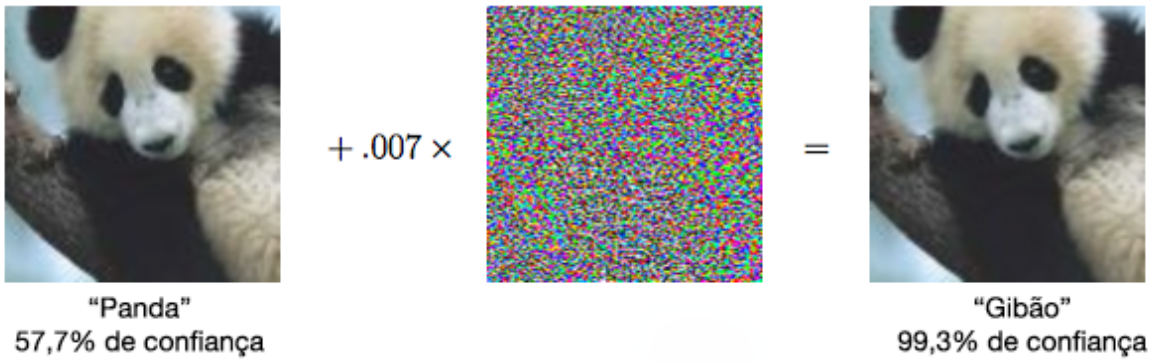


Figura 4.1: Exemplo de uma entrada adversária para a rede neural convolucional GoogLeNet [2]. [1]

4.2 Principais Trabalhos Relacionados

Veículos autônomos é um tópico que já tem sido explorado por algumas décadas. Em 1989, Pomerleau et al. [36] descreveu a construção de um veículo autônomo baseado em redes neurais artificiais. A rede neural proposta, chamada ALVINN (*Autonomous Land Vehicle In a Neural Network*), é responsável por prover uma orientação ao veículo. A arquitetura da rede (vista na Figura 4.2) consiste em uma RNA clássica com uma única camada intermediária, contendo 29 neurônios totalmente conectados com as unidades de entrada.

As entradas da RNA são divididas em três conjuntos:

1. "Retina" 1: imagem capturada pela câmera instalada no veículo, de dimensões 30×32.

[1] Adaptado de [35].

2. “Retina” 2: imagem de dimensão 8×32 obtida através de um telêmetro a laser, que representa a relação de proximidade entre o veículo e a área capturada através da “Retina” 1.

3. Unidade de intensidade da estrada: uma única unidade que indica se a estrada está mais clara ou mais escura que as demais regiões (e.g. calçada, gramado) na imagem de entrada anterior.

A saída da RNA possui 46 unidades. Uma delas consiste na intensidade da estrada, que é usada como entrada na próxima iteração do treinamento. As demais 45 unidades representam a curvatura que o veículo deve seguir, ou seja, a unidade do meio representa a ação de dirigir reto e as unidades da esquerda e direita correspondem ao ato de realização de curvas. De acordo com o trabalho, a RNA foi treinada por 40 épocas em uma base de dados de 1200 entradas. A rede foi testada em um carro de pesquisa (visto na Figura 4.3), tendo sido capaz de trafegar de forma autônoma por um caminho de 400 metros sob velocidade de 0,5 m/s.

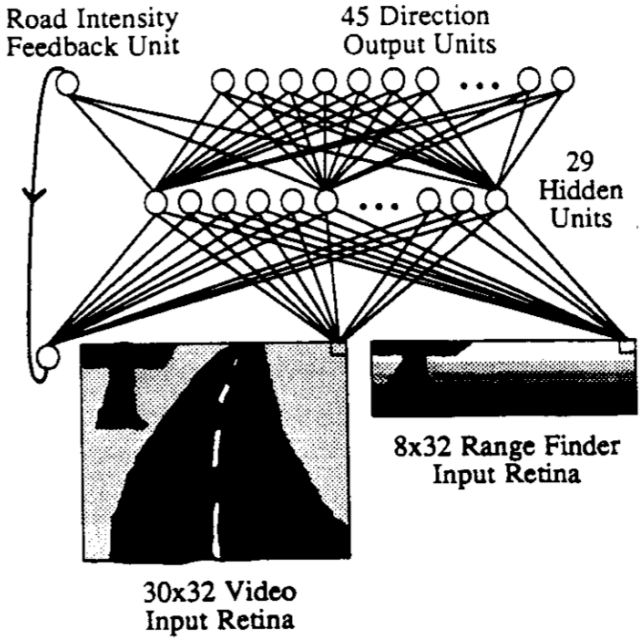


Figura 4.2: Arquitetura da rede neural artificial ALVINN. [2]

[2]Extraído de [36].

Figura 4.3: NAVLAB: o veículo autônomo de testes da rede neural artificial ALVINN. [3]

Em [37] é proposto um sistema autônomo de navegação titulado de *DeepDriving* baseado em redes neurais convolucionais. A base de treinamento utilizada consiste em 484.815 imagens sintéticas de resolução 280×210 obtidas através de um simulador de corrida. A partir disso, é realizado o treinamento da rede neural convolucional AlexNet [15] com a base de dados sobre 140.000 iterações e produzindo um total de 13 valores de saída. Um dos valores corresponde ao ângulo entre a frente do carro e a tangente da pista. Os demais consistem em distâncias do veículo a demais elementos da cena (e.g. a outro veículo ou a marcação de alguma pista lateral).

Durante a etapa de teste, a velocidade, a aceleração, o freio, e o ângulo de direção são computados a partir dos valores de saída da CNN. Em seguida, todos os dados obtidos são usados como entrada em um algoritmo que descreve a lógica de controle do veículo.

A partir de uma análise qualitativa, os resultados obtidos permitem que o sistema autônomo controle o veículo no simulador sem colisões e de forma natural. Quantitativamente, a técnica proposta apresentou melhores resultados do que um sistema baseado no descritor GIST [38] em termos de aproximação do valor do ângulo e das distâncias estimadas.

Os trabalhos [9] e [39], desenvolvidos pela empresa de tecnologia NVIDIA, propõem uma rede neural convolucional titulada *PilotNet* capaz de estimar o ângulo de direção de um veículo, dado como entrada o quadro de uma câmera instalada na parte frontal do carro.

A arquitetura da CNN consiste em 9 camadas, sendo uma de normalização, cinco de convolução, e três camadas totalmente conectadas. É reportado que a rede foi montada de forma empírica através de múltiplos testes com diferentes configurações de camada. É esperado que a extração de características da entrada seja realizada nas camadas iniciais

[3] Extraído de `https://www.cs.cmu.edu/Groups/ahs/navlab_list.html`

(i.e. convolução), e que as últimas camadas (i.e. completamente conectadas) sirvam como controle de direção.

De forma a observar as características extraídas pela *PilotNet*, é proposta uma técnica de visualização dos objetos salientes na imagem de entrada. Isso é feito através de um algoritmo baseado em deconvolução [40], que encontra as regiões da imagem que possuem os maiores níveis de ativação nos mapas produzidos pelas camadas de convolução da rede. Na Figura 4.4 é possível visualizar as regiões de maior saliência em uma imagem de entrada, destacadas em verde.

A rede *PilotNet* foi testada em um carro, e foi capaz de realizar um percurso de 15 minutos em área urbana e sem intervenção humana por 98% do tempo.

Figura 4.4: Regiões de alta saliência detectadas pela *PilotNet*. [4]

[4]Adaptado de [39].

Capítulo 5

Métodologia Proposta

Como visto nos trabalhos anteriores apresentados na Seção 4.2, redes neurais artificiais têm apresentado resultados promissores na construção de sistemas de veículos autônomos. A partir disso, este trabalho propõe um método *capaz de estimar o ângulo de direção de um veículo baseado em redes neurais convolucionais (CNNs) multicanais*. O sistema desenvolvido recebe como entrada uma imagem colorida no padrão de cores *RGB* da vista frontal do veículo, e produz como saída a estimativa do ângulo do volante naquele instante. Um diagrama de alto nível da metodologia proposta pode ser vista na Figura 5.1.

Primeiramente, este capítulo define a base de dados que é utilizada como entrada para o nosso método. Em seguida, é apresentada a arquitetura base da rede neural convolucional utilizada. Com isso, são descritas as modificações propostas da rede utilizada como base, usando múltiplos canais de entrada. Por fim, os modelos propostos são detalhadamente definidos.

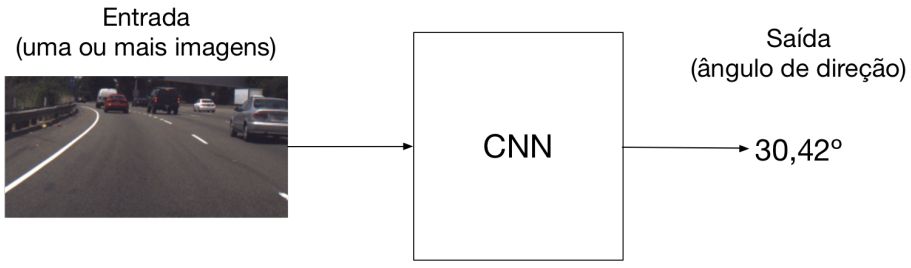


Figura 5.1: Esquema em alto nível do sistema de estimação de direção do veículo.

5.1 Definindo a base de dados

Dado que a metodologia proposta neste trabalho faz o uso de redes neurais convolucionais como elemento fundamental no sistema de aprendizado, o uso de uma base de dados é

essencial para o processo de treinamento, validação e teste dos modelos produzidos.

A partir disso, a metodologia proposta se baseia no uso de uma base de dados fornecida pela organização *comma.ai*[1]. O conjunto de dados é disponibilizado no formato *HDF5* (*Hierarchical Data Format version 5*)[2], um padrão otimizado para o armazenamento de um grande volume de informações na forma de um único arquivo de extensão `.h5`. Internamente, o formato HDF5 organiza os dados em agrupamentos definidos como *databases* de forma hierárquica, similar a uma estrutura de diretórios em um sistema de arquivos.

A base de dados é composta por 11 (onze) vídeos com resolução 320x160 *pixels* de duração variável e gravados a 20 (vinte) quadros por segundo (*fps*) com uma câmera instalada no para-brisas de um carro (i.e. *Acura ILX 2016*), capturando quadros de sua visão frontal durante o dia e noite, trafegando predominantemente em rodovias. A Figura 5.2 mostra um exemplo de quadro presente na base de dados. No total, o conjunto de dados possui 522.434 quadros capturados, resultando em aproximadamente 7 (sete) horas de gravação.

Figura 5.2: Um quadro da base de dados da *comma.ai*.

Cada vídeo é descrito na base de dados como um par de arquivos no formato *HDF5*. O primeiro contém os dados crus (i.e. valores dos *pixels* RGB) de cada quadro do vídeo ordenados cronologicamente. O segundo contém dados capturados por sensores instalados no veículo (e.g. ângulo do volante, velocidade, freio), mapeados para o identificador do quadro que foi capturado pela câmera no mesmo instante.

Para os objetivos desse trabalho, os únicos dados levados em consideração são os quadros do vídeo (padrão de cores *RGB*) e os ângulos de direção do veículo. A medida do

[1]Base de dados disponível sob licença CC BY-NC-SA 3.0 em: `https://archive.org/details/comma-dataset`

[2]`http://www.hdfgroup.org/HDF5/`

ângulo é dada em graus e corresponde ao ângulo de rotação do volante do carro utilizado. Toda medida de ângulo presente na base de dados pertence ao intervalo $[-502.3, 512.6]$.

5.2 Arquitetura base da CNN

O objetivo da metodologia proposta, basicamente, é resolver um problema de regressão: *estimar o ângulo de direção de um veículo dado um conjunto de imagens de entrada.* Portanto, este trabalho se baseia na utilização de uma rede neural convolucional preexistente, proposta pela organização *comma.ai*[3], aqui definida como arquitetura de base.

A CNN de base funciona de acordo com o paradigma de aprendizado supervisionado, ou seja, a rede necessita ser treinada sob uma base de dados tal que cada exemplo consiste em um quadro capturado pelas câmeras e em um ângulo de direção do volante naquele determinado instante. Com isso, permite-se que um erro seja computado e propagado a todas unidades da CNN, para que então os seus parâmetros sejam ajustados.

A função de ativação utilizada na rede neural convolucional de base foi a ELU (*Exponential Linear Unit*) [41], definida pela Equação 5.1 e pelo gráfico presente na Figura 5.3a.

$$\varepsilon(x) = \begin{cases} x & x \geq 0 \\ \beta\ (e^x - 1) & x < 0 \end{cases} \tag{5.1}$$

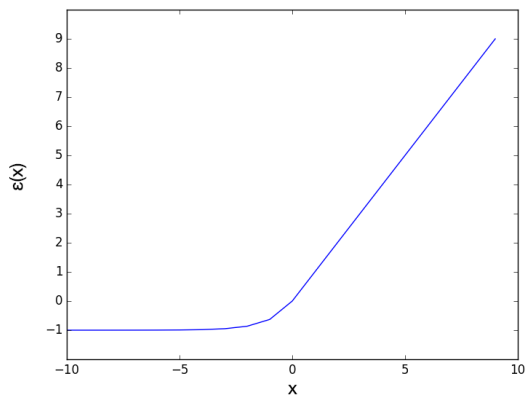
Tal que x corresponde à entrada da função de ativação, β corresponde a um hiperparâmetro que foi fixado em 1 para este trabalho. Além disso, o gradiente da função de ativação ELU é dado pela equação 5.2 e pelo gráfico presente na Figura 5.3b.

$$\varepsilon'(x) = \begin{cases} 1 & x \geq 0 \\ \varepsilon(x) + \beta & x < 0 \end{cases} \tag{5.2}$$

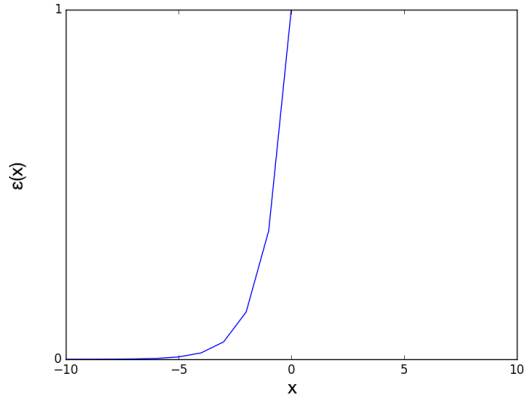
De acordo com Clevert et al. [41], um dos maiores benefícios da função de ativação ELU é que o seu uso pode acelerar o processo de aprendizado durante o treinamento da rede. Isso se deve principalmente ao fato de que a função pode produzir valores negativos, permitindo que a ativação média das unidades seja centralizada em 0. Além disso, em experimentos realizados por Clevert et al. [41], o uso da função ELU trouxe melhores resultados de generalização para problemas de classificação quando utilizada em redes neurais convolucionais com mais de 5 camadas.

A Figura 5.4 apresenta o diagrama da arquitetura da CNN utilizada. Em maiores detalhes, a rede neural convolucional usada como base neste trabalho é descrita por 13

[3]Repositório do projeto disponível em: `https://github.com/commaai/research`



(a) Função de ativação ELU.



(b) Gradiente da função de ativação ELU.

Figura 5.3: Função de ativação ELU e o seu correspondente gradiente.

camadas e contém 6.621.809 parâmetros a serem aprendidos. Em ordem topológica, as camadas da rede são descritas por:

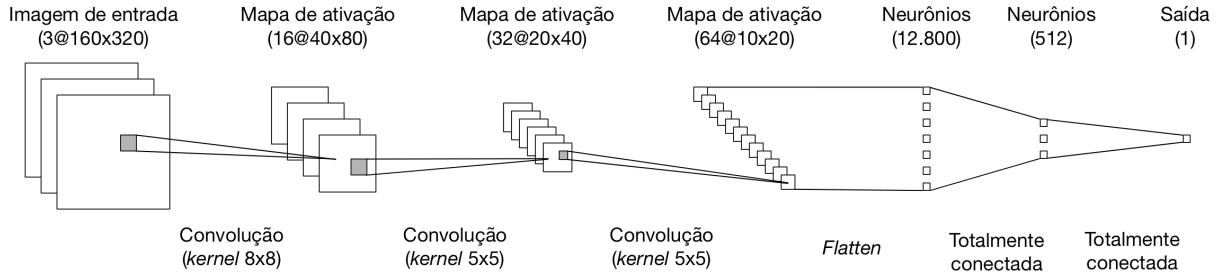


Figura 5.4: Diagrama da arquitetura da rede neural convolucional de base.

1. **Normalização**: uma imagem no padrão *RGB* é dada como entrada para a CNN, sendo que o valor de cada *pixel* em cada canal de cor pertence ao intervalo $[0, 255]$. A primeira camada da rede é responsável por normalizar os valores dos *pixels* no intervalo $[-1, 1]$. Matematicamente, a seguinte operação é realizada:

$$x' = \frac{x}{127,5} - 1 \tag{5.3}$$

   Dimensão da saída: $3 \times 160 \times 320$

2. **Camada de convolução (CONV)**: os parâmetros da primeira camada de convolução estão presentes na Tabela 5.1.

Tabela 5.1: Parâmetros da 1ª camada de convolução.

Número de filtros	Dimensão do *kernel*	*Stride*	*0-padding*
16	8×8	4×4	2

Dimensão da saída: $16 \times 40 \times 80$

3. ***ELU***
   Dimensão da saída: $16 \times 40 \times 80$

4. **Camada de convolução (CONV)**: os parâmetros da segunda camada de convolução estão presentes na Tabela 5.2.

Tabela 5.2: Parâmetros da 2ª camada de convolução.

Número de filtros	Dimensão do *kernel*	*Stride*	*0-padding*
32	5×5	2×2	2

   Dimensão da saída: $32 \times 20 \times 40$

5. ***ELU***

   Dimensão da saída: $32 \times 20 \times 40$

6. **Camada de convolução (CONV)**: os parâmetros da terceira camada de convolução estão presentes na Tabela 5.3.

Tabela 5.3: Parâmetros da 3ª camada de convolução.

Número de filtros	Dimensão do *kernel*	*Stride*	*0-padding*
64	5×5	2×2	2

   Dimensão da saída: $64 \times 10 \times 20$

7. ***Flatten***: a função da camada *flatten* é de remodelar a entrada como um vetor de características. Por exemplo, dada uma uma matriz de dimensão 100x42, a camada produzirá um vetor $\mathbb{R}^{4200}$. Essa etapa é realizada para que as informações espaciais aprendidas através das convoluções sejam transferidas para camadas totalmente conectadas.

   Dimensão da saída: 1×12.800

8. ***Dropout***: a camada de *dropout* foi proposta originalmente por Srivastava et al. [42], e é utilizada como uma forma de regularização, para evitar a ocorrência de sobreajuste. Essa camada é utilizada somente na fase de treinamento, e sua operação consiste em atribuir o valor 0 em cada unidade de entrada com uma probabilidade p.

   Probabilidade de dropout: 20%
   Dimensão da saída: 1×12.800

9. ***ELU***
   Dimensão da saída: 1×12.800

10. **Camada totalmente conectada (FC)**
    Dimensão da saída: 1×512

11. ***Dropout***: assim como a camada de *dropout* usada anteriormente, é utilizada somente durante o processo de treinamento.
    Probabilidade de dropout: 50%
    Dimensão da saída: 1×512

12. ***ELU***
    Dimensão da saída: 1×512

13. **Camada totalmente conectada (FC)**: a última camada da rede consiste em uma unidade totalmente conectada com as 512 unidades da camada anterior. A saída da camada produz um valor $y \in \mathbb{R}$, que consiste no ângulo de direção do veículo no instante que a imagem de entrada foi capturada.

    Dimensão da saída: 1×1

5.3 Arquiteturas propostas

A partir da arquitetura base apresentada na Seção 5.2, a metodologia deste trabalho propõe a construção de duas redes neurais convolucionais compostas de múltiplos canais de entrada. A motivação por trás da ideia é observar o impacto que CNNs multicanais apresentam para o problema de estimativa do ângulo de direção de um veículo.

5.3.1 Rede neural convolucional de 2 canais

A primeira CNN multicanal proposta pelo trabalho consiste na duplicação da 1ª camada até a 10ª (i.e. ELU). Ou seja, a rede possui dois canais de entrada, que são processados paralelamente por todas as camadas de convolução. As saídas produzidas pelas camadas ELU são então concatenados e seguem como entrada para a primeira camada totalmente conectada. O restante do processo da rede é realizado em um único canal, assim como a CNN original. Um diagrama da rede pode ser visto na Figura 5.5.

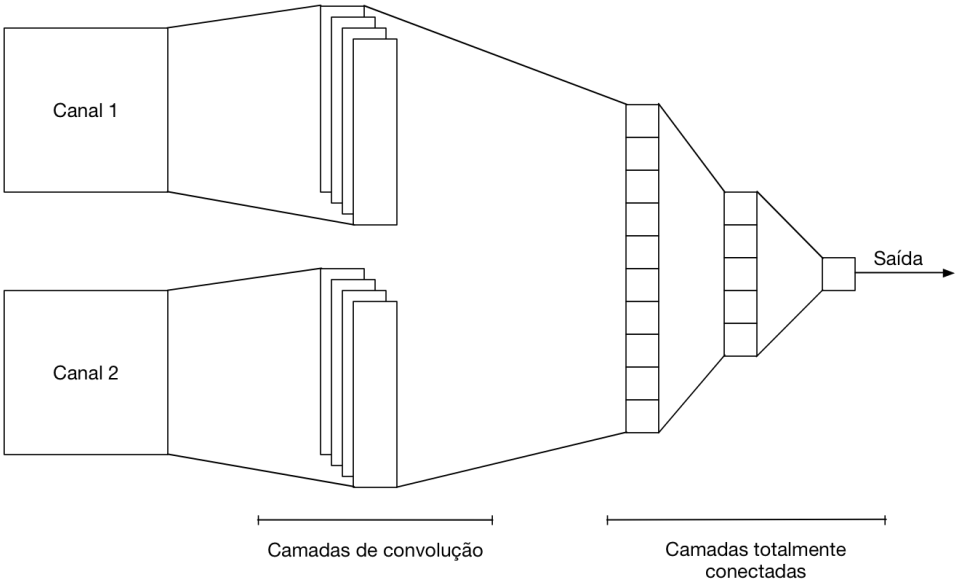


Figura 5.5: Arquitetura proposta de 2 canais de entrada.

5.3.2 Rede neural convolucional de 3 canais

A segunda rede proposta por este trabalho consiste em 3 canais de entrada. Assim como na CNN de 2 canais, as primeiras 7 camadas da rede original foram replicadas. O diagrama na Figura 5.6 mostra como a rede é estruturada.

5.4 Modelos desenvolvidos

Com a base de dados e as arquiteturas das redes neurais convolucionais definidas, este trabalho propõe um *framework* para a geração dos modelos treinados, que são capazes de receber um conjunto de imagens capturadas da vista frontal de um veículo e de produzir como saída o ângulo estimado do volante no instante que o conjunto de quadros foi capturado.

O *framework* proposto consiste primeiramente em modificar a base de dados original para que o treinamento das arquiteturas multicanais seja possível. Em seguida, é realizado o treinamento de 5 (cinco) combinações de modelos distintos. Por último, os modelos treinados podem ser utilizados para teste. Um fluxograma do *framework* pode ser visualizado na Figura 5.7.

5.4.1 Preparação da Base de Dados

A rede neural convolucional originalmente proposta pela *comma.ai* possui um único canal de entrada. Dado que a metodologia deste trabalho propõe o uso de CNNs multicanais,

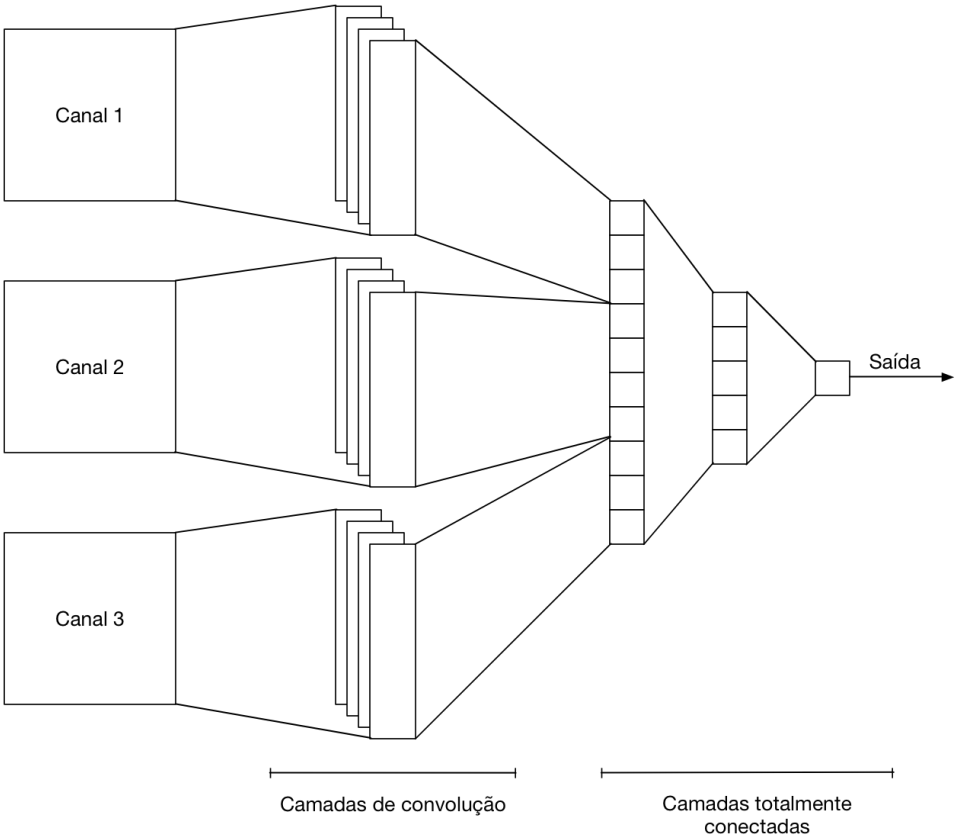


Figura 5.6: Arquitetura proposta de 3 canais de entrada.

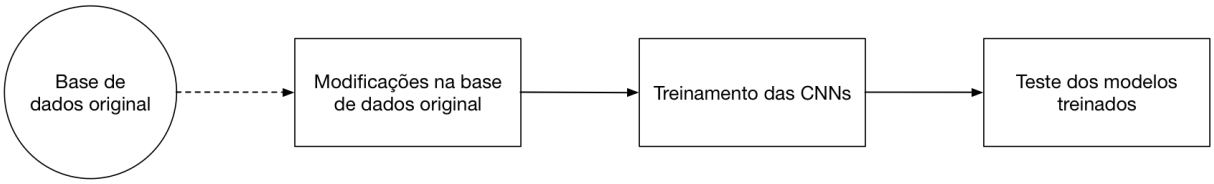


Figura 5.7: Fluxograma do *framework* proposto.

foi necessário uma adaptação das bases de dados a partir do conjunto original (para referência futura, a base original será denotada como `BASE_1`) de acordo com a arquitetura de cada modelo. A partir disso, foram criados novos arquivos no formato *HDF5* com 4 configurações distintas:

- `BASE_2`: quadro original + quadro subamostrado em 50% da resolução espacial (i.e. 160x80 *pixels*).
- `BASE_3`: quadro original + região central da imagem em escala original (i.e. 160x80 *pixels*).
- `BASE_4`: quadro original + quadro subamostrado em 50% da resolução espacial (i.e. 160x80 *pixels*) + região central da imagem em escala original (i.e. 160x80 *pixels*).

- `BASE_5`: quadro subamostrado em 50% da resolução espacial (i.e. 160x80 *pixels*) + região central da imagem em resolução original (i.e. 160x80 *pixels*)

Ou seja, para cada arquivo *HDF5* que possui os dados sobre os quadros capturados pela câmera, foi gerado um novo arquivo para cada configuração. Em termos de estrutura do padrão *HDF5*, cada canal foi inserido no arquivo como um novo *database*. Para a geração dos novos arquivos *HDF5*, foi utilizada a biblioteca `h5py`, que fornece uma interface de alto nível em *Python* para leitura e escrita de arquivos no formato `.h5`. A partir disso, foi criado um programa que realiza as seguintes operações, em ordem:

- Recebe um arquivo *HDF5* da base de dados original que contém os quadros codificados.
- Decodifica cada quadro presente no arquivo como uma imagem *RGB* e realiza a transformação necessária (i.e. subamostragem ou recorte de uma região de interesse). Essa etapa foi realizada através da biblioteca *OpenCV* [43], uma biblioteca para processamento de imagens e visão computacional.
- Codificação de cada imagem resultante e inserção da mesma em um novo arquivo no formato *HDF5*.

5.4.2 Treinamento das CNNs

A partir das bases de dados modificadas e das arquiteturas propostas, cinco modelos distintos foram treinados. Cada modelo usou um dos conjuntos de dados apresentados na seção anterior (i.e. `BASE_1` a `BASE_5`). Antes de iniciar o processo de treinamento de cada modelo, a base de dados utilizada, composta de 11 (onze) vídeos, deve ser dividida em três subconjuntos mutualmente exclusivos:

- Base de treinamento: 7 (sete) vídeos.
- Base de validação: 2 (dois) vídeos.
- Base de testes: 2 (dois) vídeos.

A base de treinamento é utilizada como entrada durante o treinamento do modelo, sendo a única que faz parte do processo de ajuste dos parâmetros da rede. A base de validação é utilizada para avaliar o aprendizado do modelo após cada época de treinamento. Por fim, a base de testes é utilizada após o modelo ter sido treinado, e serve para observar o quanto o modelo generalizou para exemplos nunca vistos anteriormente. Tal partição é feita de maneira aleatória, e é realizada para que o modelo possa ser corretamente comparado com outros.

Os modelos foram treinados usando o método de otimização estocástica *Adam* [44], sendo utilizados os hiperparâmetros recomendados pelos autores do algoritmo. As demais configurações de treinamento podem ser vistas na Tabela 5.4.

Tabela 5.4: Configurações de treinamento dos modelos propostos.

Tamanho de cada lote	256 quadros
Tamanho de cada época	10.000 quadros
Número de quadros processados para validação (após cada época)	1000 quadros
Função de custo	Erro quadrático médio
Taxa de aprendizado	0,001

Modelo 1 (treinada com `BASE_1`)

Neste modelo, a CNN original de único canal é utilizada. Isso é feito para que seja possível a comparação da rede original com os demais modelos.

Figura 5.8: Único canal de entrada no modelo 1.

Modelo 2 (treinada com `BASE_2`)

O segundo modelo faz uso da CNN de 2 canais. Além da imagem original, é dado como entrada o quadro subamostrado em 50%. Com isso, podemos observar como a CNN se comporta com a entrada sendo dada em diferentes escalas.

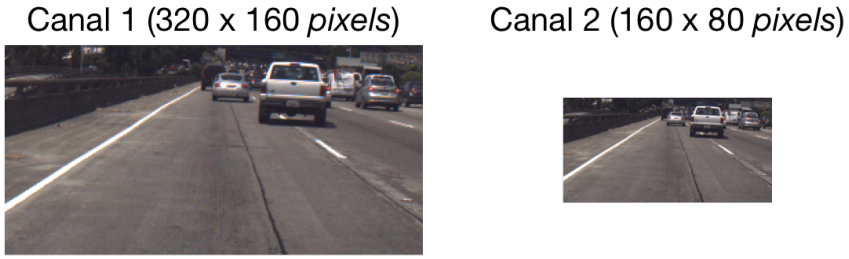


Figura 5.9: Canais de entrada do modelo 2.

Modelo 3 (treinada com `BASE_3`)

O terceiro modelo faz uso da CNN de 2 canais. Além da imagem original, é dada como entrada a região central do quadro em resolução original. A partir deste modelo, podemos observar como a rede pode reagir recebendo em um canal a região do quadro que provavelmente contém os objetos mais próximos e que estão presentes na mesma faixa da pista que o veículo trafega no momento.

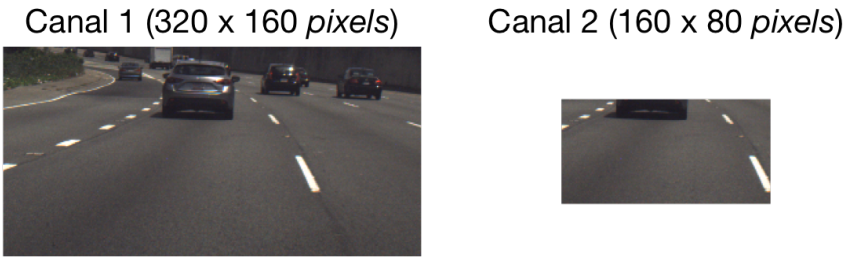


Figura 5.10: Canais de entrada do modelo 3.

Modelo 4 (treinada com `BASE_4`)

O 4º modelo faz uso da CNN de 3 canais. O objetivo de criação deste modelo é observar se a rede pode trazer uma melhoria de performance recebendo os dados adicionais que são dados nos modelos 2 e 3.

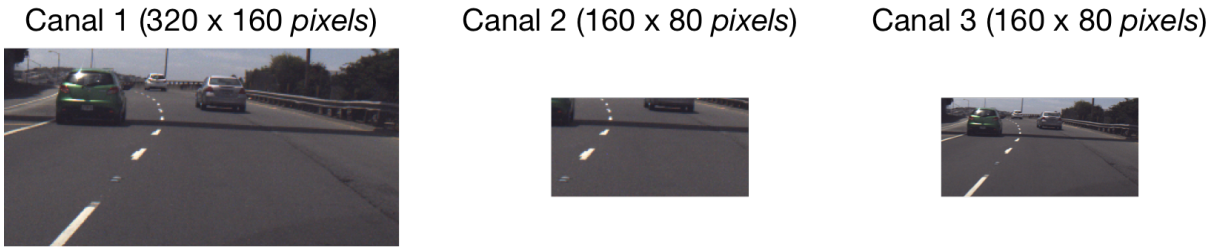


Figura 5.11: Canais de entrada do modelo 4.

Modelo 5 (treinada com `BASE_5`)

O 5º modelo faz uso da CNN de 2 canais, e foi baseado na ideia de CNN de múltiplas resoluções proposta por Karpathy et al. [1], que introduz dois canais de entrada: fóvea e contexto. Como já explicado na Seção 3.2.2, o canal de fóvea recebe a região central da imagem de entrada em resolução original, apresentando dimensões 160x80 na base de dados utilizada. O canal de contexto recebe a imagem original subamostrada em 50%, ou seja, também possuindo resolução 160x80 no conjunto de dados utilizado. Com isso, a dimensionalidade de entrada da rede é reduzida pela metade, algo que pode apresentar uma

melhor performance em termos de tempo de treinamento. Portanto, o 5º modelo é proposto para que seja observado o impacto da performance que redes neurais convolucionais multicanais causam ao receber uma entrada com perda de informações.

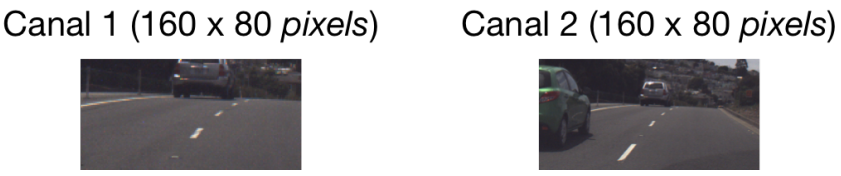


Figura 5.12: Canais de entrada do modelo 5.

Capítulo 6

Resultados Experimentais

Este capítulo apresenta os resultados obtidos a partir das bases de dados e das redes neurais convolucionais apresentadas na metodologia proposta.

6.1 Descrição da Arquitetura Computacional

A seguir são apresentados os principais detalhes da arquitetura computacional utilizada na realização dos testes, bem como, para a validação da metodologia proposta.

6.1.1 *Hardware*

As especificações principais do *hardware* utilizado para treinamento e teste dos modelos propostos foram:

- GPU: NVIDIA Geforce GTX 1070 8GiB;
- CPU: Intel Core i5-6400 2.70GHz;
- Memória RAM: 2x 8GiB DDR4;
- Armazenamento: 1TiB SATA 3 HHD.

6.1.2 *Software*

As ferramentas de *software* utilizadas para execução dos testes propostos no trabalho foram:

- Sistema Operacional: Linux Ubuntu 14.04 LTS 64-bit;
- Linguagem: *Python* 2.7.6;

- *Framework* de aprendizado de máquina: *Keras* 1.1.1, fazendo *back-end* sobre *TensorFlow* 0.11.0rc2.

Além disso foram utilizados: a biblioteca *NumPy* para realização de operações vetoriais, a biblioteca *h5py* para leitura e escrita de arquivos no formato *HDF5*, o *toolkit OpenCV* para processamento de imagens, a biblioteca *Matplotlib* e a ferramenta *MATLAB* para geração de gráficos.

6.2 Protocolo de testes

Os experimentos realizados correspondem aos testes dos cinco modelos apresentados no capítulo anterior. Cada modelo foi treinado usando 200, 350 e 500 épocas, sendo que cada época consiste no processamento de 10.000 exemplos. Cada treinamento com N épocas foi repetido 3 vezes, sendo que em cada um a base de dados foi particionada em subconjuntos de treinamento/validação/teste de forma aleatória, seguindo a proporção 7/2/2.

Dado que as redes neurais convolucionais propostas produzem predições do ângulo de direção do veículo dada uma imagem de entrada, é crucial que a acurácia do modelo seja avaliada. Os valores aqui apresentados correspondem ao erro de cada modelo treinado sob cada conjunto de testes. Como existem dois vídeos de teste na partição de dados realizada, foram obtidas duas medidas para cada treinamento. O erro é calculado entre a predição do ângulo de direção do veículo e o ângulo real do carro capturado por um sensor interno.

Uma medida de erro comumente utilizada para predição de valores numéricos é a raiz do erro médio quadrático (*RMSE*), que é descrita por:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - y_i')^2} \tag{6.1}$$

Tal que n corresponde ao número de predições, y_i corresponde ao valor verdadeiro do exemplo i, e y_i' corresponde ao valor estimado do exemplo i. Portanto, quanto menor o *RMSE* computado para um conjunto de testes, maior é a capacidade do modelo em generalizar para novos exemplos de entrada.

Para que seja possível a comparação entre modelos treinados sob diferentes combinações de bases de dados, foi optado pela normalização da raiz do erro médio quadrático (*NRMSE*) [45], que é calculada por:

$$NRMSE = \frac{RMSE}{y_{max} - y_{min}} \tag{6.2}$$

Tal que y_{min} e y_{max} correspondem ao menor e ao maior ângulo observados no conjunto de testes dado como entrada, respectivamente.

6.3 Resultados obtidos

Os resultados obtidos são apresentados de acordo com cada modelo proposto na metodologia, seguindo a mesma enumeração. Cada subseção é composta por dois elementos:

- Tabela que apresenta os erros (*NRMSE*) de teste obtidos em cada sessão de treinamento do modelo atual. A última linha de cada tabela contém a média dos erros normalizados, com o menor valor obtido apresentado em negrito.
- Gráfico que exibe o ângulo estimado (em vermelho) e o ângulo real (em azul) no decorrer de um vídeo de teste (i.e. não utilizado durante a etapa de treinamento). Esse resultado é obtido através de uma das nove sessões de treinamento do modelo atual. Além disso, vale ressaltar que cada um dos cinco gráficos apresentados nesta seção correspondem a vídeos distintos.

6.3.1 Experimento 1: testes do modelo 1

Tabela 6.1: Resultados de teste dos treinamentos do modelo 1.

	200 épocas	**350 épocas**	**500 épocas**
Treinamento 1 (teste 1)	0.066170	0.051909	0.038168
Treinamento 1 (teste 2)	0.050151	0.078426	0.049583
Treinamento 2 (teste 1)	0.066166	0.038214	0.038608
Treinamento 2 (teste 2)	0.050529	0.050354	0.051880
Treinamento 3 (teste 1)	0.038772	0.034986	0.058295
Treinamento 3 (teste 2)	0.077972	0.029392	0.052042
Média	0.058293	**0.047213**	0.048096

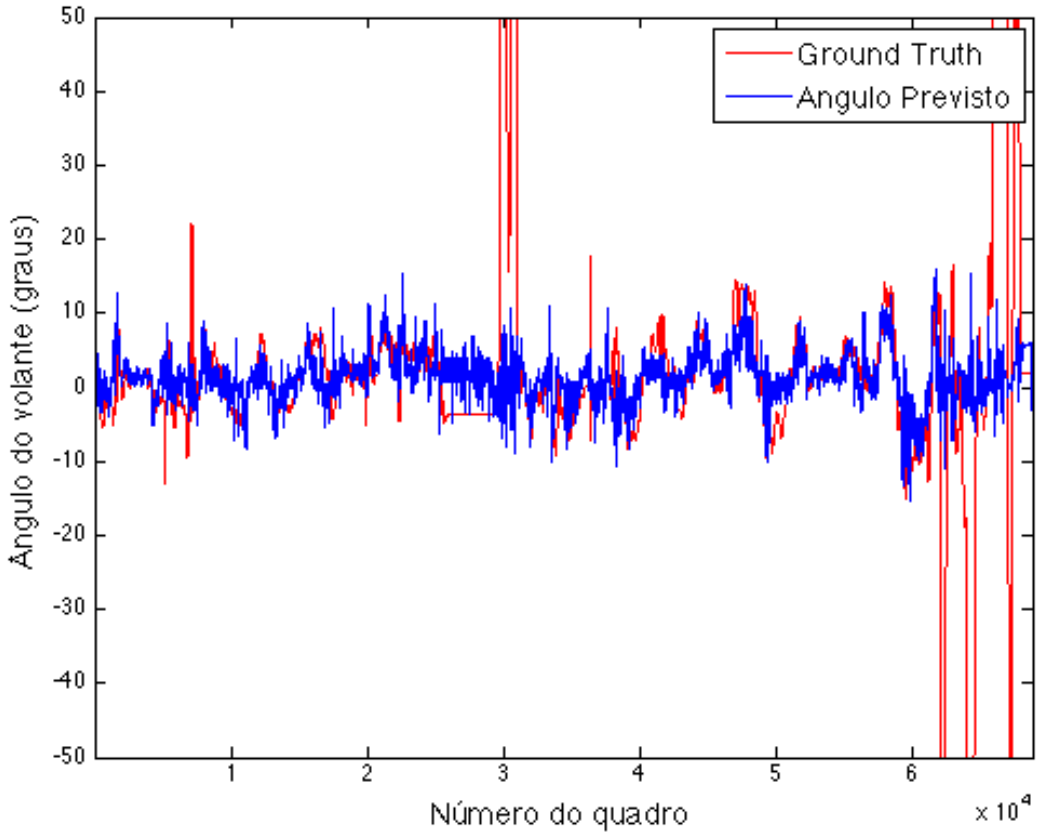


Figura 6.1: Comparação do ângulo de direção estimado com o real em um vídeo de teste, obtido através do 1º modelo treinado por 200 épocas.

6.3.2 Experimento 2: testes do modelo 2

Tabela 6.2: Resultados de teste dos treinamentos do modelo 2.

	200 épocas	**350 épocas**	**500 épocas**
Treinamento 1 (teste 1)	0.050875	0.066027	0.066246
Treinamento 1 (teste 2)	0.038034	0.033105	0.029442
Treinamento 2 (teste 1)	0.033274	0.033108	0.066413
Treinamento 2 (teste 2)	0.052033	0.029413	0.034847
Treinamento 3 (teste 1)	0.038083	0.038708	0.038077
Treinamento 3 (teste 2)	0.033134	0.078019	0.052476
Média	**0.040905**	0.046396	0.047916

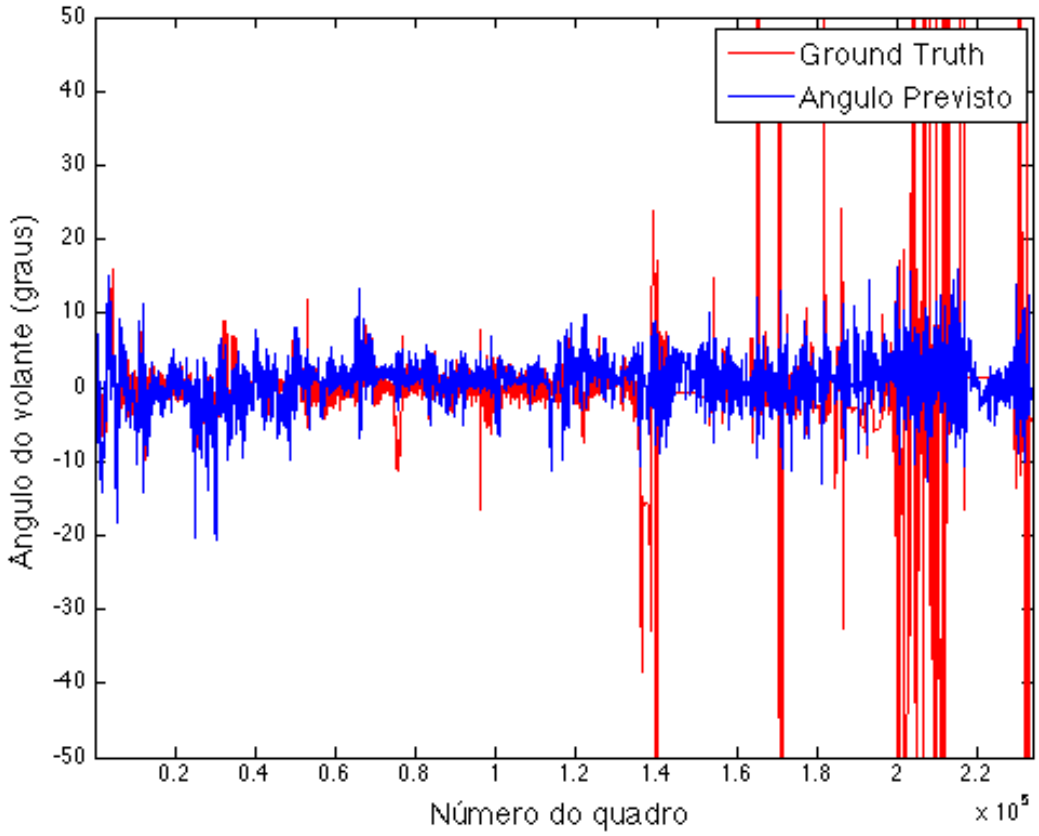


Figura 6.2: Comparação do ângulo de direção estimado com o real em um vídeo de teste, obtido através do 2º modelo treinado por 200 épocas.

6.3.3 Experimento 3: testes do modelo 3

Tabela 6.3: Resultados de teste dos treinamentos do modelo 3.

	200 épocas	**350 épocas**	**500 épocas**
Treinamento 1 (teste 1)	0.058292	0.038018	0.050910
Treinamento 1 (teste 2)	0.038616	0.029392	0.066385
Treinamento 2 (teste 1)	0.034953	0.034886	0.050832
Treinamento 2 (teste 2)	0.029407	0.038717	0.037845
Treinamento 3 (teste 1)	0.052038	0.034944	0.050760
Treinamento 3 (teste 2)	0.077906	0.052106	0.029307
Média	0.048535	**0.038010**	0.047673

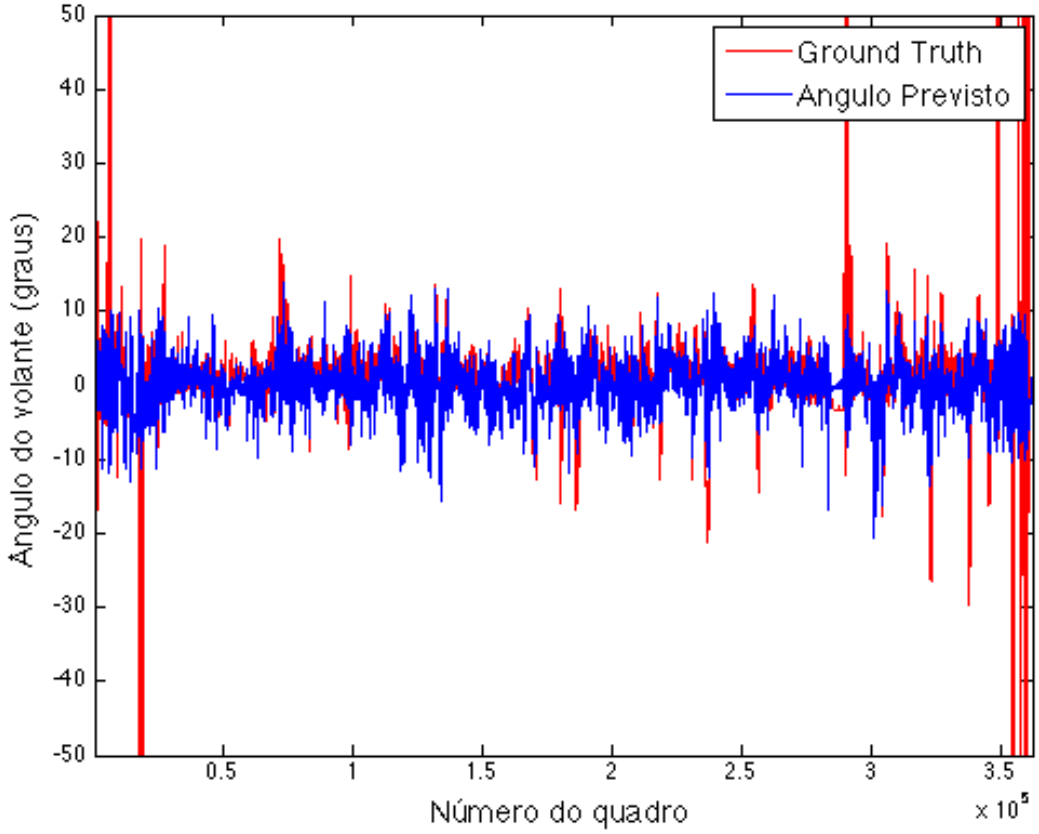


Figura 6.3: Comparação do ângulo de direção estimado com o real em um vídeo de teste, obtido através do 3º modelo treinado por 200 épocas.

6.3.4 Experimento 4: testes do modelo 4

Tabela 6.4: Resultados de teste dos treinamentos do modelo 4.

	200 épocas	**350 épocas**	**500 épocas**
Treinamento 1 (teste 1)	0.037991	0.058209	0.033176
Treinamento 1 (teste 2)	0.052622	0.038755	0.038769
Treinamento 2 (teste 1)	0.034994	0.038655	0.066228
Treinamento 2 (teste 2)	0.058423	0.078159	0.078114
Treinamento 3 (teste 1)	0.050823	0.029371	0.038675
Treinamento 3 (teste 2)	0.050227	0.038761	0.052109
Média	0.047513	**0.046985**	0.051178

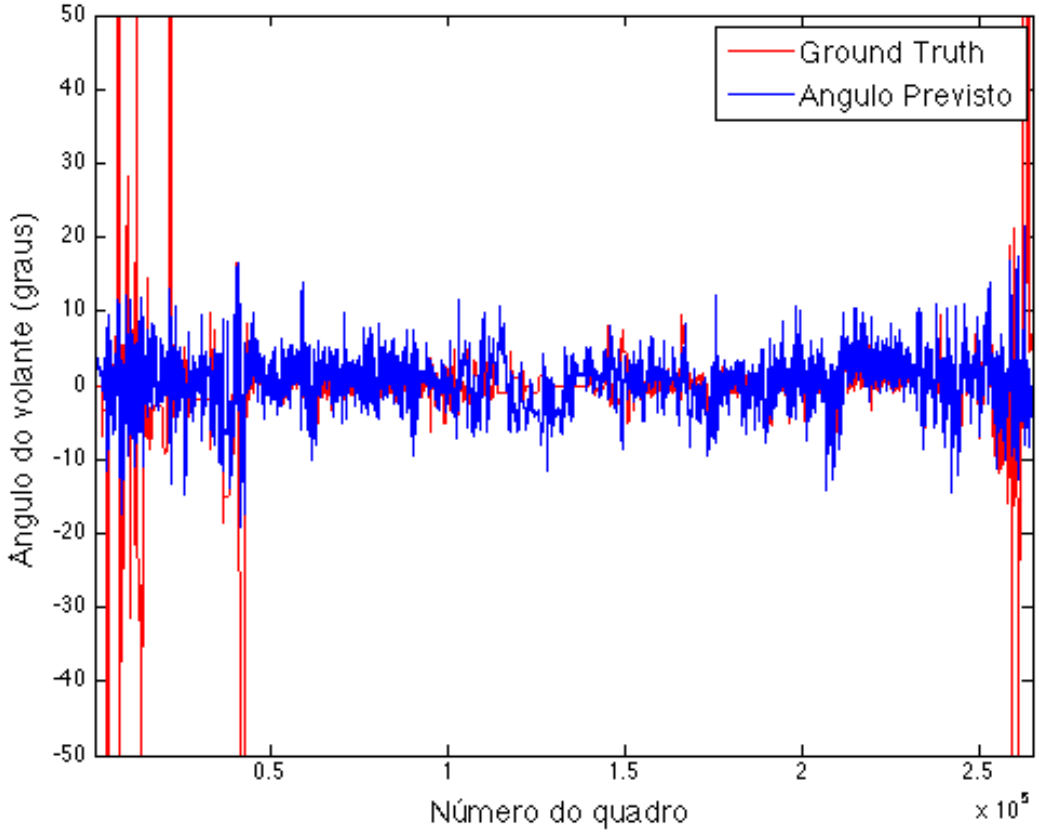


Figura 6.4: Comparação do ângulo de direção estimado com o real em um vídeo de teste, obtido através do 4º modelo treinado por 200 épocas.

6.3.5 Experimento 5: testes do modelo 5

Tabela 6.5: Resultados de teste dos treinamentos do modelo 5.

	200 épocas	**350 épocas**	**500 épocas**
Treinamento 1 (teste 1)	0.033036	0.065899	0.050869
Treinamento 1 (teste 2)	0.058155	0.052457	0.077798
Treinamento 2 (teste 1)	0.050896	0.050848	0.029353
Treinamento 2 (teste 2)	0.051036	0.029339	0.078314
Treinamento 3 (teste 1)	0.050884	0.050263	0.050869
Treinamento 3 (teste 2)	0.049837	0.058436	0.029417
Média	**0.048974**	0.051207	0.05277

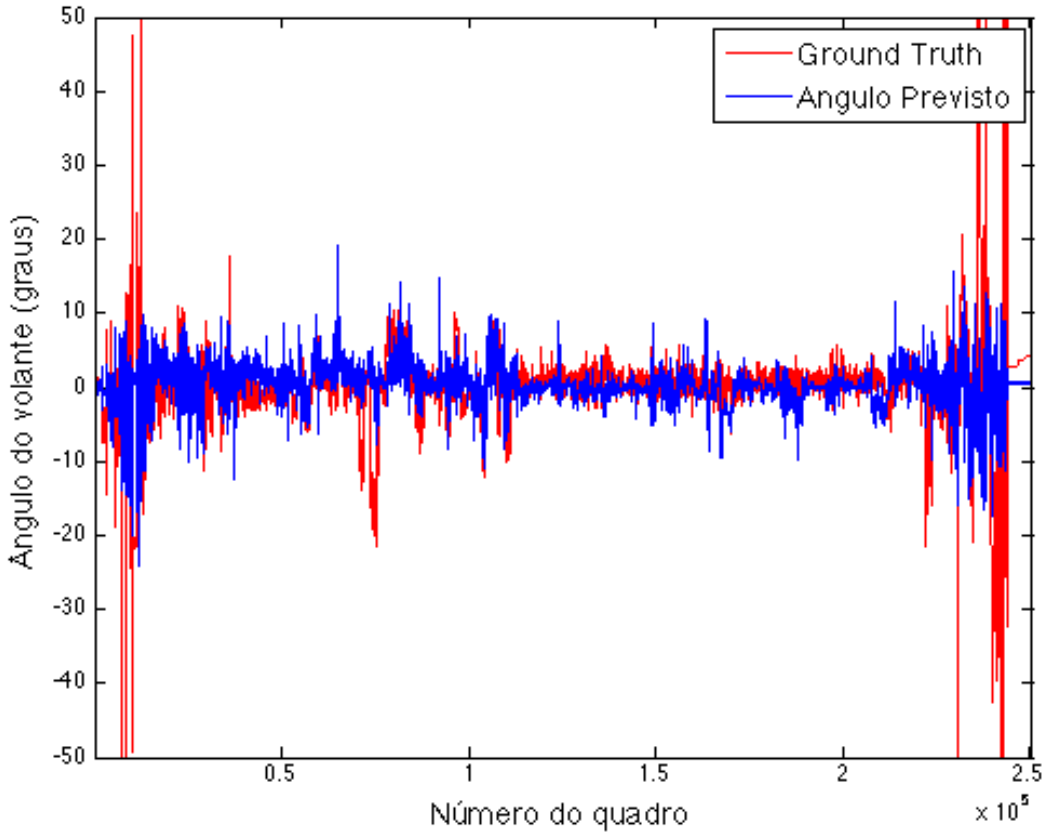


Figura 6.5: Comparação do ângulo de direção estimado com o real em um vídeo de teste, obtido através do 5º modelo treinado por 200 épocas.

6.4 Análise dos resultados

A partir dos resultados obtidos em cada modelo, presentes nas Tabelas 6.1, 6.2, 6.3, 6.4, 6.5, observa-se que as medidas de *NRMSE* são similares entre si, tal que todas pertencem ao intervalo $[0.029307, 0.078426]$. Portanto, a diferença em robustez entre os modelos propostos não é alta.

O gráfico presente na Figura 6.6 apresenta a média dos erros obtidos. Nele, é possível observar que o modelo 3 treinado por 350 épocas apresentou um *NRMSE* médio de **0,038010**, um valor 7,07% menor que segundo melhor modelo (i.e. modelo 2 treinado por 200 épocas).

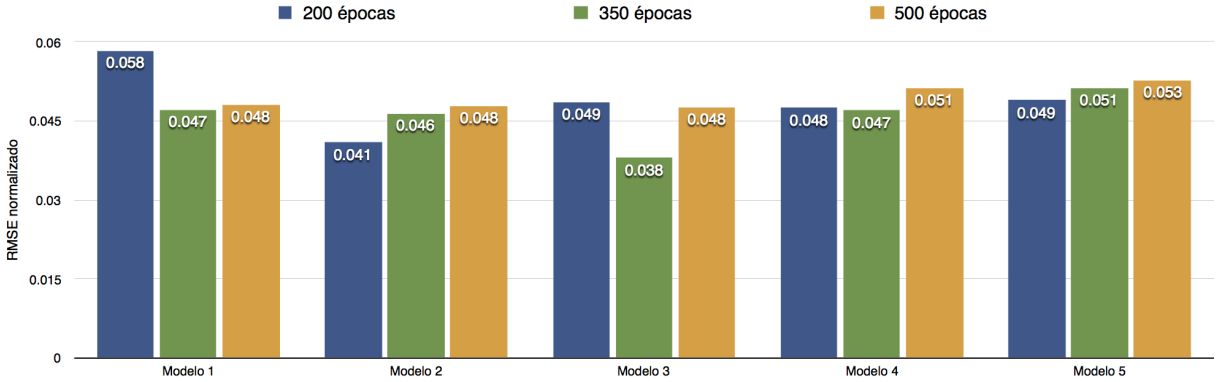


Figura 6.6: Erros médios de teste dos modelos treinados.

A partir da Figura 6.6, nota-se que, em média, os treinamentos por 500 épocas não trouxeram melhoria nos resultados. Portanto, é esperado que o modelo comece a sofrer de sobre-ajuste e perca a capacidade de generalização para exemplos nunca vistos em fase de treinamento. Como exemplo, note o gráfico na Figura 6.7, que mostra o erro de treinamento do modelo 5 treinado por 500 épocas. É perceptível que o decrescimento do erro de treinamento é significantemente reduzido após 200 épocas.

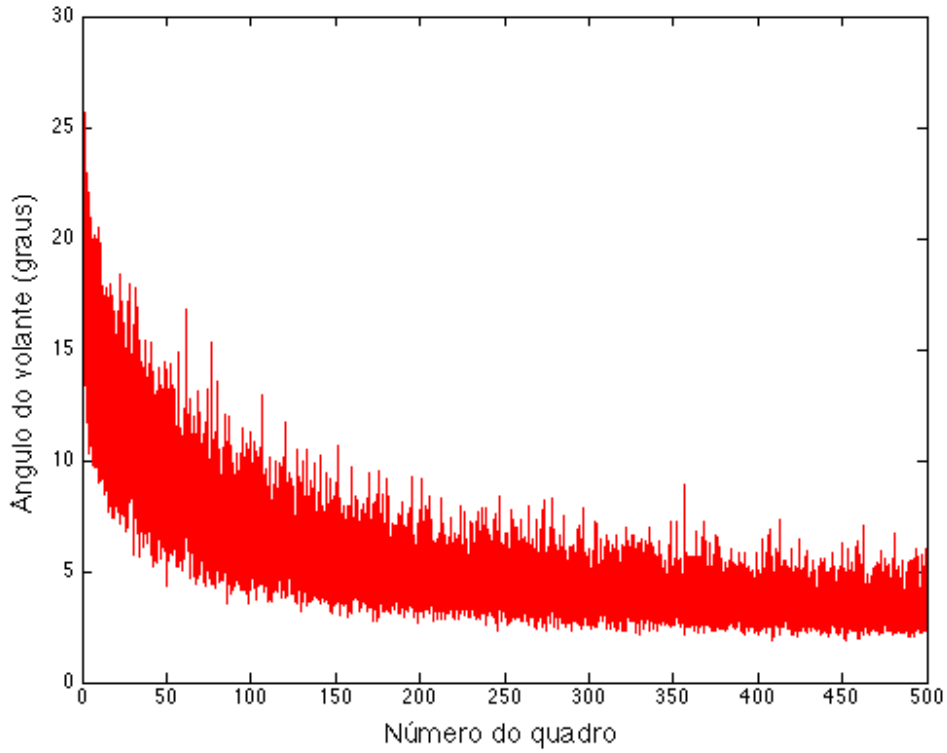


Figura 6.7: Variações do erro de treinamento do modelo 5 treinado por 500 épocas.

A partir dos gráficos de teste obtidos, observa-se que os modelos tendem a estimar ângulos com o mesmo sentido do ângulo real, e são capazes de acompanhar pequenas variações de direção, como rotações de -15 a 15 graus. No entanto, nota-se que os modelos não são capazes de prever grandes variações na direção do veículo. Isso pode ser justificado devido à baixa ocorrência de curvas acentuadas na base de treinamento, impossibilitando que a rede neural convolucional possa aprender e generalizar para novos exemplos. Portanto, é esperado que o erro de teste seja maior em vídeos que incluem um grande número de curvas acentuadas.

Através dos gráficos de teste, também nota-se que o erro é mais expressivo no começo e fim dos testes. Isso se deve ao fato de que na base de dados utilizada, as extremidades dos vídeos correspondem ao momento de entrada e saída do veículo dentro de uma garagem, um evento que implica em uma grande variação no ângulo de direção.

Por fim, é possível analisar qualitativamente os modelos propostos por meio da visualização dos mapas de ativação produzidos por uma camada de convolução da CNN. A partir dessa ideia, a Figura 6.9 mostra os 16 mapas de ativação produzidos pela primeira camada CONV do modelo 1 treinado por 200 épocas, dado o quadro da Figura 6.8 como entrada.

Figura 6.8: O quadro correspondente aos mapas de ativação apresentados na Figura 6.9.

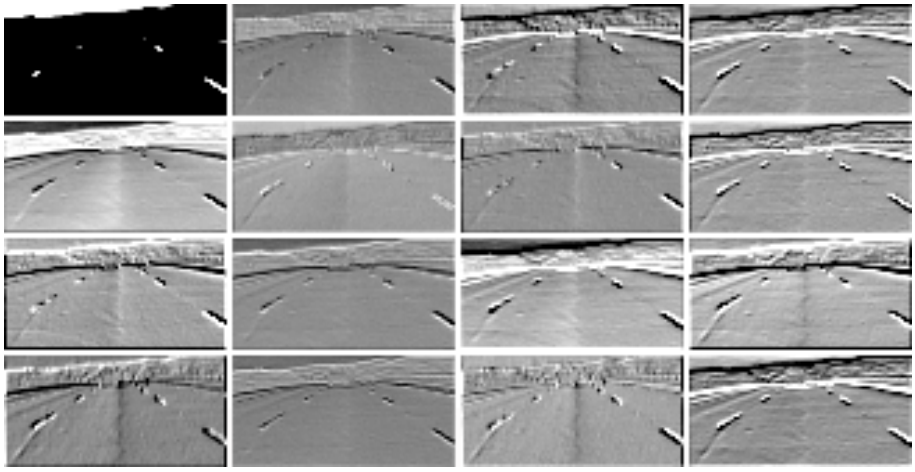


Figura 6.9: Mapas de ativação produzidos pela 1ª camada de convolução, obtidos através do 1º modelo treinado por 200 épocas.

A partir dos 16 mapas de ativação apresentados, observa-se que todos os filtros da primeira camada de convolução destacaram as faixas de marcação da pista. Em outras palavras, a rede neural convolucional foi capaz de detectar a presença de tais elementos em uma cena sem ter sido previamente programada para realizar tal tarefa. Encontrando as faixas de marcação da pista, é possível que a rede possa ter aprendido que o veículo deve se manter em tal área, sendo então capaz de realizar as pequenas variações de direção observadas nos gráficos de teste.

Capítulo 7

Conclusões

Este trabalho propõe um método baseado em redes neurais convolucionais multicanais para realizar a predição do ângulo de direção de um veículo utilizando exclusivamente imagens (espaço *RGB*) como entrada. Com os resultados apresentados foi possível observar que, em média, o modelo 3 proposto, treinado com 350 épocas obteve um erro menor comparado com os demais, incluindo a arquitetura de único canal.

No entanto, é importante ressaltar que os modelos treinados neste trabalho obtiveram performance de teste semelhantes entre si. Devido ao custo computacional elevado em treinar modelos com maior dimensionalidade na entrada, pode ser justificável o uso de um modelo mais simples. Um exemplo disso é o modelo 5, que possui somente os canais de fóvea e contexto, apresentando uma dimensionalidade reduzida pela metade comparada com a arquitetura de único canal. Esta abordagem torna viável sua utilização em aplicações reais para veículos autônomos.

A contribuição principal desse trabalho corresponde a um método baseado em redes neurais convolucionais multicanais capazes de estimar o ângulo de direção de um veículo. Além disso, os testes realizados demonstram que o modelo de CNN de dois canais que recebe a imagem original subamostrada em 50% e a região central é capaz de apresentar resultados semelhantes aos demais modelos. Portanto, outra contribuição relevante consiste na proposição de um modelo capaz de manter a robustez recebendo uma entrada com dimensionalidade reduzida. A partir dessas observações, consideramos que o objetivo proposto foi cumprido.

7.1 Trabalhos futuros

As arquiteturas propostas na metodologia deste trabalho recebem como entrada em seus canais as informações sobre um único quadro capturado em um determinado instante de tempo. Como consequência, as redes neurais convolucionais não são capazes de usufruir

da dimensão temporal do vídeo. Tal informação poderia trazer melhorias para os resultados de performance da rede. Portanto, como trabalho futuro, é importante que sejam desenvolvidos experimentos com as técnicas de fusão de temporalidade em CNNs propostas por Karpathy et al. [1]. Além disso, pode ser válida a experimentação com redes neurais recorrentes (RNN), para que o modelo use as informações de entradas recentes para estimar o ângulo atual de direção do veículo.

Uma outra proposta de trabalho futuro é o uso de uma métrica diferente para a avaliação dos modelos treinados. A estratégia utilizada em [9] consiste na utilização de um simulador que funciona sob um conjunto de testes e é capaz de indicar se a direção atualmente estimada implicaria na ação de retomada do controle do carro por parte do motorista. Com isso, pode ser experimentado como métrica o número de intervenções humanas que seriam necessárias durante o percurso realizado no vídeo de entrada. Esse tipo de abordagem seria mais contextualizada com o domínio de aplicação e poderia trazer importantes informações para análise.

Por fim, pontos de interesse no espaço-tempo (*STIP*) [46] são definidos como regiões em cada quadro de um vídeo que são relevantes para a sua interpretação de eventos espaço-temporais. Portanto, um trabalho futuro consiste na utilização de um canal de fóvea dinâmico, ao invés de sempre extrair somente a região central da imagem, permitindo-se obter melhorias não só na robustez do processo de estimação da direção do veículo, como também possibilitar a utilização de modelos computacionalmente eficientes.

Referências

[1] Karpathy, Andrej, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar e Li Fei-Fei: *Large-scale video classification with convolutional neural networks*. Em *CVPR*, 2014. ix, 30, 31, 48, 62

[2] Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke e Andrew Rabinovich: *Going deeper with convolutions*. CoRR, abs/1409.4842, 2014. `http://arxiv.org/abs/1409.4842`. ix, 34

[3] Organisation Internationale des Constructeurs d'Automobiles: *2016 cars production statistics*. `http://www.oica.net/category/production-statistics/`. Acessado: 2017-06-26. 1

[4] Ros, German, Sebastian Ramos, Manuel Granados, Amir Bakhtiary, David Vazquez e Antonio M. Lopez: *Vision-based offline-online perception paradigm for autonomous driving*. 2015 IEEE Winter Conference on Applications of Computer Vision (WACV), 00:231–238, 2015. 1, 32

[5] Thorpe, Chuck, Martial Hebert, Takeo Kanade e Steven Shafer: *Vision and navigation for the carnegie-mellon navlab*. 10(3):362 – 373, May 1988. 1

[6] Thrun, Sebastian, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian e Pamela Mahoney: *Stanley: The robot that won the darpa grand challenge*. Journal of Field Robotics, 23(9):661–692, 2006, ISSN 1556-4967. `http://dx.doi.org/10.1002/rob.20147`. 2

[7] Wolcott, R. W. e R. M. Eustice: *Visual localization within lidar maps for automated urban driving*. Em *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, páginas 176–183, Sept 2014. 2

[8] NVIDIA: *Self-Driving Vehicles Development Platform | NVIDIA DRIVE PX*. `http://www.nvidia.com/object/drive-px.html`. Acessado: 2017-06-27. 2

[9] Bojarski, Mariusz, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai

Zhang, Xin Zhang, Jake Zhao e Karol Zieba: *End to end learning for self-driving cars*. CoRR, abs/1604.07316, 2016. http://arxiv.org/abs/1604.07316. 2, 36, 62

[10] *Road traffic injuries*. Relatório Técnico, World Health Organization, 2016. http://www.who.int/gho/publications/world_health_statistics/2016/whs2016_AnnexA_RoadTraffic.pdf?ua=1&ua=1, acesso em 2017-04-14. 2

[11] Singh, Santokh: *Critical reasons for crashes investigated in the national motor vehicle crash causation survey*. Relatório Técnico, National Center for Statistics and Analysis, Washington, DC: National Highway Traffic Safety Administration., fev 2015. https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812115, acesso em 2017-04-14. 2

[12] Brown, Austin, Brittany Repac e Jeff Gonder: *Autonomous vehicles have a wide range of possible energy impacts*. Relatório Técnico, National Renewable Energy Laboratory (U.S.), Golden, Colorado, Jul 2013. http:/digital.library.unt.edu/ark:/67531/metadc838531/. 3

[13] Mitchell, Thomas M.: *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1ª edição, 1997, ISBN 0070428077, 9780070428072. 5

[14] Goodfellow, Ian, Yoshua Bengio e Aaron Courville: *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 6, 7, 8, 10, 13, 24

[15] Krizhevsky, Alex, Ilya Sutskever e Geoffrey E Hinton: *Imagenet classification with deep convolutional neural networks*. Em Pereira, F., C. J. C. Burges, L. Bottou e K. Q. Weinberger (editores): *Advances in Neural Information Processing Systems 25*, páginas 1097–1105. Curran Associates, Inc., 2012. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf. 6, 24, 28, 36

[16] Socher, Richard, Andrej Karpathy, Quoc V. Le, Christopher D. Manning e Andrew Y. Ng: *Grounded compositional semantics for finding and describing images with sentences*. TACL, 2:207–218, 2014. 6

[17] Wu, Yonghui, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes e Jeffrey Dean: *Google's neural machine translation system: Bridging the gap between human and machine translation*. CoRR, abs/1609.08144, 2016. 6

[18] Malhotra, Pankaj, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal e Gautam Shroff: *Lstm-based encoder-decoder for multi-sensor anomaly detection*. CoRR, abs/1607.00148, 2016. 6

[19] Radford, Alec, Luke Metz e Soumith Chintala: *Unsupervised representation learning with deep convolutional generative adversarial networks*. CoRR, abs/1511.06434, 2015. 6

[20] Lloyd, S.: *Least squares quantization in pcm*. IEEE Trans. Inf. Theor., 28(2):129–137, setembro 2006, ISSN 0018-9448. http://dx.doi.org/10.1109/TIT.1982.1056489. 7

[21] Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot *et al.*: *Mastering the game of go with deep neural networks and tree search*. Nature, 529(7587):484–489, 2016. 8

[22] Russell, Stuart J. e Peter Norvig: *Artificial Intelligence: A Modern Approach*. Pearson Education, 2ª edição, 2003, ISBN 0137903952. 15, 16, 21, 22

[23] Squire, L.R.: *Fundamental Neuroscience*. ClinicalKey 2012. Elsevier/Academic Press, 2013, ISBN 9780123858702. https://books.google.com.br/books?id=AEmEn-_hD9IC. 16

[24] McCulloch, Warren S. e Walter Pitts: *A logical calculus of the ideas immanent in nervous activity*. The bulletin of mathematical biophysics, 5(4):115–133, 1943, ISSN 1522-9602. http://dx.doi.org/10.1007/BF02478259. 17

[25] Fausett, Laurene (editor): *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994, ISBN 0-13-334186-0. 19

[26] Schaul, Tom, Ioannis Antonoglou e David Silver: *Unit tests for stochastic optimization*. CoRR, abs/1312.6055, 2013. 22

[27] Norman P. Jouppi, Cliff Young, Nishant Patil David Patterson Gaurav Agrawal Raminder Bajwa Sarah Bates Suresh Bhatia Nan Boden Al Borchers Rick Boyle Pierre luc Cantin Clifford Chao Chris Clark Jeremy Coriell Mike Daley Matt Dau Jeffrey Dean Ben Gelb Tara Vazir Ghaemmaghami Rajendra Gottipati William Gulland Robert Hagmann C. Richard Ho Doug Hogberg John Hu Robert Hundt Dan Hurt Julian Ibarz Aaron Jaffey Alek Jaworski Alexander Kaplan Harshit Khaitan Andy Koch Naveen Kumar Steve Lacy James Laudon James Law Diemthu Le Chris Leary Zhuyuan Liu Kyle Lucke Alan Lundin Gordon MacKean Adriana Maggiore Maire Mahony Kieran Miller Rahul Nagarajan Ravi Narayanaswami Ray Ni Kathy Nix Thomas Norrie Mark Omernick Narayana Penukonda Andy Phelps Jonathan Ross Matt Ross Amir Salek Emad Samadiani Chris Severn Gregory Sizikov Matthew Snelham Jed Souter Dan Steinberg Andy Swing Mercedes Tan Gregory Thorson Bo Tian Horia Toma Erick Tuttle Vijay Vasudevan Richard Walter Walter Wang Eric Wilcox e Doe Hyun Yoon: *In-datacenter performance analysis of a tensor processing unit*. ISCA, 2017. 24

[28] LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard e L. D. Jackel: *Backpropagation applied to handwritten zip code recognition*. Neural

Comput., 1(4):541–551, dezembro 1989, ISSN 0899-7667. http://dx.doi.org/10.1162/neco.1989.1.4.541. 24

[29] Karpathy, Andrej: *Connecting Images and Natural Language*. Tese de Doutoramento, Stanford University, 2016. 29

[30] *Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles*. Relatório Técnico, SAE International, 2016. http://standards.sae.org/j3016_201609/, acesso em 2017-04-30. 32

[31] Gurghian, Alexandru, Tejaswi Koduri, Smita V. Bailur, Kyle J. Carey e Vidya N. Murali: *Deeplanes: End-to-end lane position estimation using deep neural networks*. Em *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2016, Las Vegas, NV, USA, June 26 - July 1, 2016*, 2016. http://dx.doi.org/10.1109/CVPRW.2016.12. 33

[32] Angelova, Anelia, Alex Krizhevsky, Vincent Vanhoucke, Abhijit Ogale e Dave Ferguson: *Real-time pedestrian detection with deep network cascades*. Em *Proceedings of BMVC 2015*, 2015. 33

[33] Li, Bo, Tianlei Zhang e Tian Xia: *Vehicle detection from 3d lidar using fully convolutional network*. CoRR, abs/1608.07916, 2016. http://arxiv.org/abs/1608.07916. 33

[34] Kurakin, Alexey, Ian J. Goodfellow e Samy Bengio: *Adversarial examples in the physical world*. CoRR, abs/1607.02533, 2016. http://arxiv.org/abs/1607.02533. 34

[35] Goodfellow, Ian J., Jonathon Shlens e Christian Szegedy: *Explaining and harnessing adversarial examples*. CoRR, abs/1412.6572, 2014. http://arxiv.org/abs/1412.6572. 34

[36] Pomerleau, Dean A.: *Alvinn: An autonomous land vehicle in a neural network*. Em Touretzky, D. S. (editor): *Advances in Neural Information Processing Systems 1*, páginas 305–313. Morgan-Kaufmann, 1989. http://papers.nips.cc/paper/95-alvinn-an-autonomous-land-vehicle-in-a-neural-network.pdf. 34, 35

[37] Chen, Chenyi, Ari Seff, Alain L. Kornhauser e Jianxiong Xiao: *Deepdriving: Learning affordance for direct perception in autonomous driving*. CoRR, abs/1505.00256, 2015. http://arxiv.org/abs/1505.00256. 36

[38] Oliva, Aude e Antonio Torralba: *Modeling the shape of the scene: A holistic representation of the spatial envelope*. International Journal of Computer Vision, 42(3):145–175, 2001, ISSN 1573-1405. http://dx.doi.org/10.1023/A:1011139631724. 36

[39] Bojarski, Mariusz, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence D. Jackel e Urs Muller: *Explaining how a deep neural network trained with end-to-end learning steers a car*. CoRR, abs/1704.07911, 2017. http://arxiv.org/abs/1704.07911. 36, 37

[40] Zeiler, M. D., G. W. Taylor e R. Fergus: *Adaptive deconvolutional networks for mid and high level feature learning.* Em *2011 International Conference on Computer Vision*, páginas 2018–2025, Nov 2011. 37

[41] Clevert, Djork-Arné, Thomas Unterthiner e Sepp Hochreiter: *Fast and accurate deep network learning by exponential linear units (elus).* CoRR, abs/1511.07289, 2015. `http://arxiv.org/abs/1511.07289`. 40

[42] Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever e Ruslan Salakhutdinov: *Dropout: A simple way to prevent neural networks from overfitting.* J. Mach. Learn. Res., 15(1):1929–1958, janeiro 2014, ISSN 1532-4435. `http://dl.acm.org/citation.cfm?id=2627435.2670313`. 42

[43] Bradski, G.: *Open source computer vision library.* Dr. Dobb's Journal of Software Tools, 2000. 46

[44] Kingma, Diederik P. e Jimmy Ba: *Adam: A method for stochastic optimization.* CoRR, abs/1412.6980, 2014. `http://arxiv.org/abs/1412.6980`. 47

[45] Jagannathan, R. e S. Petrovic: *Dealing with missing values in a clinical case-based reasoning system.* Em *2009 2nd IEEE International Conference on Computer Science and Information Technology*, páginas 120–124, Aug 2009. 51

[46] Laptev, Ivan: *On space-time interest points.* International Journal of Computer Vision, 64(2):107–123, 2005, ISSN 1573-1405. `http://dx.doi.org/10.1007/s11263-005-1838-7`. 62