Universidade Federal do Estado do Rio de Janeiro
Centro de Ciências Exatas e Tecnologia
Escola de Informática Aplicada

# Um Protótipo de Veículo Autônomo a Partir de uma Arquitetura Distribuída Utilizando Arduino e Redes Neurais Convolucionais

Daniel Coelho de Oliveira

**Orientador**
Pedro Nuno de Souza Moura
**Coorientador**
Leonardo Luiz Alencastro Rocha

Rio de Janeiro, RJ – Brasil
agosto de 2022

Catalogação informatizada pelo autor

O48 Oliveira, Daniel Coelho de
Um Protótipo de Veículo Autônomo a Partir de uma Arquitetura Distribuída Utilizando Arduino e Redes Neurais Convolucionais / Daniel Coelho de Oliveira. -- Rio de Janeiro, 2022.
112

Orientador: Pedro Nuno de Souza Moura.
Coorientador: Leonardo Luiz Alencastro Rocha.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Estado do Rio de Janeiro, Graduação em Sistemas de Informação, 2022.

1. veículos autônomos. 2. aprendizado profundo. 3. redes neurais convolucionais. 4. inteligência artificial. 5. arduino. I. Moura, Pedro Nuno de Souza, orient. II. Rocha, Leonardo Luiz Alencastro, coorient. III. Título.

Um Protótipo de Veículo Autônomo a Partir de uma Arquitetura Distribuída Utilizando Arduino e Redes Neurais Convolucionais

Daniel Coelho de Oliveira

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação.

Aprovado por:

_____

PEDRO NUNO DE SOUZA MOURA (UNIRIO)

_____

LEONARDO LUIZ ALENCASTRO ROCHA (UNIRIO)

_____

SIDNEY CUNHA DE LUCENA (UNIRIO)

RIO DE JANEIRO, RJ – BRASIL.

AGOSTO DE 2022

## Agradecimentos

Aos meus pais, que sempre se esforçaram ao máximo para me dar todas as oportunidades possíveis, me dar carinho, conforto, amor e ainda me indicaram essa área que hoje amo.

À minha namorada Fernanda, por ser o amor, amiga e companheira mais incrível que a vida poderia me dar, sempre me apoiando, me ajudando e participando de cada momento de minha trajetória, tornando cada instante mais leve, especial e feliz. "One look at you, and my heart knew you were the one for me". Te amo.

À minha família de sangue e coração, que sempre me acolheram, me guiaram, me ajudaram e me deram amor.

À todos os meus amigos de faculdade, que ajudaram a fazer da faculdade a minha segunda casa. Nunca imaginei que conheceria pessoas tão incríveis na graduação e que tomariam um lugar tão especial em minha vida.

Aos meus amigos da vida, os quais de alguma forma fizeram parte da minha trajetória, desejando sempre o melhor de mim e tiveram uma enorme importância, me proporcionando momentos incríveis.

Ao Prof. Pedro Nuno de Souza Moura, o qual considero um grande professor, sempre buscando ajudar e compreender a situação dos alunos, uma pessoa de grande coração e, mais que tudo, um grande amigo que espero levar para a vida.

Ao Prof. Leonardo Luiz Alencastro Rocha, pelos conselhos e dicas, por ter se animado com o projeto e ter logo se oferecido a emprestar seus equipamentos para realização deste trabalho.

Aos professores e funcionários do Departamento de Informática Aplicada, de maneira geral, os quais, cada um de sua forma, são de grande importância na trajetória dos alunos.

A tudo e todos que participaram da minha vida de alguma forma, me ajudando, me guiando etc. Gratidão por todas as pessoas incríveis que passaram pela minha vida.

## RESUMO

Com a popularização da Inteligência Artificial na última década, através da revolução trazida pelo *Deep Learning*, este tema torna-se cada vez mais relevante tanto para pesquisadores e profissionais de Tecnologia da Informação, quanto para o público em geral. Nesse contexto se encaixam os veículos autônomos, os quais não só naturalmente atiçam a curiosidade alheia como também ao se popularizar poderão trazer diversos benefícios à sociedade.

Desta forma, este trabalho teve como objetivo criar um protótipo de veículo autônomo baseado em modelos de *Deep Learning*, utilizando um sistema distribuído composto por um celular, equipamentos de hardware como arduino e um *notebook*. Diversos experimentos com alguns modelos de redes neurais foram realizados e seus resultados foram analisados.

Por fim, este trabalho também tem como contribuição fornecer explicações inteligíveis e didáticas para um aluno de graduação, abordando diversos conceitos relacionados à inteligência artificial e *deep learning*.



**Palavras-chave**: veículos autônomos, inteligência artificial, aprendizado profundo, redes neurais convolucionais, arduino.

## ABSTRACT

With the popularization of Artificial Intelligence in the last decade, through the revolution brought by Deep Learning, this topic becomes increasingly relevant for researchers and Information Technology professionals, as well as for the general public. Autonomous vehicles fit into this context, which not only brings curiosity for the public but also has the potential to bring great improvements for society.

Thus, this work aimed to create an autonomous vehicle based on a distributed system including a smartphone, a notebook and hardware equipment related to Arduino. Several experiments were carried out and their results were analyzed.

Finally, this work also contributes to providing explanations, addressing several concepts related to artificial intelligence and deep learning.



**Keywords**: autonomous vehicles, artificial intelligence, deep learning, convolutional neural networks, arduino.

## Índice

## Índice de Quadros

## Índice de Figuras

## Lista de Siglas e Abreviaturas

**AI** - *Artificial Intelligence*

**ML** - *Machine Learning*

**DL** - *Deep Learning*

**CNN** - *Convolutional Neural Network*

**RGB** - sistema de cores *Red Green Blue*

**YUV** - Sistema de codificação de cores *luma* (Y), *red projection* (U) e *blue projection* (V)

**LSTM** - *Long Short-Term Memory Systems*

**ReLU** - Função de atividade denominada *Rectified Linear Unit*

# 1 Introdução

## 1.1 Motivação

Cada vez mais técnicas de Inteligência Artificial (*Artificial Intelligence* - AI) vêm sendo utilizadas, com o intuito de ajudar o ser humano em diversas áreas de trabalho, ou mesmo em seu dia a dia. Desde cedo, junto com o surgimento do advento do computador, começaram a surgir também teorias de onde essa nova tecnologia seria capaz de nos levar ou até que nível seríamos capazes de evoluí-la. Assim, essas teorias acabaram dando vida a diversos filmes como "2001: Uma odisséia no espaço" ou mesmo filmes mais recentes como "Eu robô". Neste último, robôs controlados por uma AI começam a ter vontade própria de modo que se voltam contra os próprios humanos, que os construíram. O filme "Matrix" segue essa mesma linha de raciocínio, em que a AI criada pelo homem se revolta contra o ser humano, por conta dos problemas que ele está causando ao mundo, usando-os como fonte energética.

Dessa forma, é possível perceber um padrão que essas histórias seguem: possuímos essa tendência de acreditar que se fôssemos capazes de criar máquinas tão inteligentes quanto nós, estas perceberiam a incoerência de nos servir, ou mesmo perceberiam os danos que causamos ao planeta que agora é a sua casa, e poderiam então se virar contra nós. É interessante refletir sobre o porquê de tendermos, por vezes, a essa versão de teoria futurista distópica. Será que as bases de funcionamento de uma AI permitiriam que isso se tornasse verdade, ou isso tudo não passa de uma fantasia absurda?

De qualquer forma, querendo ou não, esse tipo de tecnologia cada vez mais faz parte do nosso dia a dia. Hoje a AI é aplicada em áreas como de atendimento ao cliente, medicina, comércio eletrônico, e-mail, na fabricação de carros autônomos e até mesmo na seleção de candidatos, como é descrito em (ALBERT, 2019).

Nesse contexto, carros autônomos têm sido um tópico de grande interesse nos últimos anos, tanto por parte da indústria quanto por parte dos consumidores. Com isso, diversas empresas têm colocado esforços em pesquisa e desenvolvimento de hardware e software necessário não só para a criação, como também para o barateamento e consequente popularização desse tipo de carro.

De acordo com o ONSV (OBSERVATÓRIO Nacional de Segurança Viária), as três principais motivações dos acidentes de trânsito estão relacionadas e podem ser agrupadas em “Fator Humano, Fator Veículo e Fator Via” [1].

Segundo a entidade, 90% dos acidentes ocorrem por falhas humanas – que podem envolver desde a desatenção dos condutores até o desrespeito à legislação. Os exemplos são claros, excesso de velocidade, uso do celular, falta de equipamentos de segurança como o cinto de segurança ou capacete, o uso de bebidas antes de dirigir ou até mesmo dirigir cansado [1].

Isso poderia ser consideravelmente mitigado com o uso de carros completamente autônomos ou mesmo carros que possuíssem algum tipo de assistência de direção.

## 1.2 Objetivos

Dada então a importância do tópico “Direção Autônoma”, o objetivo deste trabalho é a apresentação não só de uma pesquisa, como também de experimentos relativos à criação de um mini-modelo de carro, o qual seja capaz de se locomover de modo autônomo. A ideia é que, com o auxílio de peças de hardware simples como arduino, e a partir do treinamento de um modelo de rede neural, o carrinho seja capaz de navegar sozinho por uma pista com a qual a rede nunca tenha tido contato.

Além disso, o presente trabalho possui como objetivo explicar diversos tópicos relacionados à Inteligência Artificial, de modo que mesmo pessoas leigas no assunto consigam lê-lo e compreendê-lo.

## 1.3 Metodologia

Primeiramente foram analisados diversos artigos relacionados ao tema, para que pudesse ser entendido quais tipos de peças de hardware e quais redes neurais que são normalmente utilizadas nesse tipo de problema. Além disso, através dessas referências,

[1] Disponível em: <https://www.onsv.org.br/90-dos-acidentes-sao-causados-por-falhas-humanas-alerta-observatorio/>. Acesso em 15 de Mar. 2022.

foi possível entender, mesmo que não muito detalhadamente, o passo a passo necessário para a construção do protótipo.

Dessa forma, foi possível comparar os resultados obtidos pelos trabalhos analisados e portanto definir quais seriam os equipamentos mais apropriados para montagem do carrinho autônomo. Também possibilitou analisar quais redes se encaixam melhor para diferentes cenários, como, por exemplo, *path planning,* no qual se deseja fazer com que o carrinho continue seguindo uma determinada pista delineada, reconhecimento e classificação de objetos de sinais de trânsito, definição da aceleração do carrinho em um determinado momento e assim por diante.

Outras informações importantes obtidas a partir das pesquisas realizadas foram de possíveis *datasets* a serem utilizados e ferramentas que ajudam a fornecer dados de entrada para treinamento da rede. Uma ferramenta importante descoberta foi o simulador do site de cursos online *Udacity*[2]. Ele é utilizado por um curso de AI da página citada e consiste em um simulador de corrida de carros que auxilia a conseguir o *dataset* necessário para treinar uma rede neural para realizar o *path planning*. Além disso, foi possível notar que a maioria dos trabalhos pesquisados fazia uso da biblioteca *OpenCV*[3], a qual foi utilizada neste trabalho, para realizar diferentes pré-processamentos.

Possuindo essas informações em mãos, foi possível elencar as que levaram aos melhores resultados e dessa forma testá-las, a fim de realizar uma comparação e chegar à conclusão de quais foram mais efetivas.

As redes foram treinadas então através do *Google Colab* (ferramenta que disponibiliza poder computacional tanto de CPU quanto GPU para realizar processamentos desse tipo). Após o treinamento, as melhores redes eram carregadas em um *script python* e utilizadas para processar imagens que chegavam de um *stream*. Após isso, era enviado um sinal via *bluetooth* para o Arduino, determinando a direção que o carrinho deveria seguir.

Foram testadas diversas redes e suas características e resultados são apresentados no capítulo 5, chegando à conclusão, por fim, de qual foi a melhor configuração.

[2] https://www.udacity.com/
[3] https://opencv.org/

## 1.4 Organização do texto

O presente trabalho está estruturado em capítulos e, além desta introdução, será desenvolvido da seguinte forma:

- Capítulo II: Conceitos Preliminares – apresenta os conceitos necessários para entendimento deste trabalho, a respeito do tema inteligência artificial.
- Capítulo III: Revisão Bibliográfica – apresenta trabalhos que tenham lidado com tema parecido e que serviram como base para este.
- Capítulo IV: Arquitetura adotada e arduino – apresenta o modelo de rede neural que foi utilizado no trabalho, assim como os componentes de *hardware* necessários.
- Capítulo V: Experimentos – relata os experimentos feitos, passando pelas tentativas, explicando o caminho até chegar ao modelo final e as análises realizadas.
- Capítulo VI: Conclusão – Reúne as considerações finais, assinala as contribuições da pesquisa e sugere possibilidades de aprofundamento posterior.

# 2 Conceitos Preliminares

Este capítulo aborda conceitos a respeito da área de AI e mais especificamente a respeito do aprendizado de máquina, com objetivo de preparar o leitor para o entendimento dos capítulos que virão a seguir.

## 2.1 Inteligência Artificial

Primeiramente, é de grande relevância entender que a AI é baseada em sistemas de computadores (*softwares*) que utilizam entradas de dados para de alguma maneira aprender com eles e conseguir realizar uma tarefa de modo automatizado.

Entretanto, um *software* normal já possui esse objetivo de realizar tarefas de maneira automatizada, então qual seria a diferença entre ambos? A grande diferença é que, neste caso, o programador precisa declarar regras explícitas para que o programa realize certas ações de acordo com determinadas circunstâncias. Dessa forma, de modo geral, precisamos definir caso a caso como o *software* deverá se comportar e, sobretudo, o sistema não possui capacidade de aprender a partir dos casos que lhe forem apresentados.

Já no caso da AI, é criado um *software* que automaticamente, a partir de *inputs* que lhe serão fornecidos, irá aprender como deve agir de acordo com determinadas circunstâncias. A arquitetura deste programa de AI é construída de modo intencional, porém o aprendizado para realizar as tarefas que este foi criado para resolver é realizado de maneira autônoma. Este programa é então baseado em funções e operações matemáticas que permitem a otimização de uma determinado custo (ou perda), até o ponto em que ficam ideais para a resolução daquele problema. Essa otimização ocorre durante o processo de treinamento do algoritmo, o qual é um dos processos mais relevantes em seu ciclo de vida.

A área de AI, pode parecer confusa em um primeiro momento, pelo fato de envolver diversas áreas e subáreas. A Figura 1 ilustra as diferentes áreas contidas pela Inteligência Artificial.

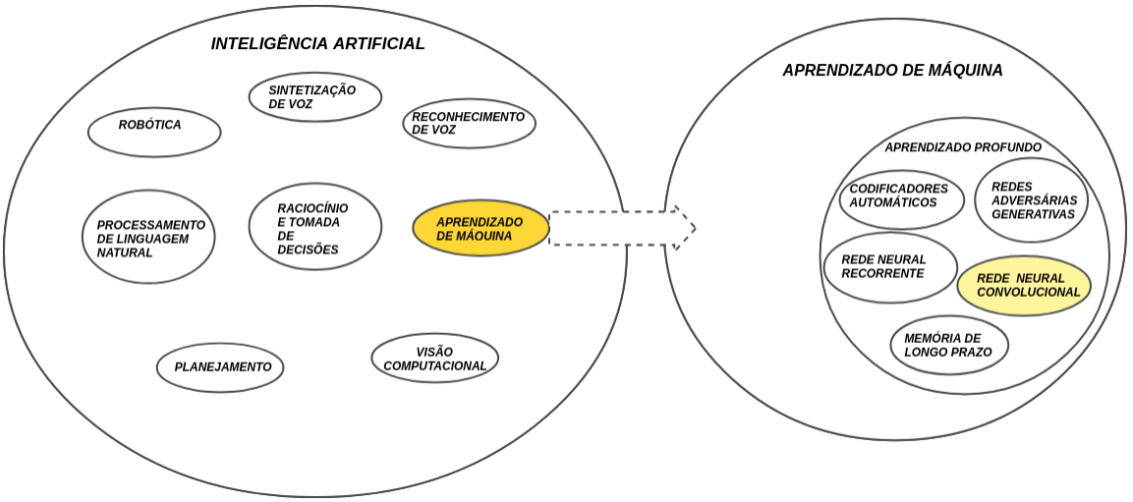



**Figura 1: Visão geral das subáreas da Inteligência Artificial (extraído de OLIVEIRA, 2020).**

Como é possível avaliar, a AI engloba diversas áreas de estudo, como a Robótica, Processamento de Linguagens Naturais, Visão Computacional, Reconhecimento de Voz, Aprendizado de Máquina (*Machine Learning* - ML), etc.

Neste trabalho, como será possível avaliar mais adiante, foi feito uso de técnicas pertencentes a uma subárea contida pelo Aprendizado de Máquina, que corresponde ao Aprendizado Profundo (*Deep Learning* - DL). Dentro desta, assim como dentro da AI, existem diversas especializações, sendo a Rede Neural Convolucional (*Convolutional Neural Network* - CNN), a mais relevante para este trabalho.

## 2.2 Aprendizado de Máquina

Como dito anteriormente, ML é uma subárea da AI composta por diferentes algoritmos de aprendizado os quais possuem características em comuns. A principal é que estes buscam imitar o comportamento de um cérebro humano, utilizando matemática (principalmente estatística) e buscando assim alcançar, em certa medida, a capacidade de inteligência humana.

## 2.3 Técnicas de aprendizado de máquina

As principais técnicas de treinamento/aprendizado de um algoritmo de ML podem ser divididas entre **aprendizado supervisionado**, **aprendizado não supervisionado** e **aprendizado por reforço.**

No caso do **aprendizado supervisionado**, são fornecidos os dados de *input* e, para cada um desses dados, é também fornecido um rótulo *(label)*. Dessa maneira, o algoritmo irá processar cada dado de entrada e chegar a um resultado, o qual será comparado com o rótulo associado fornecido. É possível fazer uma analogia com um aluno que, para estudar para uma prova, faz uso de uma prova antiga de que ele já possui todas as respostas corretas. À medida que o aluno vai então fazendo tal prova, vai também comparando com o resultado correto e, caso seja necessário, vai ajustando o seu conhecimento. Quando ele considera então estar preparado, vai realizar a prova para a qual estava estudando, que possui questões similares àquelas vistas anteriormente, e utiliza os conhecimentos obtidos no período de estudo. Se esse treinamento foi realizado de forma correta e foram resolvidas questões suficientes na preparação, o desempenho obtido na nova prova tende a ser bom e a nota vir alta, caso contrário, o aluno tende a ter um mau desempenho e tirar uma nota baixa. Óbvio que esta simplificação foi utilizada somente para motivos de explicação.

Os algoritmos que utilizam essa abordagem de aprendizado funcionam de maneira bem semelhante, ajustando parâmetros internos do modelo de ML à medida que processam dados de entrada, geram saídas e as comparam com os rótulos, durante o processo de treinamento.

Já durante a etapa de validação, o algoritmo é exposto a dados da mesma distribuição original dos dados de treinamento, mas a cujos rótulos ele não possui acesso, e a sua qualidade é avaliada através da comparação entre a saída gerada (predita) pelo modelo e o valor do rótulo. Normalmente, os dados totais são divididos em dados de treinamento e dados de validação, sendo a proporção normalmente em torno de 70% e 30%, respectivamente. Ademais, existem outras formas de fazer a divisão dos dados para realizar o treinamento e validação.

No caso do **aprendizado não supervisionado**, uma grande quantidade de dados também é requerida, porém não existem rótulos associados a serem aprendidos. Nesse

caso, o algoritmo de ML descobre sozinho padrões entre os dados fornecidos, a partir de características em comum e, dessa maneira, vai ajustando os parâmetros de seu modelo. Algumas técnicas conhecidas, dentro desse grupo, são a compressão de dados, a distribuição probabilística, a clusterização etc.

Por fim, no **aprendizado por reforço**, um agente (algoritmo) aprende a partir da tentativa e erro, ao lidar com uma situação em um determinado ambiente. Nesse caso, existem dados, mas somente do ambiente, e o objetivo desse agente é maximizar a recompensa. Para o caso deste tipo de aprendizado, podemos pensar em um agente que tenta aprender a jogar uma fase do jogo Mario, clicando aleatoriamente nos botões. Nesse caso, quanto mais longe o agente chega na fase (avança o máximo que conseguir para a direita), no menor tempo, maior será a recompensa obtida. Dessa maneira, ele joga milhares de vezes, até entender a forma mais otimizada de vencer a fase no menor tempo.

## 2.4 Regressão e Classificação

O aprendizado dos modelos utilizados neste trabalho foi o **aprendizado supervisionado**. Dentro deste, possuímos a divisão entre problemas de **regressão** e problemas de **classificação**. O primeiro engloba problemas como de predição de um valor numérico contínuo. Já no caso dos problemas de classificação, lidamos com problemas como de classificação de imagens, detecção de objetos, diagnósticos médicos (como, por exemplo, se em uma radiografia do pulmão de um paciente, é detectada uma pneumonia ou não), etc.

A grande diferença entre ambos os problemas é que, no caso de problemas de **regressão**, a saída do modelo é contínua, ou seja, o resultado que é fornecido ao final do processamento pode ser qualquer um, dentro do universo de possibilidades daquele cenário específico. Em geral, essa saída corresponde a um valor numérico contido no conjuntos dos reais. Por exemplo, pode ser gerado um modelo, que a partir de informações de um paciente em fase de crescimento, consiga prever a altura máxima que ele pode chegar até o final de sua formação. Neste caso, o valor resultante será contínuo, pois não será delimitado por uma quantidade específica de classes.

Já os problemas de **classificação** dizem respeito a problemas cuja saída do modelo é discreta, de modo que o objetivo é mapear as variáveis de entrada a um

conjunto de valores discretos. Nesse caso, a saída do modelo se encontra restrita a um número delimitado de classes. Por exemplo, um modelo pode ser treinado, a partir de um *dataset* de imagens de cachorros e gatos, a classificar uma dada imagem fornecida. Nesse caso, os únicos resultados possíveis, para qualquer entrada, seriam gato ou cachorro. A Figura 2 representa um problema de classificação, enquanto a Figura 3 representa um problema de regressão.

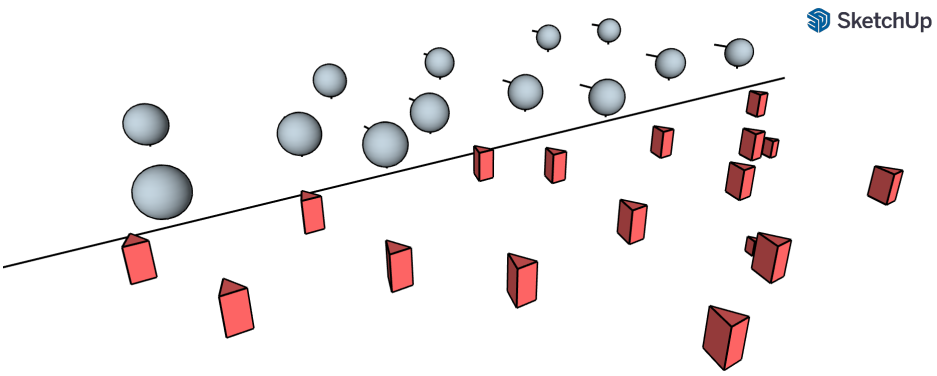


**Figura 2:** Representação de um problema de classificação.

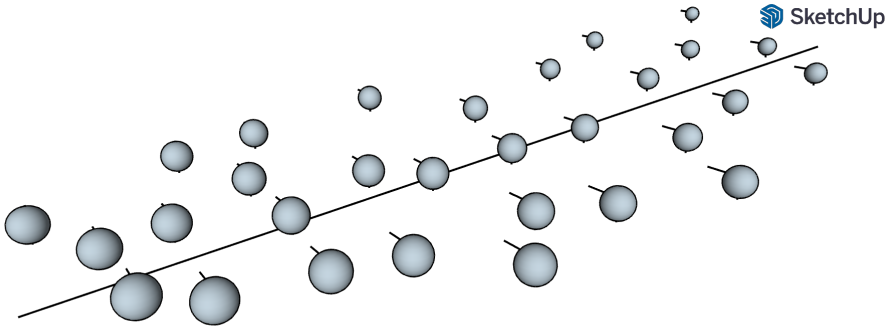


**Figura 3:** Representação de um problema de regressão.

Na Figura 2, como é possível observar, a linha representa um separador linear entre exemplos de dois grupos distintos. Assim sendo, o objetivo de um problema de

classificação é exatamente este, ou seja, definir a separação entre dois ou mais grupos, com objetivo de classificá-los.

Já um problema de regressão possui como objetivo entender um certo tipo de comportamento para tornar-se capaz de predizê-lo nos casos desconhecidos. Dessa forma, a linha na Figura 3 se encontra acompanhando o fluxo das esferas, ao invés de tentar separar diferentes grupos, como ocorre na Figura 2.

## 2.5 Aprendizado Profundo

Como dito anteriormente, DL se encontra dentro da área de aprendizado de máquina e é composto por um grupo de algoritmos que processam dados visando a simular o funcionamento de um cérebro humano. O objetivo é que, assim como o cérebro recebe estímulos externos e gera respostas a eles, os modelos de aprendizagem profunda possam receber *inputs* de dados e gerar respostas, com algum propósito previamente definido. Para isso, esses algoritmos são formados por diversas camadas, as quais realizam processamentos não lineares em cima dos dados de *input* fornecidos.

Dessa maneira, cada camada vai extraindo características de maneira hierárquica em relação às camadas anteriores, de modo que a saída de uma camada anterior se torna a entrada da camada seguinte. Não há um consenso em relação ao número mínimo de camadas necessárias para um modelo ser considerado de aprendizado profundo, porém, em geral, precisa conter pelo menos um número maior que duas.

Um aspecto importante de DL é a representação dos dados, que podem ser diversas. Por exemplo, uma imagem pode ser representada de maneira unidimensional, através de um vetor em que cada posição representa a intensidade de um pixel. Neste caso, abre-se mão da informação contida na posição e vizinhança de cada pixel. Entretanto, no caso de imagens RGB, a imagem poderia ser representada por três vetores, sendo que cada um deles representaria a intensidade de cada pixel da sua respectiva camada. Por exemplo, caso o primeiro vetor representasse a camada *R* (vermelho - *red*), cada uma de suas posições representaria a intensidade de vermelho de um pixel específico. O maior problema desse tipo de abordagem para a representação de imagens, é que características espaciais destas acabam se perdendo. Isso porque os pixels da imagem, nesse caso, são achatados em somente um grande vetor.

Outra representação possível seria fazendo uso de matrizes. Nesse caso a ideia poderia ser a mesma (de representação das intensidades dos pixels), a diferença é que as características espaciais seriam mantidas.

Dessa maneira, o objetivo dos algoritmos e modelos de aprendizagem profunda é extrair características relevantes e muitas vezes escondidas dos dados que estão sendo processados, para poder por fim chegar a conclusões a respeito deles, tanto aprendendo padrões sobre os dados, como classificando-os.

Os algoritmos mais famosos e mais bem sucedidos que dizem respeito ao aprendizado profundo envolvem redes neurais artificiais, de modo que algumas das principais arquiteturas de aprendizagem profunda são redes neurais profundas, redes neurais convolucionais, redes neurais recorrentes etc. A arquitetura que foi utilizada no modelo deste trabalho foi a de *redes neurais convolucionais.*

## 2.6 Redes Neurais Convolucionais

CNN é um tipo de arquitetura de rede neural que possui uma ou mais camadas convolucionais e que é normalmente utilizada para a tarefa de classificação de imagens. Isso porque esse tipo de rede é bastante eficiente no reconhecimento de padrões espaciais o que a torna excelente para tarefas de visão computacional.

Como foi dito anteriormente, duas representações computacionais possíveis para imagens são *arrays* e matrizes, sendo que, para o caso de uma imagem em escalas de cinza, por exemplo, cada posição do *array* ou matriz irá representar o valor de cada pixel da imagem, variando de 0 (preto) até 255 (branco). A grande diferença é que enquanto o *array* "achata" a imagem, a matriz guarda as características espaciais da mesma, facilitando na hora da classificação. No caso de uma imagem colorida (RGB), a representação se daria da mesma forma, tirando o fato de que seriam necessárias 3 camadas, também variando de 0 a 255, para o vermelho, verde e azul.

Portanto, a rede que é capaz de processar imagens utilizando a representação em matrizes, a qual como informado, a mais indicada é a rede neural convolucional. Cada camada convolucional possui filtros (também chamados de *kernels*), os quais vão passando pela imagem (como se a tivessem escaneando), partindo do canto superior esquerdo e finalizando no inferior direito. A cada etapa desse *scan*, é realizada uma operação de multiplicação matricial, cujo resultado vai formando, valor a valor, uma

matriz resultante, a qual servirá de entrada para a próxima camada. A Figura 4 ilustra o funcionamento deste *scan*.

**Figura 4: Representação do processo de passagem de um filtro convolucional por duas imagens diferentes (extraído de KROHN; BEYLEVELD; BLASSENS, 2020).**

Os filtros possuem pesos os quais vão sendo otimizados durante o processo de *backpropagration*, o qual será explicado mais adiante neste trabalho, além de normalmente possuírem o tamanho de 3x3 ou 5x5. Cada camada convolucional irá então possuir diversos filtros, os quais irão aprender diferentes aspectos a respeito da imagem em questão. Normalmente também, as camadas vão progressivamente aprendendo aspectos mais complexos da imagem, de modo que as primeiras camadas aprendem, por exemplo, a reconhecer linhas retas, enquanto camadas mais profundas já conseguem interpretar curvas etc. A Figura 5 representa um exemplo de rede neural convolucional.

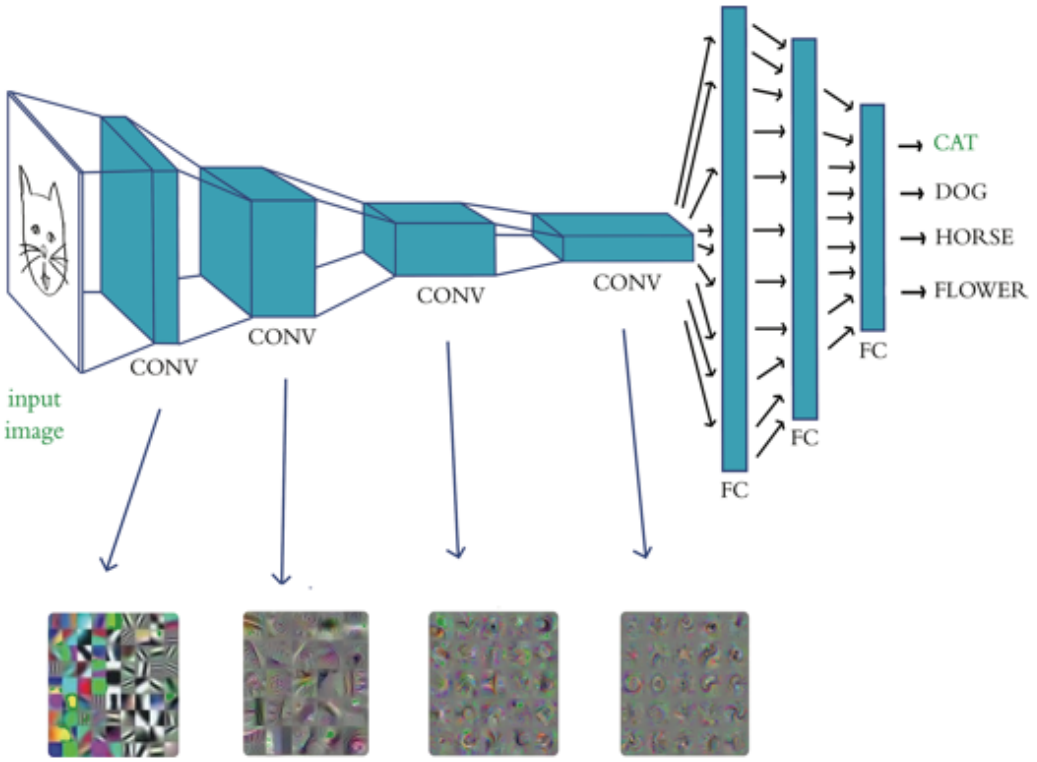



**Figura 5: Exemplo de rede neural convolucional (extraído de KROHN; BEYLEVELD; BLASSENS, 2020).**

Como é possível ver na Figura 5, uma rede convolucional não possui somente camadas convolucionais (representadas pelos retângulos cúbicos com legenda CONV), mas também camadas densas (representadas pelos 3 retângulos ao final com legenda FC). As camadas convolucionais possuem 3 dimensões, pois as duas primeiras representam as dimensões dos filtros enquanto a terceira representa a quantidade de filtros daquela camada. Já as camadas densas possuem somente 2 dimensões, as quais representam as dimensões da matriz dos neurônios daquela camada, a qual possui, em geral, tamanho um de largura e tamanho de altura igual à quantidade de neurônios daquela camada, funcionando na prática como um vetor. Normalmente, a arquitetura de uma rede CNN se dá de forma que as primeiras camadas são convolucionais, enquanto ao final a matriz é "achatada" em um vetor, possuindo então ao fim algumas camadas densas, antes da camada de *output*.

A Figura 6 é um exemplo do tipo de processamento que ocorre em uma camada convolucional.

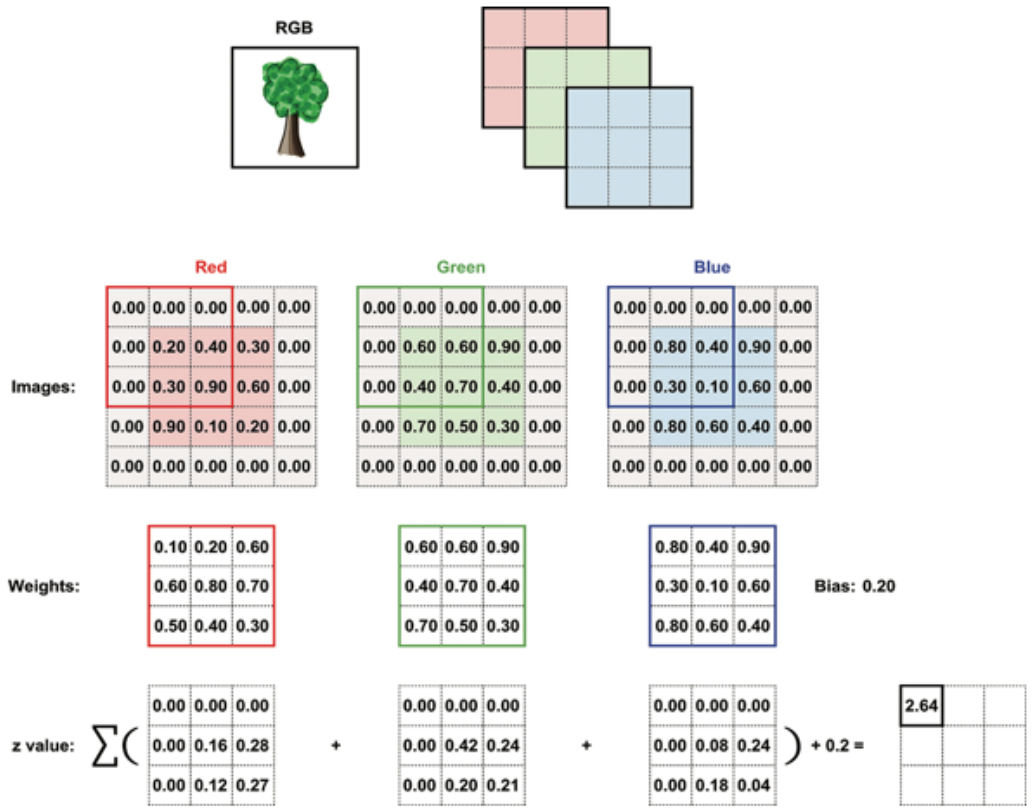



**Figura 6: Exemplo de processamento de uma camada convolucional (extraído de KROHN; BEYLEVELD; BLASSENS, 2020).**

Neste exemplo, os quadrados da primeira linha representam os canais RGB de uma imagem colorida. Já os do meio representam filtros (os quais normalmente possuem dimensões 2x2 ou 3x3), com seus respectivos pesos em cada posição da matriz. Estes escaneiam a imagem, realizando a operação de multiplicação matricial, resultando por fim nas matrizes da última linha, cujos valores são somados para gerar cada posição da matriz do canto inferior direito. Essa matriz resultante final, é chamada de **mapa de ativação**, a qual servirá de entrada para a próxima camada.

Cada camada convolucional possui diferentes hiperparâmetros. Os mais utilizados são o *kernel size*, o *stride length* e o *padding*.

### 2.7 Padding

Um exemplo de *padding* pode ser visto na Figura 6. É possível verificar, nas matrizes da primeira linha, as quais representam os canais RGB, que os valores das bordas são todos “0.00” . Isso porque esses valores não fazem parte da imagem de fato, mas foram adicionados de maneira artificial, com o objetivo de manter o tamanho da matriz resultante (mapa de ativação) igual ao das matrizes que representam cada canal da imagem. É possível perceber que, caso essa borda não tivesse sido adicionada, o tamanho da matriz iria diminuir ao final do processo de passar o filtro pela imagem, de modo que a camada seguinte receberia como entrada uma matriz menor. Isso não representa necessariamente um problema, porém por vezes pode querer ser evitado.

### *2.8 Kernel Size*

O hiperparâmetro *kernel size* (tamanho do *kernel*) representa as dimensões da matriz dos kernels, ou seja, dos filtros daquela camada. O *kernel size* do exemplo da Figura 6, é 3x3, porém também é comum usar o tamanho 5x5 e, às vezes, 7x7. Um tamanho de *kernel* muito grande pode ser ruim, pois muitas características (*features*) da imagem estarão sendo processadas ao mesmo tempo, o que pode dificultar o aprendizado. Tamanhos muito pequenos também podem acabar prejudicando, impossibilitando aprendizados de estruturas mais complexas, já que é possível acabar perdendo características espaciais pelo fato de o filtro estar limitado a uma quantidade muito pequena de *pixels*.

### *2.9 Stride Length*

Por fim, o *stride length* representa a quantidade de “passos” dados horizontal e verticalmente a cada movimentação do filtro sobre a imagem. Normalmente é utilizado o valor 1, no qual o filtro se movimenta um *pixel* de cada vez. Valores muito pequenos podem aumentar o custo computacional de processamento rede, aumentando o tempo de treinamento, enquanto valores altos podem acabar pulando regiões relevantes da

imagem e prejudicando no aprendizado de representação de *features* relevantes pelas camadas convolucionais.

### *2.10 Pooling Layers*

As *pooling layers* são muitas vezes utilizadas junto com as camadas convolucionais, com o intuito de reduzir a complexidade computacional de processamento da rede e prevenir o *overfitting*, a partir da redução dos parâmetros desta. As *pooling layers* funcionam então diminuindo o tamanho da matriz do mapa de ativação. A Figura 7 representa um exemplo de *max pooling*, no qual um filtro (nesse caso de tamanho 2x2) passa pelo mapa de ativação (da mesma forma que ocorre no processo de convolução), de modo que a cada etapa mantêm somente o maior valor daquela respectiva área.

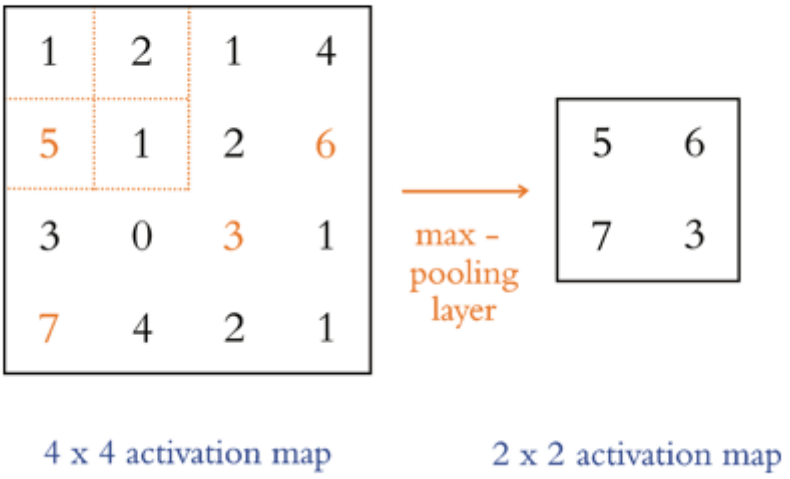



**Figura 7: Exemplo de operação de *max pooling* (extraído de KROHN; BEYLEVELD; BLASSENS, 2020).**

Dessa forma, neste exemplo, as dimensões do mapa de ativação diminuem de 4x4 para 2x2.

## 2.11 Outros conceitos relacionados a redes neurais

Antes de treinar uma rede neural, é preciso definir diversas configurações de extrema importância. Dependendo do que for definido, a rede pode acabar obtendo ótimos ou péssimos resultados, sendo que diferentes tipos de configurações são indicados para diferentes tipos de tarefas. A seguir serão apresentadas mais a fundo alguns conceitos relacionados a essas configurações.

## 2.12 Função de Ativação

A função de ativação possui grande relevância no desempenho de uma rede neural. Para entendê-la melhor, é necessário também entender como ocorre o processamento dos dados em cada camada de uma rede.

Nesse caso, será usado como exemplo uma rede densa, que é a mais comum e que facilitará o entendimento para este caso. Este tipo de rede pode possuir diversas camadas e diversos neurônios por camada (essa quantidade vai depender da arquitetura da rede), sempre começando com a camada de *input*, terminando com a de *output* e no meio contendo as chamadas *hidden layers*. Cada neurônio, neste caso, é representado pelo valor de uma posição específica de uma matriz de uma dimensão (vetor). Ou seja, na prática o processamento de uma rede neural é totalmente baseado em operações matemáticas. É utilizado o nome neurônio para fazer a associação destas unidades com neurônios de um cérebro humano. Cada um deles é associado a valores como peso e *bias*, ajuda no processamento da entrada e colabora na geração da saída. Esses valores associados vão então sendo atualizados durante o processo de treinamento (aprendizado da rede).

É importante ressaltar que no caso das redes densas, cada neurônio de uma camada é ligado com todos os neurônios da camada posterior. A *input layer* basicamente serve para receber os dados de *input*, portanto ela precisa ter a mesma dimensão que os dados que irão ser processados. Já a quantidade de *hidden layers* varia, assim como a quantidade de neurônios de cada uma delas.

Estas camadas efetuam o processamento dos dados de entrada, camada a camada, de modo que a saída de cada uma delas vira a entrada da camada imediatamente posterior. Cada camada efetua operações matemáticas sobre os dados

que recebem, multiplicando o valor de entrada pelos pesos dos neurônios, somando esse valor processado ao *bias* e passando esse resultado pela função de ativação. Esta permite que a rede passe a modelar relações de não linearidade entre a entrada e saída de dados, para cada camada, de modo a conferir maior capacidade de aprendizado à rede ao introduzir esse tipo de relação mais complexa. Como lado negativo, essas relações de não linearidade acabam por aumentar o custo computacional da rede, de modo a aumentar assim o tempo de treinamento da rede.

As funções de ativação podem ajudar a impedir o *vanishing gradient*, o qual ocorre quando o gradiente tende a zero e a rede começa a perder a capacidade de se otimizar e, dessa forma, aprender. Esse tipo de problema costuma ocorrer em redes com um grande número de camadas, de modo que o *backpropagation* acaba por vezes não sendo tão efetivo nas camadas iniciais.

Um dos pré-requisitos da função de ativação, é que os resultados dela devem ter igual probabilidade de serem positivos ou negativos. Isso deve ocorrer pois, caso o resultado seja sempre positivo ou sempre negativo, o ajuste dos pesos e *biases* também sempre seguirá somente uma direção, o que poderá acabar por prejudicar a qualidade de aprendizado da rede.

Além disso, pelo fato dos pesos e *biases* da rede serem ajustados através do processo de *backpropagation*, o qual ocorre através de cálculos diferenciais, a função de ativação deve ser diferenciável, de modo que seja possível obter sua derivada.

As principais funções de ativação são a *sigmoide*, *tangente hiperbólica*, *Softmax*, *Relu* e *elu*. As funções que foram utilizadas nesse trabalho foram a *elu* (unidade linear exponencial) e a *softmax* (esta somente para as redes treinadas para classificação).

A Figura 8 exemplifica a função *elu* e sua derivada.

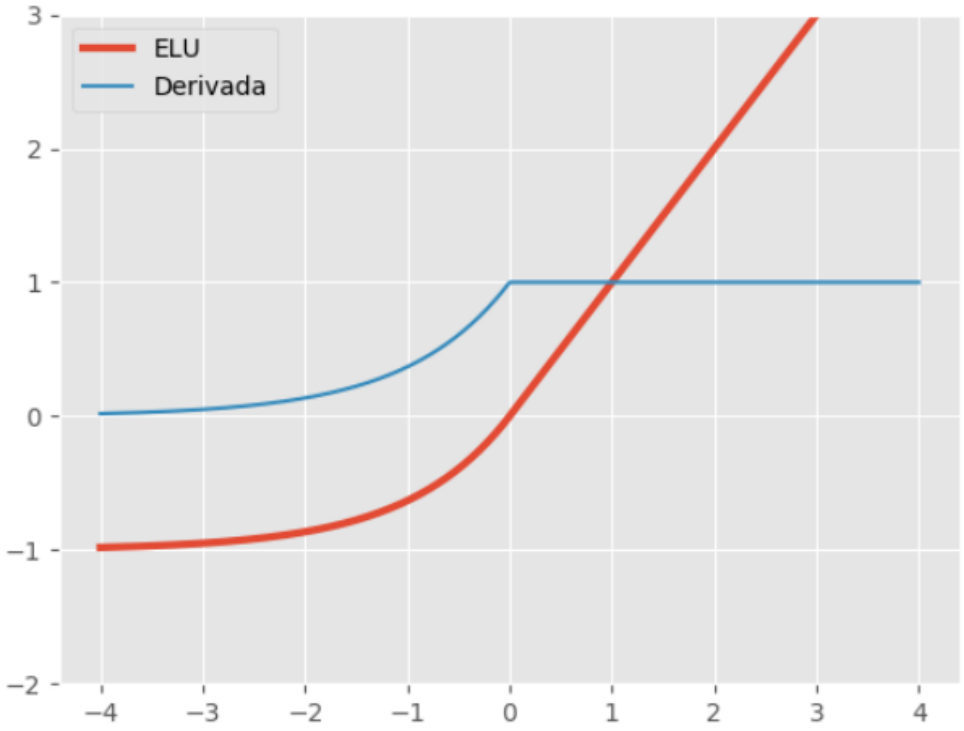



**Figura 8: Função de ativação elu e sua derivada [4].**

### 2.13 *Bias*

Como foi falado anteriormente, em cada *hidden layer*, os valores de *input* são multiplicados pelos pesos, somados com o valor do *bias* e por fim processados pela função de ativação. No final das contas, o que uma rede neural faz é criar uma relação entre dados de entrada e uma saída específica. O propósito dos pesos, é definir o impacto que cada variável de entrada têm em relação à saída que será gerada. Esse valor vai sendo ajustado durante o treinamento da rede, mais especificamente durante o processo de *backpropagation*, o qual se repete diversas vezes durante o processo de aprendizagem.

Já o *bias* é um valor constante e invariante à entrada, o qual é associado a cada neurônio. Seu objetivo é ajustar a saída, de modo que o valor resultante para cada um dos neurônios será sempre somado ao bias (para redes que estejam utilizando este artifício) e o somatório total, como explicado anteriormente, irá passar pela função de ativação. Isso pode ajudar para casos em que o resultado de determinado neurônio seja

[4] Disponível em: <https://matheusfacure.github.io/2017/07/12/activ-func/>. Acesso em 20 de Jul. 2022.

zero, já que este valor pode prejudicar o treinamento, pois qualquer número que o multiplique também resultará em zero. Portanto, o *bias* ajuda a rede a se adaptar melhor aos dados fornecidos.

## 2.14 *Backpropagation*

*Backpropagation* é um método utilizado para ajustar os pesos de uma rede neural, de acordo com o erro calculado pela função de custo. Durante o processo de treinamento de uma rede que utiliza treinamento supervisionado, a rede processa uma entrada gerando uma saída. Ao final, ela compara essa saída com um rótulo fornecido, utilizando alguma função de custo. De acordo com o erro calculado a partir desta função, a rede realiza o processo de *backpropagation*, no qual ela volta camada a camada, de trás pra frente, realizando operações diferenciais para calcular o acerto de cada peso, mudando assim seus valores, com o intuito de ir otimizando a função de custo e assim melhorando o desempenho da rede.

O algoritmo de *backpropagation* calcula então a perda do gradiente para cada peso, utilizando a regra da cadeia, de forma a calcular o quanto cada peso precisa ser ajustado. Este algoritmo é essencial para as redes neurais, visto que é ele que permite no final das contas que a rede, de fato, aprenda.

## 2.15 Função de perda

Como relatado anteriormente, a função de perda, também chamada de função de custo, quantifica o quão longe o resultado que a rede está gerando (predizendo) se encontra do valor real do rótulo, para determinada entrada. O valor de custo possui então uma relação inversamente proporcional com a qualidade da rede. Algumas funções de custo famosas são a *Categorical Cross-Entropy*, *Mean squared Error* e *Mean Absolute Error*.

## 2.16 Overfitting

*Overfitting* é um problema que pode ocorrer durante o processo de treinamento de uma rede. Para entender esse problema, é possível usar o seguinte exemplo: imagine um aluno que para estudar para uma prova usa como base uma lista de exercícios fornecida pelo professor, a qual possui o gabarito de cada uma das questões. Agora imagine que ao invés deste aluno aprender de fato a resolver as questões, ele acaba gravando as respostas das questões do simulado. Ao final, no momento da prova, a qual possui questões diferentes da lista, ele obtém um resultado péssimo apesar de ter ido muito bem no simulado. Isso se deve ao fato de que ele não aprendeu de fato a resolver as questões, mas somente gravou as relações entre questões específicas e seus resultados. Portanto, se alguma daquelas questões mudar, mesmo que pouco, ele não irá mais saber como chegar ao resultado.

Algo muito parecido pode ocorrer no caso de uma rede neural que está sendo treinada. Às vezes, por diferentes razões, a rede pode acabar modelando as relações de entrada e saída especificamente para os dados do *dataset* de treinamento. Isso pode ocorrer, por exemplo, no caso em que a quantidade de dados do dataset de treinamento seja pequena, de modo que o treinamento não seja capaz de generalizar aquele aprendizado para dados fora daquele *dataset*. Lembrando que os dados de validação são sempre diferentes dos dados do *dataset* de treinamento, justamente para comprovar que a rede está sendo capaz de generalizar o aprendizado obtido.

## 2.17 Learning Rate

O *learning rate* é um hiperparâmetro de treinamento da rede, o qual define o quanto que os parâmetros da rede serão ajustados de acordo com o gradiente da perda, o qual é calculado a partir da função de perda. A conta feita é a seguinte: peso atual menos gradiente vezes *learning rate* é igual ao novo peso. Portanto, o *learning rate* é relacionado também com a velocidade de treinamento da rede, de modo que quanto maior, maiores serão os ajustes a cada etapa do treinamento.

Isso pode levar ao seguinte questionamento: porque então não definir um valor bem alto para o *learning rate*, fazendo assim a rede treinar bem mais rápido? A resposta para essa questão é que, para um treinamento ser bem sucedido, os ajustes dos

parâmetros precisam ser feitos aos poucos, porque esse ajuste não é tão simples e, para ser bem feito, é preciso que muitos *inputs* tenham sido processados e suas perdas calculadas. No final das contas, ele determina o tamanho do passo que será dado, ao final de cada iteração, em uma direção que minimize a perda. Se o passo dado for muito largo, a direção correta pode ser perdida.

Além disso, é importante ressaltar que existe um ponto de equilíbrio ótimo para o valor de *learning rate*: se for muito pequeno, as alterações nos parâmetros serão igualmente pequenas e levarão a um tempo de treinamento excessivamente longo; por outro lado, se for muito grande, o processo de otimização tende a "pular" as regiões de ótimos locais, implicando em um valor elevado para a função de perda. O *learning rate* pode ser definido antes do processo de treinamento ou podem ser utilizadas funções de otimização como a Adam, a qual vai ajustando-o ao longo do processo de aprendizado, buscando encontrar o valor ótimo.

### 2.18 *Batches*

*Bacthes* são agrupamentos de dados do *dataset*, os quais são treinados juntos. Um *batch* reúne um conjunto de exemplos que passarão pelo treinamento. A quantidade de *batches* que uma rede irá possuir para cada época do treinamento depende do *batch size* definido, assim como a quantidade de dados disponíveis. Se, por exemplo, tivéssemos um *dataset* de treino de tamanho 10 mil e um um *batch size* de tamanho 200, a quantidade de *batches* seria 50. A quantidade de épocas também é um hiperparâmetro de treinamento da rede, de modo que a cada época, todos os dados passam uma vez pela rede. O tamanho do *batch* (*batch size*) normalmente é definido como uma potência de 2, sendo 32 e 64 dois valores muito utilizados.

### 2.19 *Época*

A quantidade de épocas também é um hiperparâmetro o qual pode ser ajustado de rede para rede. Cada época funciona como uma etapa do treinamento, de modo que para completar uma época o treinamento precisa passar por todos os *batches*, ou seja, por todos os itens do dataset de treino. A quantidade de épocas varia muito dependendo do tipo de treinamento e do que está sendo treinado. É muito comum que vários tamanhos de época sejam testados em busca de achar o valor ideal.

### 2.20 *Data Generator*

*Data generators* no contexto de redes neurais servem para gerar dinamicamente os *batches* de determinada fase do treinamento, o que é especialmente útil para o caso de treinamentos relacionados à imagens, por exemplo. Imagine um *dataset* de treinamento possuindo 10 mil imagens. Agora imagine o custo computacional de carregar em memória 10 mil imagens ao mesmo tempo. Ao invés disso, o que pode ser feito é salvar 10 mil *strings*, cada uma representando o caminho para cada imagem do *dataset*. Dessa forma, durante o treinamento, somente as imagens de determinado *batch* são carregadas ao mesmo tempo. Portanto, se um *batch* possui tamanho 64, somente 64 imagens serão carregadas, ao invés de 10 mil.

# 3 Revisão Bibliográfica

Este capítulo apresenta trabalhos que estão relacionados ao tema escolhido, isto é, que tenham lidado com o emprego de modelos de *deep learning* para a tomada de decisão de protótipos de veículos autônomos.

Para automatizar tarefas complexas como a direção de carros, é necessário pensar em todas as atividades e habilidades que permitem que um ser humano consiga levar um carro de um ponto A a um ponto B. Por exemplo, seria preciso que o carro de alguma forma identificasse a faixa e os limites da pista, para que conseguisse segui-la do início ao fim do trajeto, sem subir a calçada ou cometer qualquer outra infração do gênero. Outra habilidade primordial seria que o carro deveria ser capaz de detectar e reconhecer semáforos e sinais de trânsito, além de saber tomar uma decisão em cima desse reconhecimento. Teria também que conseguir detectar objetos em geral, para evitar colisões com outros carros, pessoas, animais, ou qualquer tipo de objeto que possa cruzar seu caminho na pista.

Para esse tipo de aprendizado, existe um método conhecido como clonagem de comportamento (*behavioral cloning)*. Nele, habilidades de um ser humano podem ser aprendidas e posteriormente reproduzidas, com exatidão, por um programa de computador. Com intuito de tornar isso possível, são capturadas as reações de um ser humano em resposta a determinados estímulos, de modo que essas ações possam depois servir de base para o aprendizado de um programa. Essa técnica se encaixa quase que perfeitamente no caso de uma rede neural que utiliza aprendizado supervisionado. Nesse caso, os estímulos do ambiente servem como entrada da rede, enquanto as ações de resposta servem como rótulos, para que durante o treinamento haja a comparação com as saídas da rede, possibilitando assim que a rede ajuste seus parâmetros internos e, por fim, aprenda a atividade desejada.

Portanto, dadas tantas habilidades primordiais para que seja possível a manutenção de uma direção autônoma e segura, é preciso um sistema eficiente e ao mesmo tempo econômico, para que seja possível um dia popularizar esse tipo de carro,

tornando mais viável a sua produção e levando maior segurança às ruas e conforto às pessoas.

Primeiramente será tratada a habilidade do carrinho seguir a pista de modo autônomo. Este trabalho foca em modelos de treinamento *end-to-end*, no qual uma entrada é fornecida e uma saída é gerada, ao invés do modelo *pipeline* padrão. Neste último, são realizados uma série de processamentos em sequência (os processamentos variam de caso em caso) e, por isso, ele é conhecido como *pipeline*. Normalmente, no modelo *pipeline*, são utilizados métodos para detectar as bordas da pista na imagem e calcular uma linha imaginária entre elas, de modo que seja traçada então a trajetória que o carrinho deve seguir.

Porém, utilizando esse tipo de método, a quantidade de cálculos, processamento e consequentemente poder computacional são bem maiores do que no modelo *end-to end*. Tratando-se de um modelo de carro autônomo, no qual é preciso que sejam dadas respostas rápidas, caso contrário se torna impossível uma direção autônoma, e dado que o objetivo desse trabalho é conseguir alcançar isso da forma mais eficiente e com o um custo baixo de hardware, um modelo *end-to end* se apresenta como uma solução mais viável.

Entretanto, é possível pensar que uma rede neural, adotando uma estratégia *end-to-end*, também não é um tipo de algoritmo leve em termos computacionais, de modo que exigiria um maior poder computacional envolvido e, por conseguinte, um maior custo de hardware. Porém, como será possível avaliar mais adiante, é possível utilizar redes mais simples as quais não demandam tanto computacionalmente, e que atingem níveis satisfatórios de acurácia. Sendo um modelo *end-to-end,* as saídas são fornecidas mais rapidamente (em frações de segundos), de modo que para o objetivo deste trabalho, é um tempo suficientemente bom para permitir que o carrinho consiga navegar pela pista.

Abaixo serão apresentadas, então, técnicas e modelos utilizados em diferentes trabalhos para tornar o carrinho capaz de seguir a pista, de modo autônomo.

Em (SWAMINATHAN *et al.*, 2019), é utilizada a biblioteca *OpenCV*, para que fosse possível detectar as bordas da pista, gerando por fim uma das seguintes saídas: frente, trás, direita e esquerda. *OpenCV* [5]é uma biblioteca de código aberto desenvolvida

[5] https://opencv.org/

pela Intel, que disponibiliza algoritmos voltados para a área de visão computacional. Ela é utilizada, por exemplo, para atividades de reconhecimento facial, edição de fotos e vídeos, detecção de objetos assim como seu rastreamento, etc.

Já em (CUPERTINO, 2018), são utilizados 2 modelos de aprendizagem profunda: uma CNN e uma LSTM (*Long Short-Term Memory*). A primeira foi utilizada com intuito de predizer o ângulo de direção das rodas, para que o carrinho pudesse seguir as curvas da pista, enquanto a LSTM foi utilizada para predizer a aceleração do carrinho, também de acordo com a pista. A CNN foi utilizada com esse objetivo, pois ela é um modelo de rede neural eficiente para detecção e reconhecimento de imagens.

No caso deste trabalho, a rede recebe como entrada uma imagem, que no caso seria a imagem da pista logo à frente do carrinho, e gera como saída um *array* de 10 posições, em que uma dessas posições possuiria “1" como valor enquanto as outras possuiriam “0”. O número da posição do *array* que possuísse valor “1”, seria a resposta do ângulo de direção que as rodas do carrinho precisam virar para fazer a curva corretamente. Caso a posição 0 possuísse valor “1”, significaria que o carrinho precisa virar 100% para a esquerda, enquanto se fosse a posição 9, 100% para a direita.

A CNN utilizada neste trabalho faz uso de camadas *batch normalization*, possuía uma camada de *max pooling* e um *stride* de tamanho 2 (para cada camada convolucional). Ao final era feito o achatamento (flattening) dos valores em um *array*, seguido por algumas camadas densas e um *dropout*. Foi utilizado o otimizador Adam, a função de erro quadrático, como função de erro, a função de ativação ReLU e, para a camada de saída, a função de ativação Softmax. A Figura 9 ilustra a rede criada no trabalho.

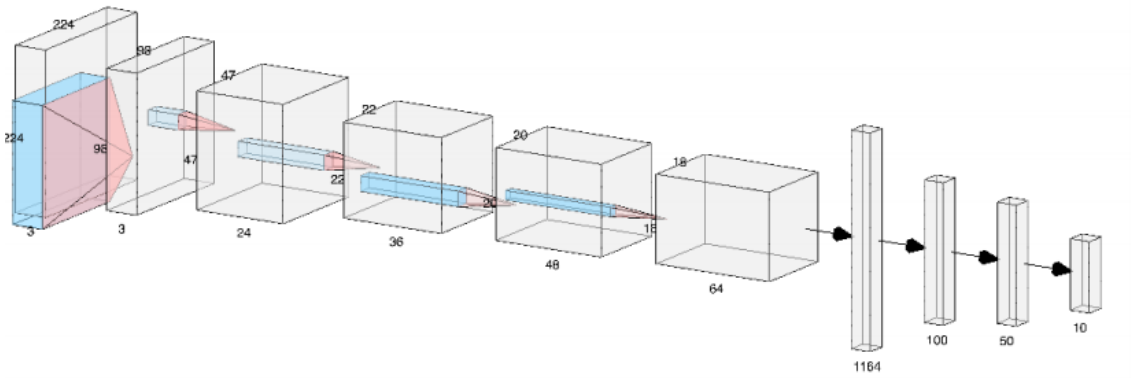



**Figura 9:** Representação da Rede Neural Convolucional utilizada (extraído de CUPERTINO, 2018).

Os primeiros retângulos cúbicos representam as camadas convolucionais da rede, enquanto a partir da camada com 1164 neurônios é feito o flattening da rede, de modo que a partir desse ponto as camadas utilizadas são densas.

Já a LSTM foi utilizada com o objetivo de predizer a aceleração do carrinho, pois é um tipo de rede que leva em conta valores de saída anteriores (o que é uma característica de redes recorrentes). Tomemos como exemplo o seguinte caso: se, na última predição, a LSTM gerou um valor de saída 100, ela não irá gerar na próxima predição um valor 50 ou mesmo 150. No máximo irá variar entre 80 e 120. Dessa forma, o carrinho não irá fazer acelerações ou frenagens bruscas, mas sim suaves.

O modelo LSTM utilizado possui então três portões: um portão de entrada, um de saída e por fim um de esquecimento. O portão de entrada serve para decidir quais valores do *input* a rede deve atualizar. O de saída decide qual saída será gerada baseada no *input* e na memória da unidade. Já o de esquecimento decide quais valores de memória serão descartados pela unidade. Dessa forma, a rede vai sempre realizando a manutenção dos valores convenientes que deve manter em memória. A LSTM utilizada é representada na Figura 10.

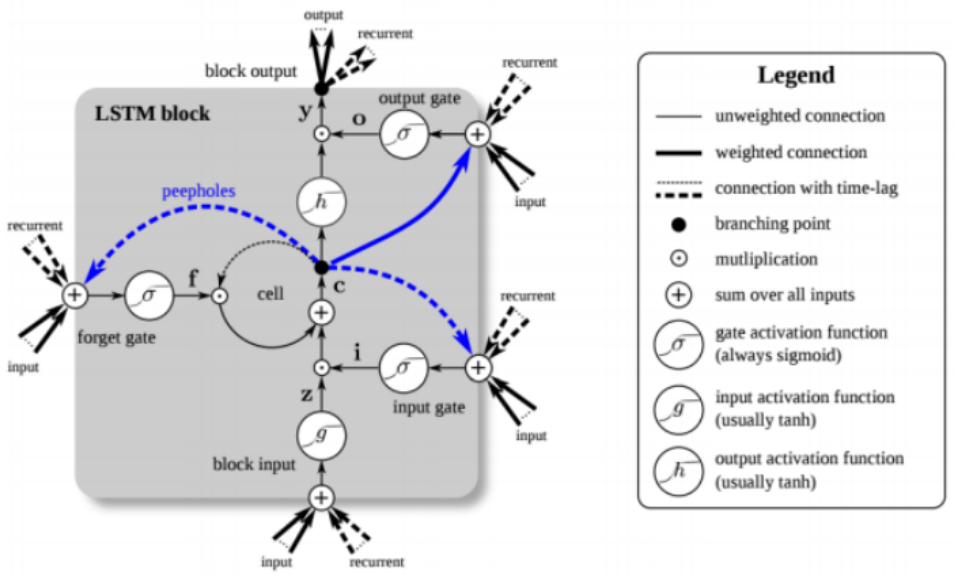



**Figura 10:** Representação da rede LSTM utilizada (extraído de CUPERTINO, 2018).

No trabalho de (LEÓN-VERA *et al.*, 2019), para predição do ângulo de direção também foi utilizada uma rede neural convolucional (CNN). No caso desse trabalho, a configuração da CNN adotada era a seguinte: Conv24 - Conv36 - Dropout - Conv48 - Dropout - Conv64 - Dropout - Conv64 - Dropout - FC - FC - FC - FC, em que *Conv* representa uma camada convolucional e *FC*, uma camada densa. Ademais, foi utilizada a função de ativação ReLU, o *dropout* era com probabilidade de 0.5% (para tentar evitar o *overfitting*), função de erro quadrático e otimizador Adam.

Já no trabalho de (DEVI *et al.*, 2020), também foi utilizado um modelo CNN. A rede recebia como entrada uma imagem em formato YUV e gerava como saída o ângulo de direção do carrinho.

O *color space* YUV é um tipo de codificação de cores, diferente do RGB, o qual é famoso por ser utilizado nas tvs analógicas. É uma abreviação para *luma* (Y), *red projection* (U) e *blue projection* (V), de modo que o Y define a quantidade de brilho, enquanto o U e o V definem as cores.[6]

[6] Disponível em <https://dexonsystems.com/news/rgb-yuv-color-spaces>. Acesso em 20 de Ago. 2022.

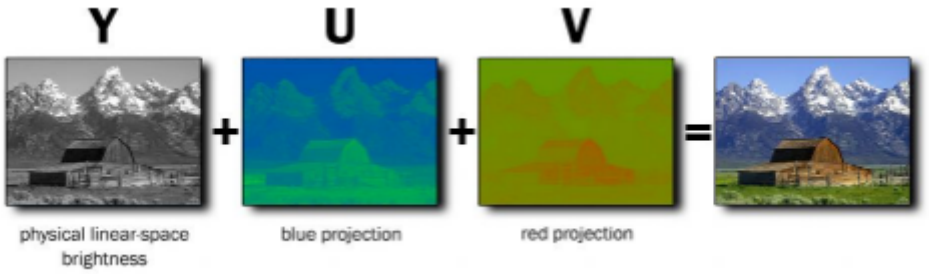



**Figura 11:** Representação de cada letra da abreviatura do formato YUV [6].

O artigo explica que foi utilizada uma rede convolucional de 17 camadas, otimizador Adam e função de ativação ReLU. Das 17 camadas, cinco eram convolucionais, com tamanho de *kernel* 5x5 e *padding* para manter o tamanho da imagem. Foram utilizadas também uma camada *dropout* e uma camada *max pooling* com janela 2x2. A Figura 12 ilustra a rede utilizada.

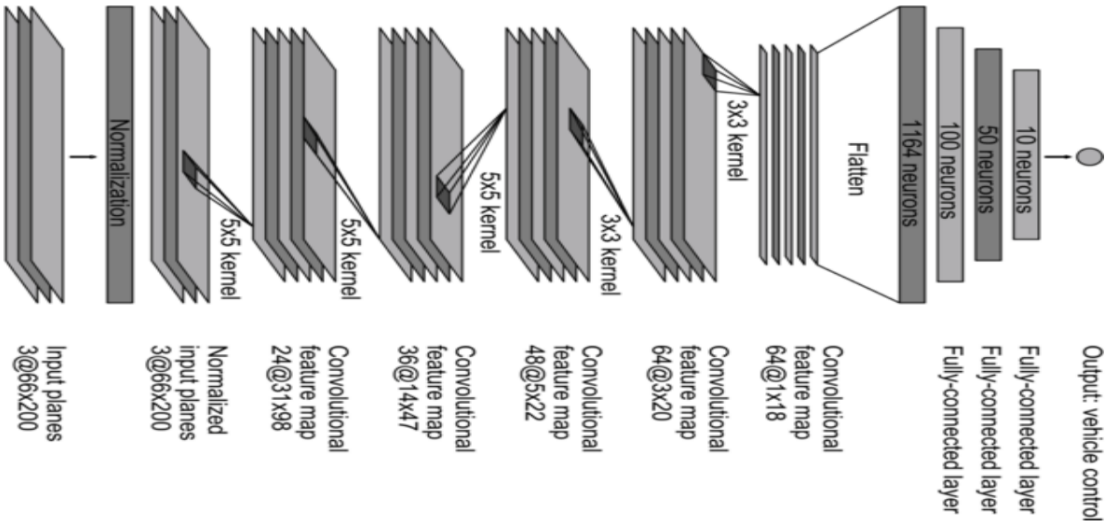



**Figura 12:** Representação de rede convolucional (extraído de DEVI *et al.*, 2020).

Por sua vez, no trabalho de (YIQIN *et al.*, 2018), foi utilizada uma *Improved Stack Autoencoder*. Essa arquitetura *Improved Autoencoder* é utilizada na fase de pré-treino, quando é feito uma espécie de treinamento não supervisionado. Essa fase serve então para definir os pesos iniciais, gerando a rede camada por camada. Segundo o artigo, essa técnica serve para aumentar a capacidade de generalização da rede, diminuir o tempo de treinamento (de modo que a rede atinja uma boa acurácia mais

rapidamente) e evitar problemas como que a rede caia em um mínimo local, já que em redes profundas o problema de desaparecimento do gradiente (*vanishing gradient*) pode prejudicar o treinamento.

Depois dessa fase preliminar, ocorre a fase de treinamento da rede propriamente dita, na qual uma rede neural é treinada de modo supervisionado, a partir de imagens rotuladas (ou seja, treinamento supervisionado), cujos rótulos representavam as curvas que deveriam ser feitas de acordo com a imagem (direita, esquerda ou seguir em frente). Na rede, foi utilizada a função de ativação ReLU, além de *dropout* e *batch normalization*. Este último foi adicionado à camada de *input* e possuía o objetivo de ajustar os dados do *input*. Por fim, na camada de saída, é utilizada a função de ativação Softmax, a qual é muito utilizada em casos em que a rede precisa gerar uma classificação, entre múltiplas classes (mais do que duas), como saída. No caso do trabalho, as classes eram "esquerda", "direita" e "seguir em frente". É importante ressaltar que, no caso do trabalho em questão, foi utilizada uma rede puramente densa (*fully connected*), sem nenhuma camada convolucional, o que acaba indo na contramão do que os outros trabalhos citados fizeram. Uma representação da rede pode ser vista na Figura 13.

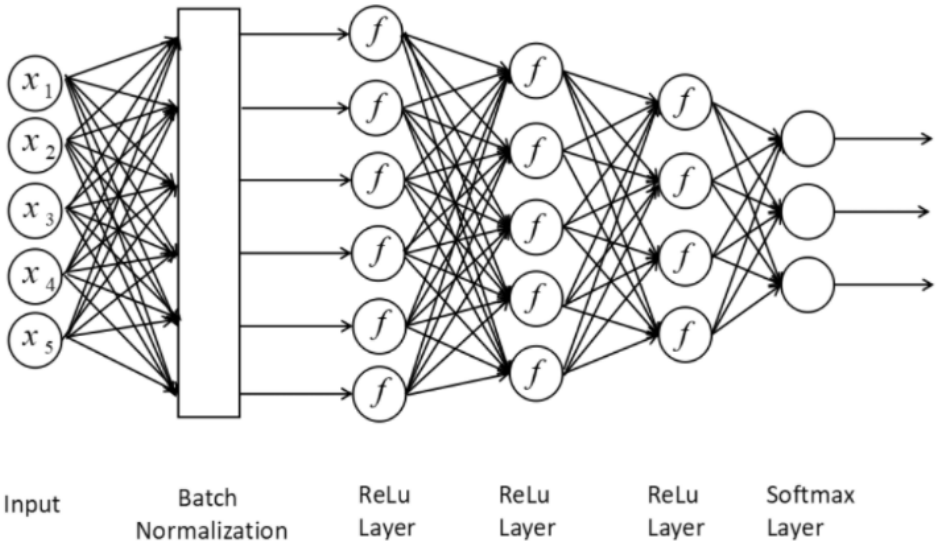



**Figura 13:** Representação de rede *fully connected* (extraído de YIQIN *et al.*, 2018).

O artigo apresenta também uma imagem dos pesos localizados entre a camada de entrada e a primeira camada oculta, nas etapas de pré-treinamento e de treinamento. É possível verificar, já no pré-treinamento, os contornos da pista, a partir da Figura 14.

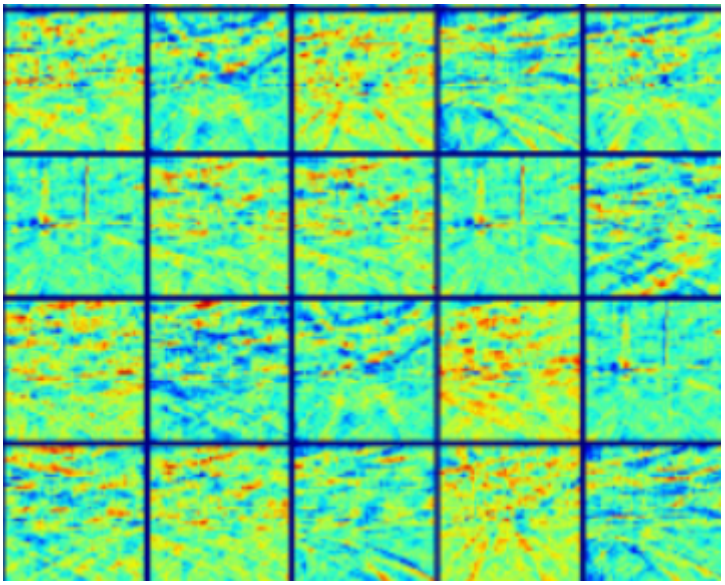


**Figura 14:** Representação de pesos entre a camada de entrada e camada oculta, após a etapa de pré-treinamento de rede *fully connected* (extraído de YIQIN *et al.*, 2018).

Após a etapa de treinamento, é possível ver esses contornos bem mais claramente, tal como refletido na Figura 15.

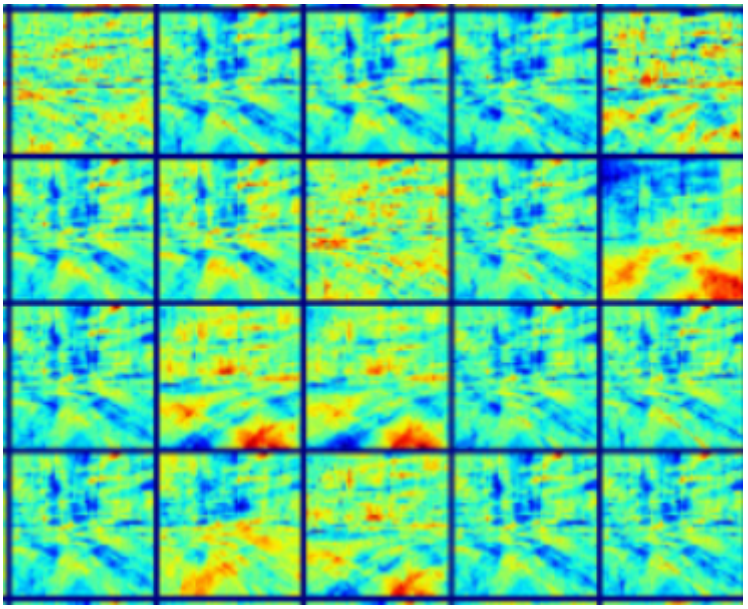


**Figura 15:** Representação de pesos entre a camada de entrada e camada oculta, após a etapa de treinamento de rede *fully connected* (extraído de YIQIN *et al.*, 2018).

Finalmente, é possível ver nos pesos da camada de saída o significado claro da classificação predita pela rede, tal como ilustrado pela Figura 16. Note-se que cada uma das três cores representa uma saída diferente (esquerda, direita e seguir em frente).

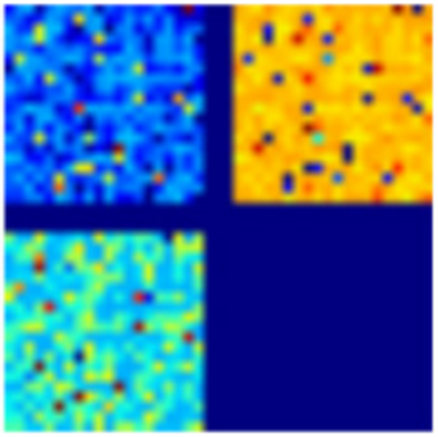


**Figura 16:** Representação de pesos da camada de saída, após a etapa de treinamento de rede *fully connected* (extraído de YIQIN *et al.*, 2018).

Por seu turno, a Figura 17 exibe o resultado obtido em dois experimentos feitos pelos autores com o carrinho, em que a marca preta designa a pista e a branca, a trajetória que o carrinho executou de fato. Esses experimentos foram realizados sem o uso da rede *Stacked Autoencoder* como pré-treinamento, à medida que a Figura 18 ilustra a trajetória obtida pelo carrinho com o auxílio dessa estratégia.

(a) One of the experiments (b) Another experiments

**Figura 17:** Duas imagens que mostram a pista construída, em preto, e a trajetória feita pelo carrinho, em branco, quando não foi adotado o pré-treino na rede neural (extraído de YIQIN *et al.*, 2018).

**Figura 18:** Duas imagens que mostram a pista construída, em preto, e a trajetória feita pelo carrinho, em branco, quando foi adotado o pré-treino na rede neural (extraído de YIQIN *et al.*, 2018).

Em (SAJJAD *et al.*, 2021), para que o carrinho aprendesse a seguir a pista, foi utilizada uma rede *fully connected* com apenas duas camadas ocultas. Segundo os autores, eles começaram utilizando uma rede simples, com 128 neurônios na primeira camada oculta e apenas 16 na segunda, e foram evoluindo paulatinamente para verificar o mínimo que alcançaria uma acurácia aceitável. Por fim, acabaram terminando com 4800 neurônios na primeira camada oculta e 64 na segunda.

A saída da rede continha 4 classes, que eram “direita”, “esquerda”, “frente” e “trás”, porém, segundo o artigo, a classe de “trás” não chegou a ser utilizada. A Figura 19 é uma ilustração da rede criada.

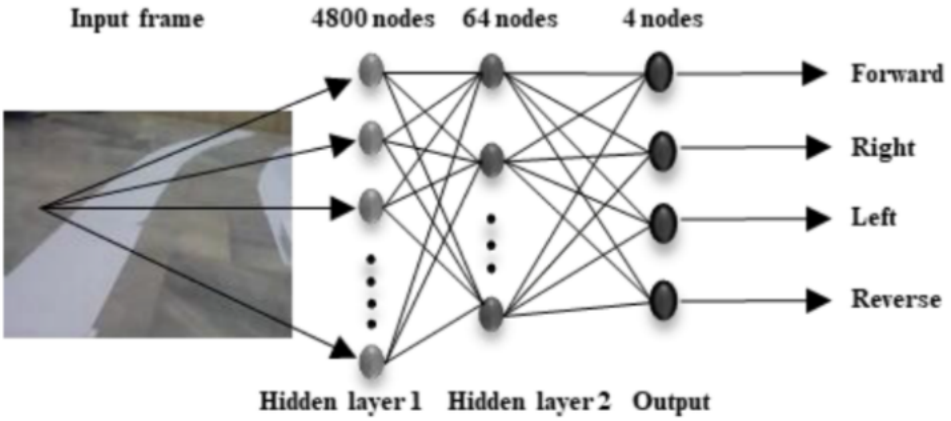



**Figura 19:** Representação da rede *fully connected* utilizada (extraído de SAJJAD *et al.*, 2021).

Por fim, o artigo apresenta no Quadro 1 os valores de acurácia obtidos para cada uma das saídas possíveis do modelo. Como é possível ver, a rede somente alcançou uma boa acurácia nas predições referentes à classe “frente” (*straight*).

**Quadro 1:** Acurácia observada para as possíveis saídas da rede (extraído de SAJJAD *et al.*, 2021).

| Frame | Accuracy |
|---|---|
| Straight | 96.2 |
| Left | 36.1 |
| Right | 33.5 |
| Reverse | Optional |

# 4 Arquitetura adotada e arduino

O presente capítulo possui como objetivo apresentar a arquitetura de rede neural utilizada, assim como o ambiente computacional e os componentes de hardware necessários para construção do protótipo. Além disso, explica como funcionou o fluxo de comunicação entre os componentes dentro do sistema distribuído construído.

## 4.1 *Arquitetura adotada*

A arquitetura usada foi a NVidia[7] Dave-2, a qual já havia sido testado com sucesso pela própria NVidia para essa mesma função de direção autônoma, conforme demonstrado em (CALDERON *et al.*, 2021). Dessa forma, esse modelo já possui um histórico de sucesso nessa área, o que proporciona segurança quanto à capacidade de aprendizado referente a esse tipo de atividade e fornece um ponto de partida.

A arquitetura possui 9 camadas, sendo 5 convolucionais (3 camadas com filtro 5x5 e 2 camadas com filtro 3x3), 3 densas (uma com 100 neurônios, outra com 50 e outra com 10) e uma de saída, como exposto na Figura 20 e 21.

[7] NVidia é uma das principais empresas mundiais de placas gráficas: https://www.nvidia.com/gtc/?ncid=pa-srch-goog-771271

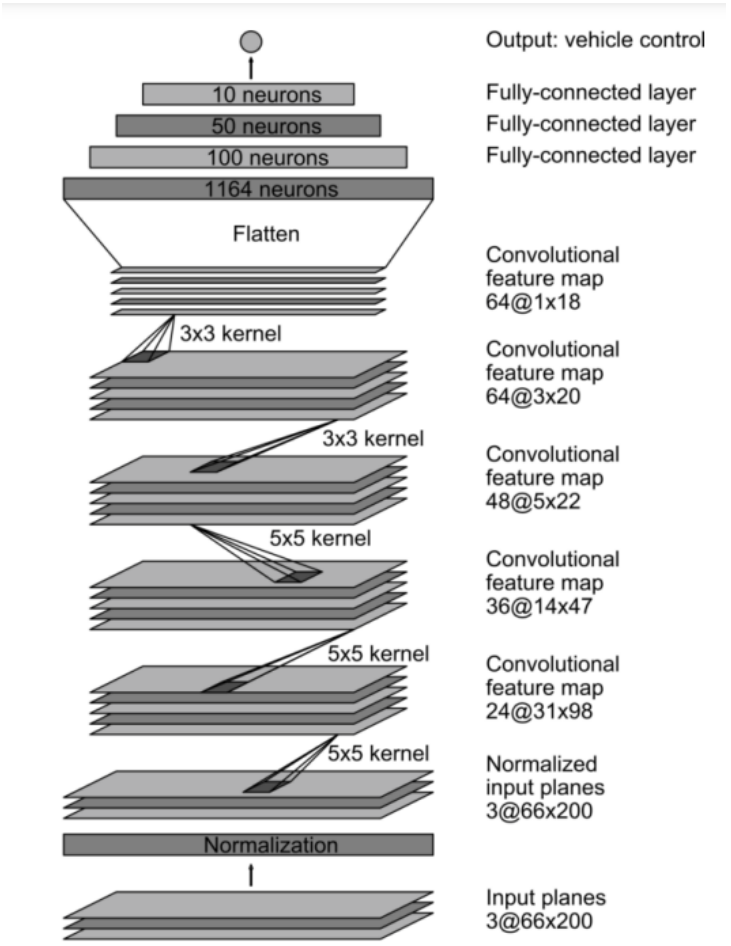



**Figura 20:** Representação da arquitetura da rede NVidia Dave-2 (extraído de CALDERON *et al.*, 2021).

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 lambda_2 (Lambda)            (None, 160, 320, 1)       0

 conv2d_8 (Conv2D)            (None, 78, 158, 24)       624

 conv2d_9 (Conv2D)            (None, 37, 77, 36)        21636

 conv2d_10 (Conv2D)           (None, 17, 37, 48)        43248

 conv2d_11 (Conv2D)           (None, 8, 18, 64)         27712

 conv2d_12 (Conv2D)           (None, 3, 8, 64)          36928

 dropout_2 (Dropout)          (None, 3, 8, 64)          0

 flatten_2 (Flatten)          (None, 1536)              0


 dense_8 (Dense)              (None, 100)               153700

 dense_9 (Dense)              (None, 50)                5050

 dense_10 (Dense)             (None, 10)                510

 dense_11 (Dense)             (None, 1)                 11

=================================================================
Total params: 289,419
Trainable params: 289,419
Non-trainable params: 0
_________________________________________________________________
```

**Figura 21:** Resumo da rede NVidia Dave-2.

É possível perceber que a camada de saída da rede descrita possui somente um neurônio. Isso se deve ao fato de que a tarefa a qual a rede foi treinada pela *NVidia* para realizar foi de regressão e não de classificação. Desse modo, a saída a ser predita pela rede é contínua, representando o ângulo de direção que o carro deve virar de acordo com a imagem da pista naquele exato instante.

No Capítulo 5, são descritos diversos experimentos de treino desta rede e teste com o carrinho e componentes relacionados. Ao longo destes experimentos, diversas

redes foram treinadas, de modo que alguns treinamentos foram realizados para tratar o problema como de regressão (assim como no caso da arquitetura proposta pela NVidia), enquanto outros para tratá-lo como de classificação, de modo que é feita a comparação entre o desempenho de ambos no capítulo 5.

Para adaptar a rede para treinar para classificação, sua camada de saída passou a possuir 3 neurônios: o primeiro representa que o carrinho deveria virar para a esquerda, o segundo significa seguir em frente, enquanto o terceiro seria virar para a direita. No caso de um problema de classificação, é definida uma quantidade específica de neurônios para a camada de saída, quantidade a qual depende de quantas classes o problema inclui. Quando a rede realiza a classificação, ela gera um vetor correspondente a uma distribuição de probabilidade, em que cada neurônio da camada de saída possui um valor entre 0 e 1, de modo que a soma destes deve resultar 1. O neurônio que possuir maior valor é o que tem mais chance de ser a saída correta para o exemplo apresentado e é portanto o resultado da rede.

## 4.2 *Dataset*

As imagens que serviram para o treinamento e validação da rede foram obtidas através do simulador *UDacity*[8], o qual foi feito na plataforma *Unity*[9] e, ao ser rodado, possui a opção de gerar um *dataset* com as imagens da pista, assim como uma planilha com o caminho do diretório para cada imagem e o respectivo valor de *steering* e *throttle* para aquele instante registrado (além de outros valores). Lembrando que o valor de *steering* representa o ângulo de direção efetuado em determinado instante, enquanto o de *throttle* representa o valor de aceleração.

O simulador foi baseado no modo usado pela *NVidia* para obter seu *dataset*. Neste caso, ela rodou um carro por diversas estradas, gravando as imagens da pista e valores associados, principalmente *steering* e *throttle*. No final usou todo esse *dataset* gerado para realizar o treino e validação da rede criada.

O simulador possui duas pistas para criação de *dataset*, as quais serão apresentadas no capítulo de experimentos. Para alguns dos modelos treinados neste

[8] https://www.udacity.com/
[9] https://unity.com/

trabalho, foi utilizado um *dataset* composto de imagens de somente uma das pistas, enquanto para outros foi utilizado um *dataset* que possuía imagens de ambas. Ao longo do Capítulo 5, os experimentos são descritos com detalhes, incluindo os *datasets* utilizados.

## 4.3 *Ambiente computacional*

Todos os treinamentos foram feitos utilizando o *Google Colab*[10] e a biblioteca *Keras*[11], a qual é baseada em outra, chamada *TensorFlow*[12]. Ambas são bibliotecas muito utilizadas para ajudar com problemas relacionados a *deep learning*. A linguagem utilizada em todos os *scripts* criados (tirando os do arduino) foi *Python*, a qual é famosa por possuir diversas bibliotecas orientadas a tratar problemas relacionados a AI, ciência de dados, etc.

Por sua vez, o *Google Colab* disponibiliza um ambiente *on-line* para escrita e execução de *notebooks* em *Python* e que tem sido bastante utilizado nas áreas de Educação e Ciência de Dados, sobretudo para modelos baseados em aprendizado profundo. Isso se deve ao fato de que o processamento e o treinamento de redes neurais profundas exige um alto poder computacional, o qual não é acessível para boa parte da população. Dessa forma, esse ambiente disponibiliza processamento de CPUs, GPUs e TPUs em nuvem, de maneira gratuita.

Finalmente, é importante citar que o computador pessoal do autor, utilizado em uma parte do fluxo de treinamento da rede, possui microprocessador Intel I7 e memória Ram de 16gb.

## 4.4 *Fluxo de funcionamento do sistema distribuído*

Os melhores modelos treinados foram armazenados em pastas do *Google Drive*, em um arquivo do tipo h5. Arquivos h5 são salvos no formato *Hierarchical Data Format* (HDF), o qual serve para armazenar redes neurais para, por exemplo,

[10] https://colab.research.google.com/
[11] https://keras.io/
[12] https://www.tensorflow.org/

carregá-las depois em outro ambiente. Depois de dado treinamento, o melhor modelo gerado era baixado e aberto em um *script Python* diretamente no computador pessoal do autor. Esse *script* tinha a função de carregar o modelo de rede treinado, a partir da leitura do arquivo h5, se conectar com o *stream* de imagens fornecido pelo celular colocado na frente do carrinho, se conectar ao *bluetooth* do carrinho e entrar em um *loop* infinito, no qual as imagens do *stream* iriam ser usadas como *input* da rede neural e o output iria servir como base para definir para onde o carrinho deveria seguir e qual sinal mandar via *bluetooth* para o carrinho. A Figura 22 apresenta um esquema de visualização deste processo.

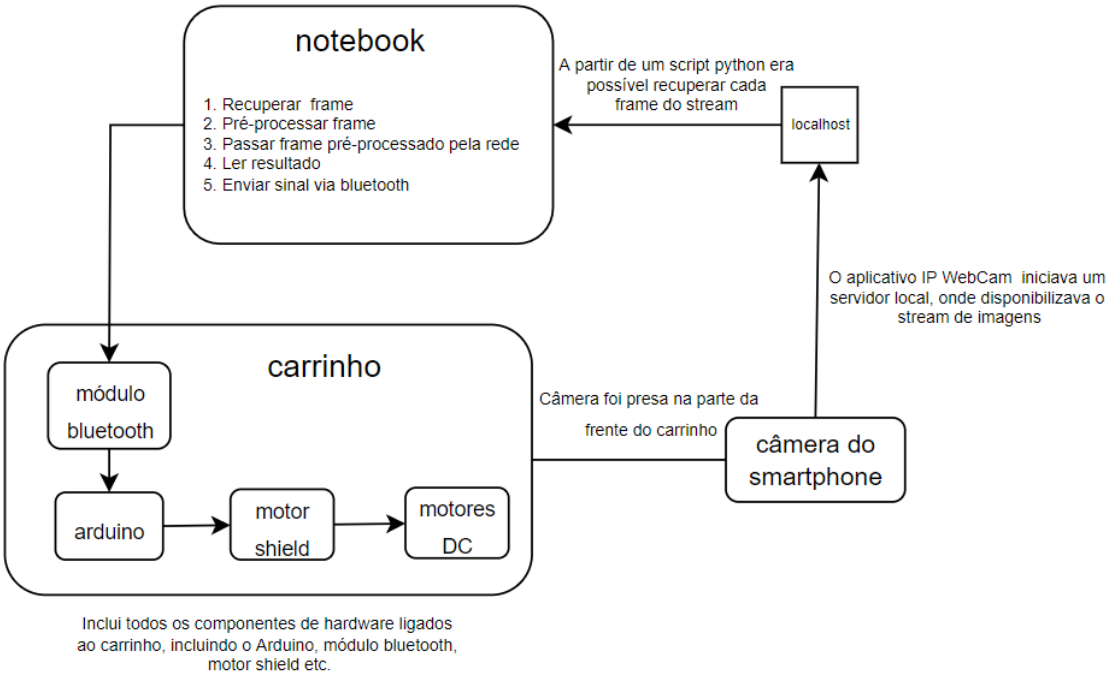



**Figura 22:** processo de funcionamento do sistema distribuído.

Como a maior parte dos modelos criados tratou o problema como sendo de regressão, o *output* para a maioria das redes treinadas era um valor contínuo, podendo ser positivo (direita) ou negativo (esquerda). Quanto mais positivo o valor do *output*, maior deveria ser a curva feita para a direita e portanto mais sinais com o caractere "R" (*right*) deveriam ser enviados ao carrinho via bluetooth. O mesmo valia, ao contrário, para os valores negativos, que indicavam que o carrinho deveria fazer a curva para a esquerda. Por seu turno, valores de *output* próximos a zero significavam que o carrinho

deveria virar muito pouco ou simplesmente seguir em frente. A definição de quando e quanto deveria virar depende de cada teste e é relatada no Capítulo 5.

Já para as redes treinadas para tratar o problema como sendo de classificação, como informado anteriormente, a camada de saída incluía 3 neurônios, contemplando as três possíveis classes: esquerda, centro ou direita. A saída que a rede fornecia nesse caso era um vetor representativo de uma distribuição de probabilidade, contendo valores entre 0 e 1 para cada uma das possíveis classes, de modo que a saída com maior valor representava a classificação feita pela rede.

O *stream* de imagens era capturado e transmitido pelo celular através de um aplicativo chamado *IP Webcam*[13]. Esse aplicativo permite que seja iniciado um servidor, acessível através do *localhost* que exibe o vídeo que está sendo filmado pelo celular em tempo real. Dessa maneira, foi possível recuperar esse *stream* utilizando o método *VideoCapture* do *OpenCV*[14]. Foi possível, portanto, recuperar e consumir *frame* a *frame* o conteúdo do *stream* e consequentemente passar os *frames* pelo modelo para gerar a saída predita.

Como dito, após o treinamento ser realizado na nuvem, o modelo era carregado no computador pessoal do autor, em que também era recebido o *stream* de imagens. As imagens eram então processadas pela rede neural, gerando uma saída que definia qual sinal seria enviado via *bluetooth* para o arduino. Portanto, é possível perceber que o sistema completo se tratava de um sistema distribuído com 3 partes principais: celular, computador e equipamentos de *hardware* relacionados ao arduino (módulo *bluetooth*, *motor shield*, *protoboard*, o próprio arduino, etc).

Foi tomada a decisão de carregar a rede no computador ao invés de carregá-la no arduino por conta da limitação computacional deste. Caso o arduino fosse utilizado para este propósito, teria que ser utilizada uma rede muito simples, a qual talvez não fosse capaz de realizar a tarefa em questão. O problema da solução adotada foi que o tempo tomado por cada iteração, que contemplava o recebimento e processamento da imagem, assim como a geração da saída, acabava aumentada consideravelmente, já que dependia da comunicação entre celular e computador, e computador e arduino. Uma alternativa, para conseguir utilizar o modelo de rede neural em um sistema embarcado, poderia ser adotar o *Rapsberry pi* ao invés do arduino, visto que o primeiro possui maior poder de

[13] https://play.google.com/store/apps/details?id=com.pas.webcam&hl=pt_BR&gl=US
[14] https://opencv.org/

processamento. Entretanto, por questões de limitação financeira, foi decidido que o melhor seria usar o sistema distribuído relatado.

O código desenvolvido na construção do sistema distribuído e usado na realização dos experimentos, a serem apresentados no Capítulo 5, foi disponibilizado no GitHub[15].

### 4.5 *Carrinho*

O carrinho utilizado neste trabalho foi um 4wd, que era formado por quatro motores dc, 1 *arduino UNO*, 1 *motor shield L293D - Driver Ponte H*, 1 módulo *bluetooth RS232 HC-05*, 1 placa *protoboard*, 1 *case* suporte bateria 9v com saída P4 e 1 suporte de 4 pilhas AA canoa para arduino. As Figuras 23 a 27 ilustram os diversos componentes do carrinho.

O arduino funciona como mestre, controlando os motores DC, a partir dos sinais recebidos do módulo *bluetooth*. Este último serve para permitir a conexão via protocolo *bluetooth* entre o carrinho e o computador. Por seu turno, o *motor shield* serve como intermediário para enviar sinais do arduino para os motores DC. Por fim, estes servem para girar as rodinhas, enquanto a *protoboard* servia para realizar as conexões sem a necessidade de solda. A bateria serve para alimentar o arduino, enquanto as 4 pilhas AA serviam para alimentar o *motor shield*. As Figuras 23 e 24 exibem o carrinho utilizado neste trabalho.

[15] https://github.com/DanielCoelho97/TCC

**Figura 23: Foto de frente do carrinho utilizado neste trabalho.**

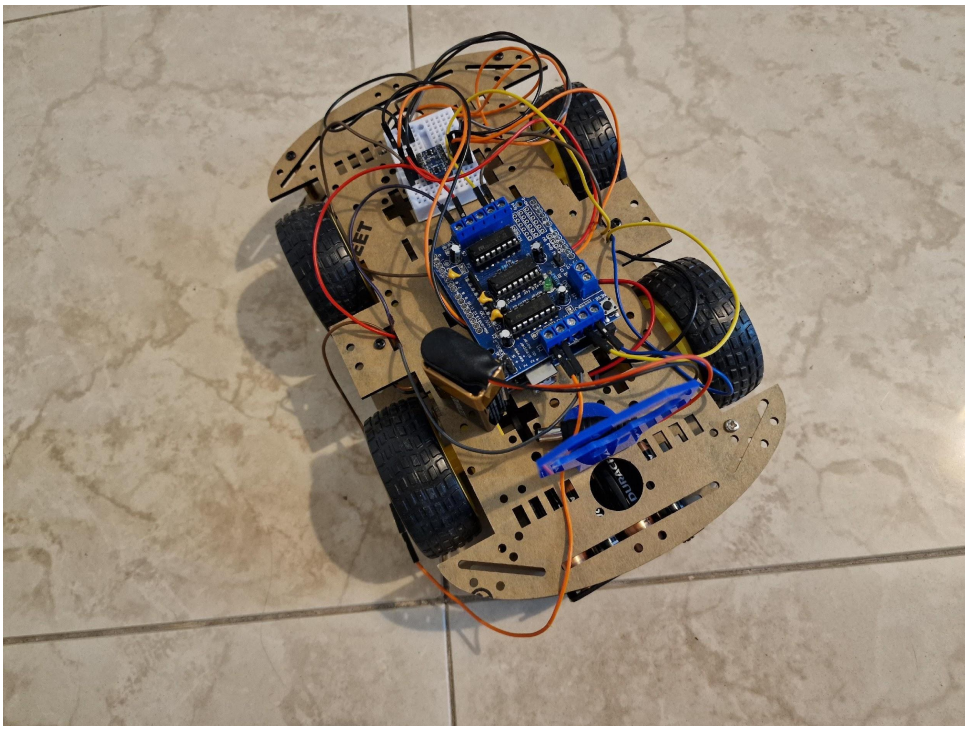


**Figura 24: Foto de cima do carrinho utilizado neste trabalho.**

O sistema distribuído desenvolvido emprega um celular preso ao carrinho fazendo uma transmissão do vídeo sendo capturado, *frame* a *frame*, no formato de *stream*, as quais eram recuperadas em um *script Python* no computador e passadas por uma rede neural no mesmo *script*. Como último passo de tal *script*, um sinal *bluetooth* era enviado ao carrinho contendo a saída da rede, referente ao movimento a ser feito pelo carrinho.

Os detalhes referentes a cada treinamento, como número de épocas, função de custo utilizada, pré-processamento, uso de *data augmentation*, etc. são descritos no Capítulo 5.

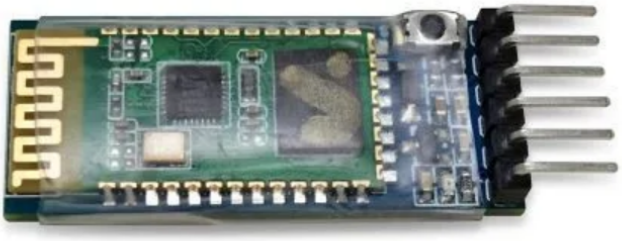


**Figura 25:** módulo *bluetooth* RS232 HC-05 [16].

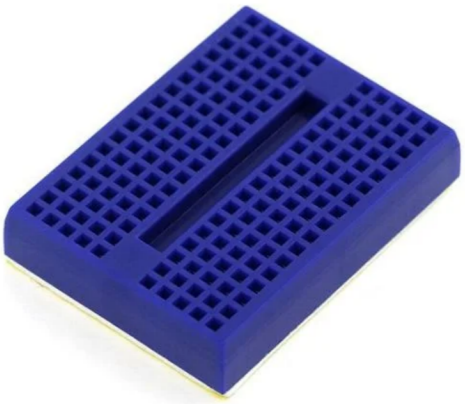


**Figura 26:** protoboard do carrinho[16].

[16] Disponível em: <https://www.arducore.com.br/>. Acesso em 02 de Ago. 2022.

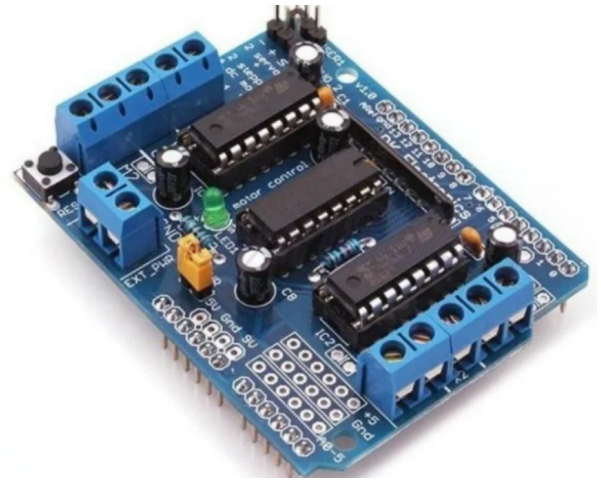



**Figura 27:** *motor shield* L293D - Driver Ponte H [17].

**Figura 28:** *Case* Suporte Bateria 9v com saída P4 [17].

**Figura 29:** Suporte 4 pilhas AA canoa para arduino [17].

# 5 Experimentos

O objetivo deste capítulo é apresentar os detalhes dos treinamentos realizados com a rede neural, como hiperparâmetros utilizados, tipos de pré-processamento, *data augmentation* etc, assim como os resultados obtidos ao testar a rede em um cenário real, utilizando o carrinho.

Foram realizados diversos treinamentos e experimentos para buscar obter a rede ideal (ou pelo menos uma boa o suficiente), cuja validação não viria somente a partir do dataset de validação durante o processo de treinamento, porém também após este processo, ao testar a rede utilizando imagens do cenário real. É importante lembrar que o treinamento foi todo realizado utilizando imagens do simulador, porém o objetivo final era que a rede fosse capaz de predizer imagens da pista do cenário real, as quais seriam a base para o carrinho poder percorrê-la com sucesso.

Os experimentos realizados foram divididos em dois grupos. O primeiro grupo consistiu nos experimentos iniciais, os quais foram feitos sem registro de todos os detalhes de treinamento e testes, servindo mais para um panorama inicial. Note-se que treinamento, neste caso, significa todo o processo de treinamento do modelo de rede neural, considerando treino e validação. Já o teste significa a parte do experimento com o carrinho, ou seja, o teste que verifica se as redes de fato funcionam em um cenário real e levam o caminho a realizar uma trajetória adequada na pista delineada.

Por sua vez, o segundo grupo de experimentos foi realizado contemplando todos os detalhes dos treinamentos e testes, de modo que os dados foram registrados e reunidos em dois Quadros, para facilitar a comparação entre as diversas variações de redes treinadas e testadas.

## 5.1 Primeiro conjunto de experimentos

### 5.1.1 Primeiro Experimento

O primeiro experimento foi conduzido utilizando somente o *dataset* obtido a

partir da primeira pista do simulador *Udacity*. A Figura 30 representa um *frame* dessa pista.


**Figura 30:** Exemplo de *frame* da primeira pista do simulador *Udacity* [17].

O treinamento foi feito adotando 10 épocas. Como função de perda, foi utilizada a de erro médio quadrático e foi utilizado o otimizador Adam, com *learning rate* de $1.0e^{-4}$.

O pré-processamento usado nas imagens incluía um *crop* da imagem, isto é, um recorte, para remover a frente do carrinho e a parte de cima que mostrava o céu e as montanhas (mantendo dessa forma somente a região de interesse). Além disso é feito um *resize* na imagem, transformando-a em 160px por 320px, para ficar do tamanho esperado pelo modelo de rede neural usado. Finalmente, ela é convertida de RGB para YUV. O resultado do pré-processamento, comparado com a imagem inicialmente capturada diretamente do simulador, é apresentado na Figura 31.

[17] Disponível em: <https://ideiasesquecidas.files.wordpress.com/2017/02/screenshot01.jpg>. Acesso em 22 de Ago. 2022.


**Figura 31:** Exemplo de imagem obtida da primeira pista do simulador (cima) e a imagem obtida após o pré-processamento (baixo).

Em relação à divisão entre treino e validação, o **dataset** foi dividido da seguinte maneira: 70% treino e 30%, sem uso de *data augmentation* neste primeiro caso.

Para esse primeiro experimento, a pista real criada era bem simples, de modo que incluía somente uma curva simples. Era uma pista bem básica, criada a partir da utilização de uma fita crepe azul, a qual servia para demarcar as bordas da pista. O experimento foi realizado de modo a testar a capacidade do carrinho de fazer uma curva, tanto para a esquerda quanto para a direita, para verificar se ele se saía bem em ambos os casos. A Figura 32 representa a pista que foi criada.

**Figura 32:** Ilustração da pista utilizada no primeiro experimento.

Como esse tipo de carrinho é mais simples e somente é possível fazer curva girando as rodas de um dos lados (como um tanque de guerra), foi preciso criar um algoritmo que levasse isso em conta. Isso porque se a rede gerasse um *output* de que ele devia virar para a esquerda e fossem enviados sinais para o carrinho virar sem levar essa

limitação em consideração, o carrinho iria girar parado, como um pião. Dessa maneira, o algoritmo criado possui a seguinte lógica: inicialmente a rede gera a saída, indicando se o carrinho deve virar para a esquerda, direita ou seguir reto. Caso deva ser feita uma curva (para direita ou esquerda), são enviados somente alguns sinais para o carrinho girar e depois são enviados alguns sinais para que siga reto. Assim, ele gira um pouco e segue também adiante. Caso a rede determine que o carrinho deva seguir reto, ele somente segue reto e calcula novamente a próxima ação, repetindo sempre então esse mesmo fluxo.

Neste primeiro experimento, o carrinho se saiu bem nas vezes em que teve de fazer a curva para a esquerda, porém ao tentar fazer para a direita o resultado foi bem pior. Em algumas partes, ele entendia corretamente que deveria realizar a curva para a direita, mas a partir de determinado ponto se perdia e virava para a esquerda passando então a perder a pista de vista e ficando então totalmente perdido sem o referencial de trajeto.

Entretanto, ao analisar as imagens que estavam sendo fornecidas como entrada da rede neural, vindas do *stream*, percebeu-se que essas estavam mostrando uma parte considerável acima da pista, não focando somente na região de interesse. Quando essa região acima da pista aparecia, foi possível perceber que a maioria dos *outputs* da rede eram negativos, enquanto quando inclinava-se a câmera para baixo, focando somente na pista, os resultados aparentavam meio aleatórios. Portanto, foi possível perceber que, apesar dos resultados bons obtidos na hora da validação da rede com as imagens do simulador, a rede ainda não havia se tornado capaz de generalizar o suficiente para compreender uma imagem de um cenário real. Outro detalhe percebido neste teste foi que, apesar de negativos ou positivos, os resultados sempre estavam próximos de zero.

Nos relatos dos experimentos a seguir, será somente informado aquilo que houver de mudança em relação ao experimento corrente, de modo que o que não for informado pode-se assumir que foi feito da mesma forma. As mudanças foram graduais, portanto, se for feito, por exemplo, uma mudança de função de custo da rede neural do Experimento 1 para o Experimento 2, essa mudança será informada na seção referente ao “Segundo Experimento”. Entretanto, se essa mudança persistir no experimento 3, nada será informado na seção referente ao “Terceiro Experimento”. Caso se mude de volta, por exemplo, para a função de custo utilizada no primeiro experimento, será então informado novamente. Portanto, pode-se assumir que um teste por *default* possui as mesmas características de seu predecessor.

### 5.1.2 Segundo Experimento

No segundo experimento, ao invés de utilizar somente um *dataset* de treino e validação contendo imagens da primeira pista, foi utilizado um com imagem de ambas as pistas providas pelo simulador da *Udacity*. A Figura 33 representa um *frame* da segunda pista.


**Figura 33:** Representa um frame da segunda pista do simulador *Udacity*.

Novamente o modelo se saiu bem ao tentar predizer o *steering* de imagens do simulador, porém ao tentar prever imagens do cenário real, a maioria das predições era negativa, indicando que o carrinho deveria fazer uma curva à esquerda, independente de pra que lado estivesse a curva. A hipótese levantada é de que a segunda pista provida pelo simulador não é tão interessante assim para treinar o carrinho, pois a marcação das bordas não é tão boa. Portanto, o modelo não foi bem treinado para a função que deveria executar no cenário real, apesar dos bons resultados obtidos nos cenários artificiais. Ademais, nesse caso não se chegou a rodar o carrinho na pista real, somente colocando-o à frente da pista, pra ver como se comportava.

Depois deste teste, tomou-se a decisão de não mais usar imagens referentes à segunda pista, portanto nenhum teste posterior a este usou-as.

### 5.1.3 Terceiro Experimento

Para realizar este experimento, foi gerado novamente um *dataset* através do emprego do simulador, baseado somente em imagens da primeira pista, porém, desta vez, foram registradas imagens percorrendo o carrinho em um sentido e depois no oposto. Isso porque a direção que vem por *default* faz mais curvas para a esquerda, o que levou a suspeitar que poderia estar enviesando o treinamento.

Mais especificamente, foram dadas oito voltas na pista para cada lado, de modo a gerar no total 14 mil imagens a serem utilizadas no treinamento do modelo (isto é, 70% do dataset possuía em torno de 14 mil imagens). Neste experimento, o carrinho também não foi rodado, de modo que foi somente apontado para a pista criada para verificar os resultados que a rede fornecia.

Ao realizar este experimento, foi possível perceber que os resultados gerados pela rede eram muito próximos de zero em sua maioria, o que levou à hipótese de que talvez a quantidade de imagens usadas no treinamento com *steering* zero, isto é, com ângulo zero que representa o fato de que o carrinho deve seguir em frente, pudesse estar enviesando a rede, influenciando-a a gerar *outputs* sempre muito próximos de zero. Essa hipótese foi confirmada através da verificação de que a quantidade de imagens que possuíam *steering* zero representava 80% do *dataset* de treino.

### 5.1.4 Quarto Experimento

Para o quarto experimento, a única modificação feita foi que a porcentagem relativa de imagens relacionadas ao *steering* zero no conjunto de treino foi diminuída de 80% para somente 10%. Ainda assim, os resultados não foram muito satisfatórios, já que a rede continuava sem conseguir predizer a direção de forma consistente, a partir das imagens do cenário real.

### 5.1.5 Quinto Teste

Os resultados obtidos no experimento anterior levaram à alteração do algoritmo de pré-processamento a partir deste quinto experimento. O *crop* e o *resize* se

mantiveram, porém em vez da conversão de RGB para YUV passou a ser feita a conversão de RGB para *GrayScale*, com o objetivo de diminuir o tempo de resposta da rede, usando uma imagem de um único canal ao invés de três.

Essa alteração no pré-processamento das imagens impactou na definição da camada de entrada da rede neural, visto que as imagens possuem agora apenas um canal. Além disso, passou a ser utilizado o *GaussianBlur* (método do *OpenCV* o qual ajuda a reduzir ruídos da imagem, adicionando um blur à imagem), passou a ser utilizado o método *canny* do *OpenCV*, o qual demarca as partes da imagens que possuem uma variação brusca de cor. A intenção era ajudar a marcar as bordas da pista. Entretanto, foi possível perceber que as imagens acabam apresentando diversos "ruídos", como é mostrado nas marcações em vermelho da Figura 34.

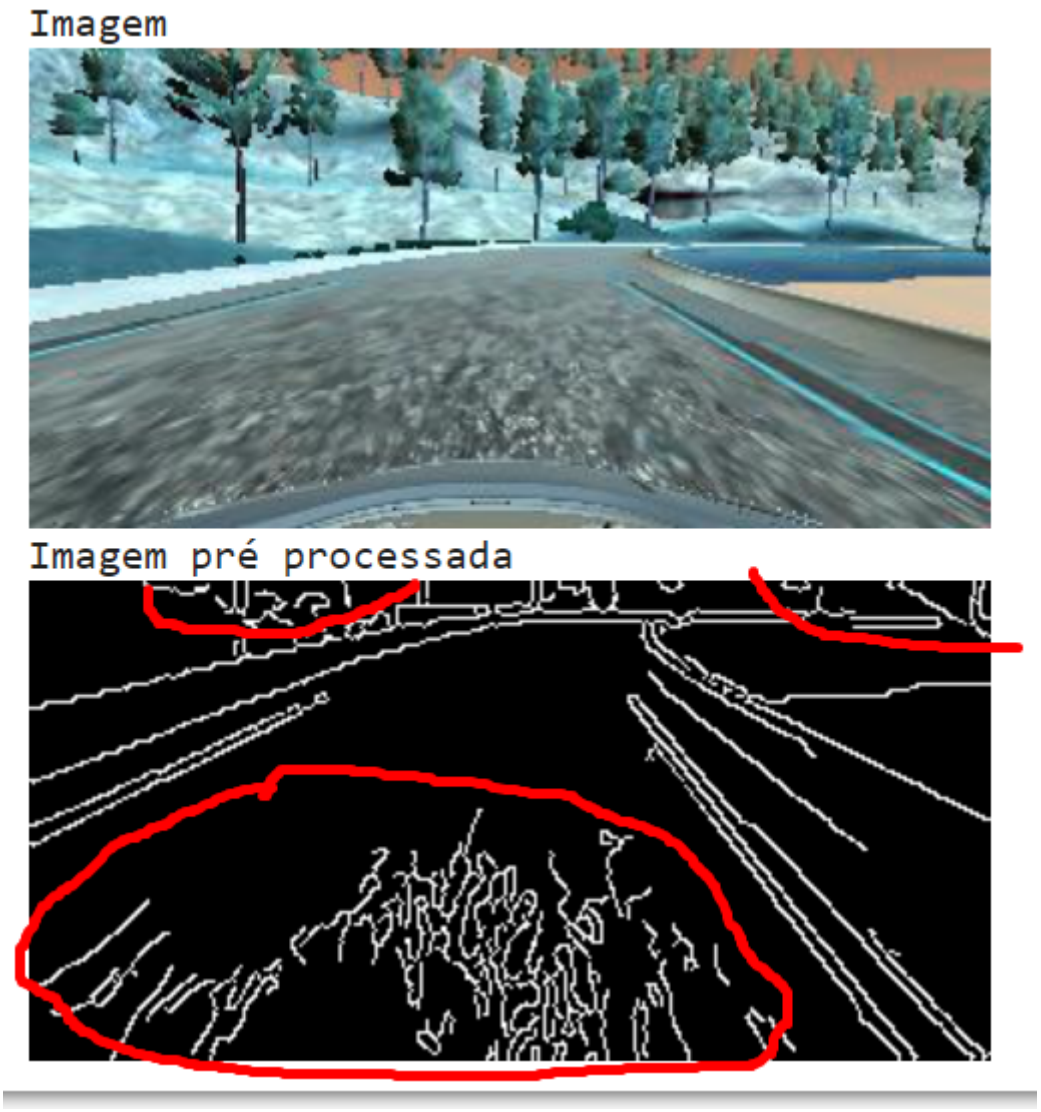



**Figura 34:** Exemplo de imagem obtida do simulador (cima) e a imagem obtida após o novo pré-processamento (baixo), em que houve foram marcados em vermelho as partes que representam os "ruídos".

Para diminuir tais "ruídos", foi utilizada uma funcionalidade de máscara do *OpenCV*, que preenche com preto as partes indesejadas. A Figura 35 ilustra a mesma imagem pré-processada, dessa vez com o uso da máscara. É possível perceber que ainda se mantiveram alguns "ruídos", em comparação à Figura 34, porém a maioria foi eliminada.

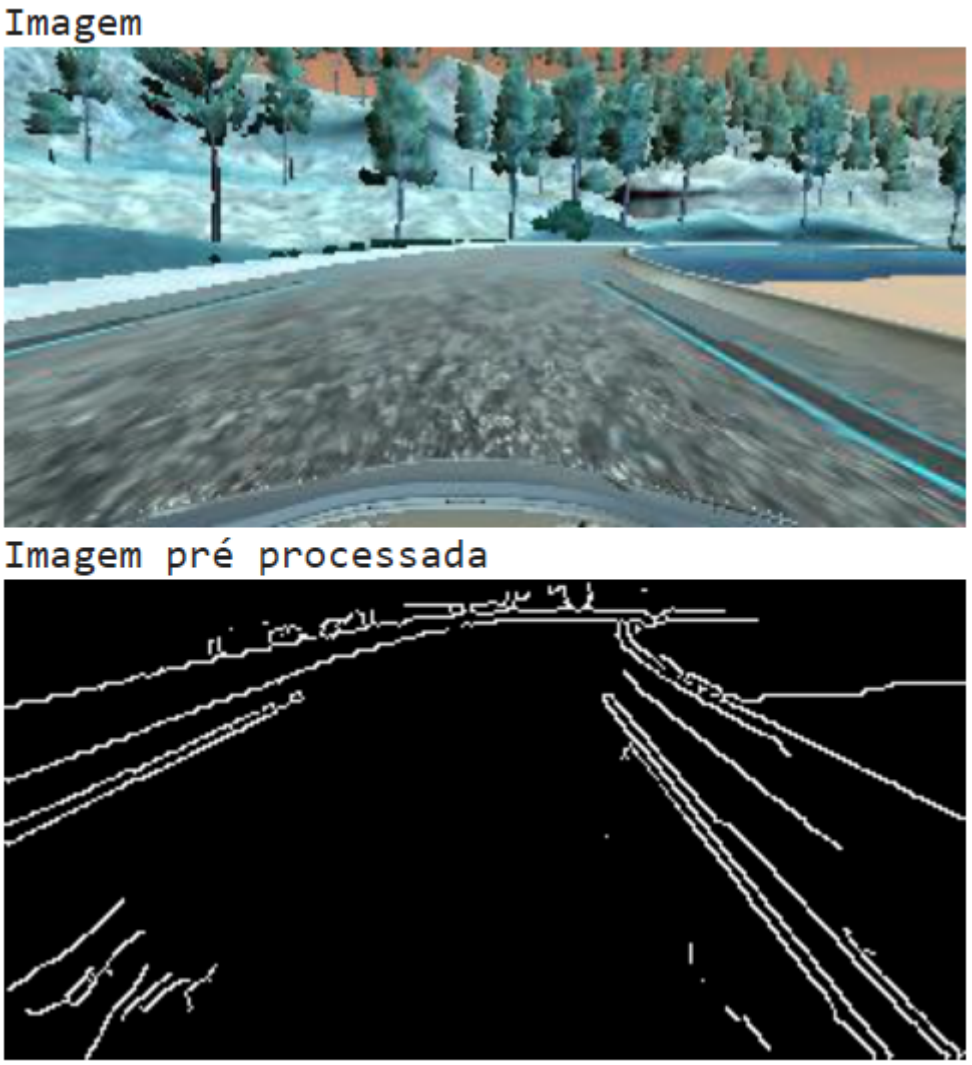



**Figura 35:** Exemplo de imagem obtida do simulador (cima) e a imagem obtida após o novo pré-processamento (baixo), dessa vez com o uso da máscara.

Para este experimento, a menor perda para o conjunto de validação foi conseguida na última época (época 10) e ficou em 0,0430, tendo começado em um valor perto de 0,1, o que pode ser considerado uma melhora significativa. Exemplo de dois resultados para o conjunto de validação, isto é, após o treinamento da rede, que foram promissores podem ser vistos na Figura 36.

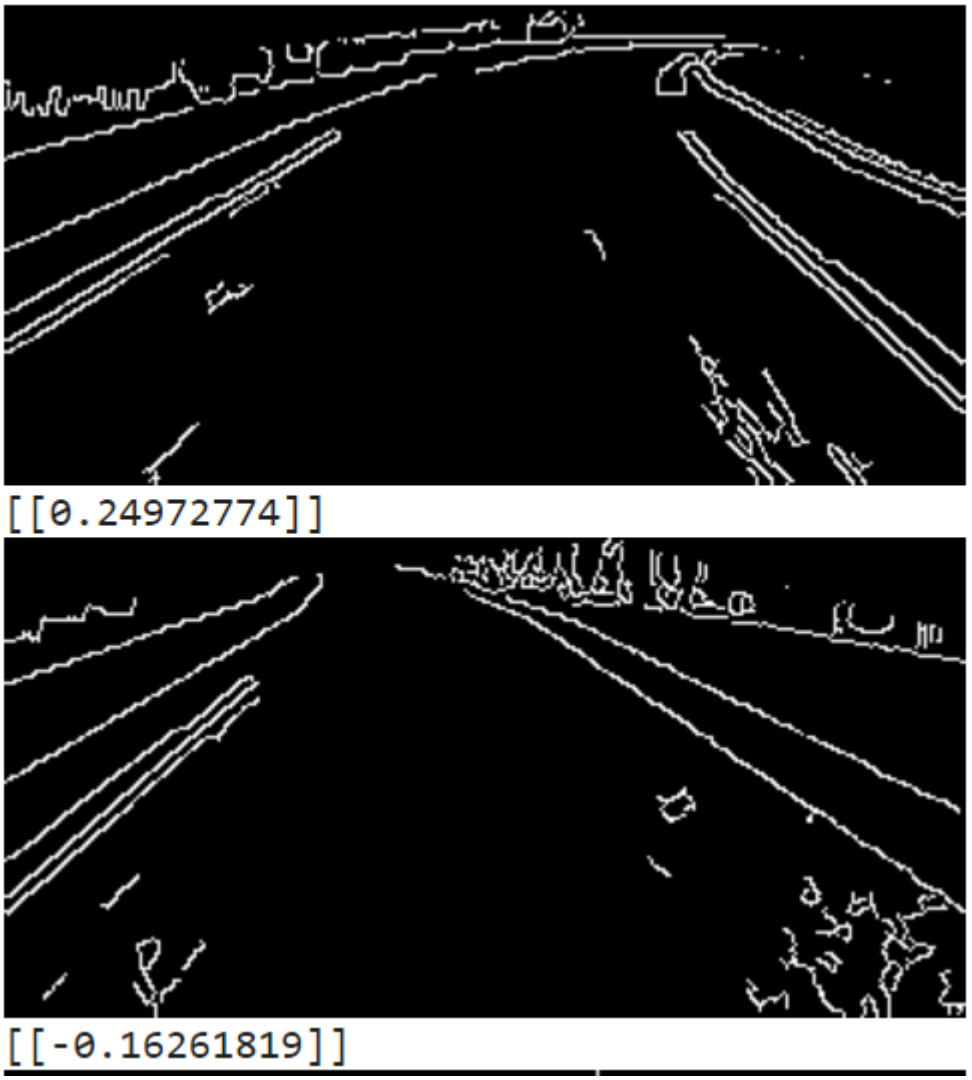



**Figura 36:** Exemplos de duas imagens apresentadas à rede no conjunto de validação, com os respectivos *outputs* da rede discriminados.

É possível ver que os *outputs* são coerentes de acordo com a curva que deve ser efetuada em cada uma das imagens. A imagem de cima claramente representa uma curva para a direita, de modo que o *output* gerado pela rede foi de 0,24, enquanto a de baixo, que representa uma curva um pouco menor para a esquerda, possui um *output* de - 0,16. Entretanto, esses resultados coerentes também já estavam sendo fornecidos pelas redes dos experimentos anteriores. Neste ponto, era preciso obter um resultado coerente também para os casos em que a rede estivesse recebendo como *input* imagens do cenário real.

Portanto, o próximo passo foi testar na prática para ver se a rede conseguia alcançar bons resultados. Ao rodar primeiramente o *stream* de imagens, antes de colocar o carrinho para rodar, foi possível perceber que a marcação das bordas da pista não

estava aparecendo direito. As Figuras 37 e 38 demonstram esses fenômeno. Ademais, a Figura 39 exibe a visão de cima tanto do carrinho quanto da pista.

**Figura 37:** Imagem capturada pelo celular acoplado ao carrinho, que foi transmitida via *streaming* para a rede neural.

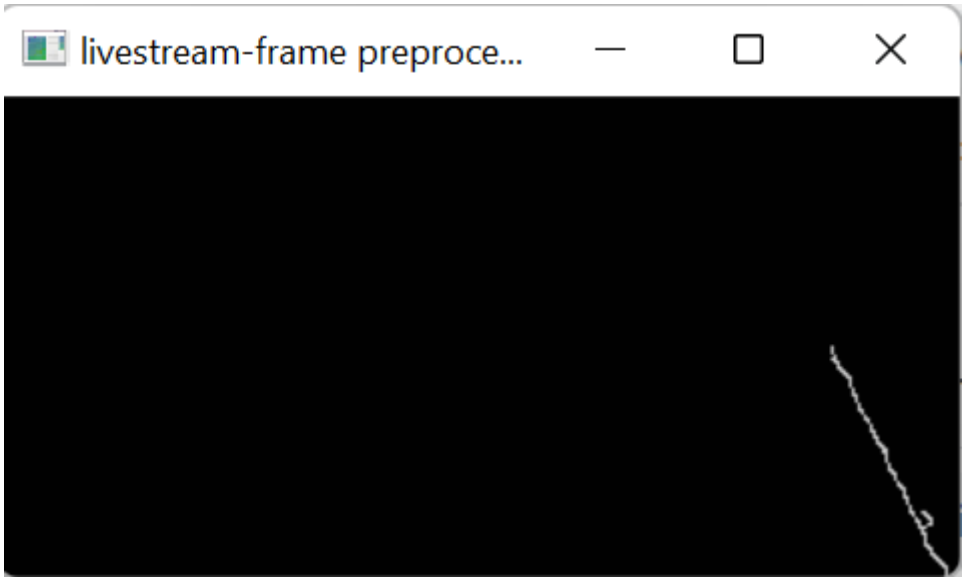



**Figura 38:** Resultado do pré-processamento aplicado à imagem da Figura 37.

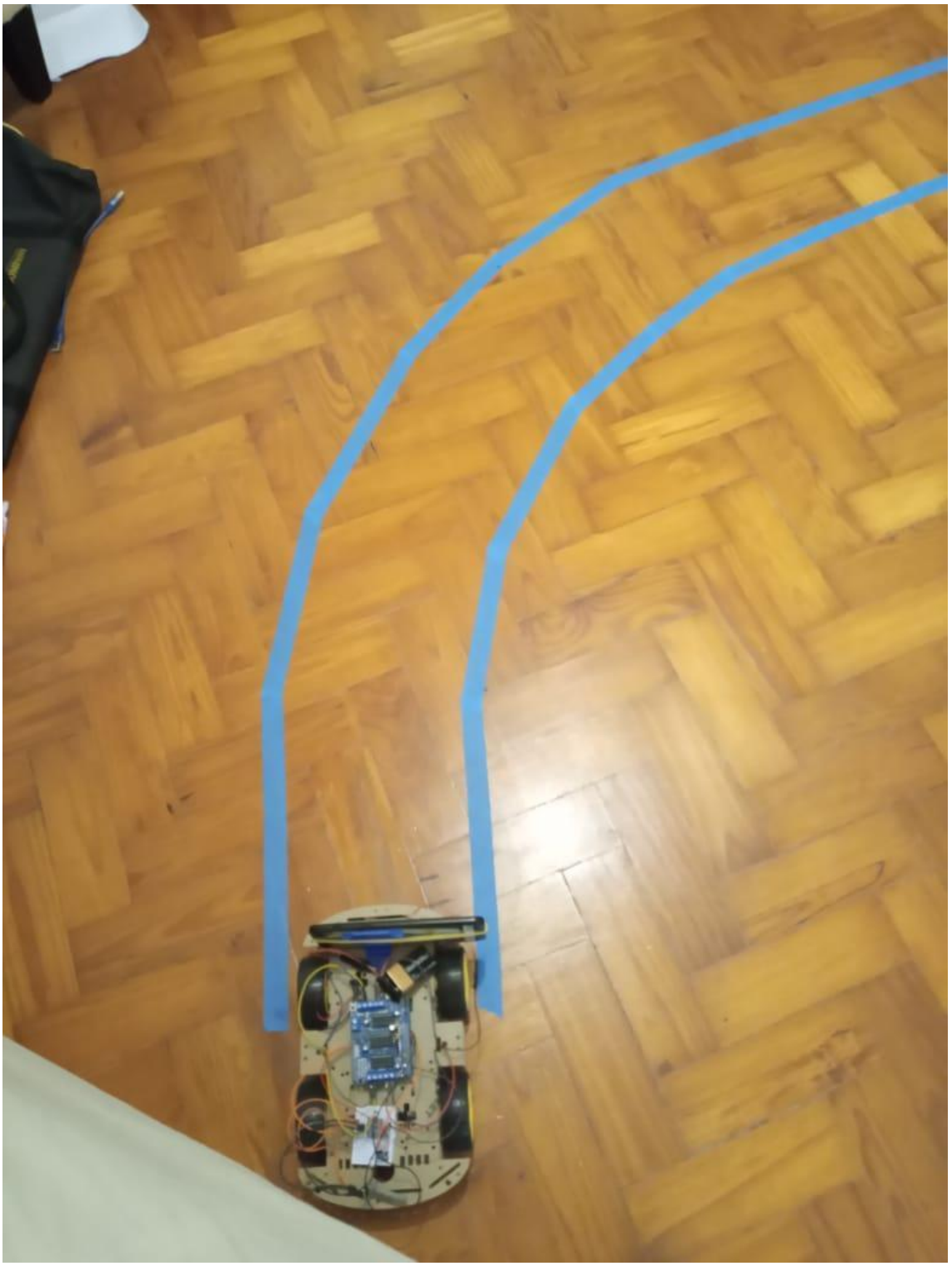


**Figura 39:** Visão de cima do carrinho e da pista utilizada.

Esse fato levou a crer que talvez o mesmo problema estivesse ocorrendo com os modelos dos experimentos anteriores. Talvez a marcação da pista, dentro de um cenário com um piso que não era totalmente liso e de uma cor só e com outros objetos aparecendo, pudesse estar confundindo a rede de modo que esta não compreendesse

sempre o que exatamente fazia parte da pista. sendo portanto um problema do pré-processamento e não da rede. Poderia também, possivelmente, ser um problema de iluminação do ambiente, de modo que a iluminação ou a falta desta prejudicasse a imagem, tornando a marcação da pista menos nítida. Claro que normalmente esses fatores não deveriam influenciar, já que um carro real autônomo deveria conseguir funcionar independente do tempo, local, etc. Porém, para um modelo mais simples, construído com um investimento infinitamente menor, essas interferências poderiam ser cruciais.

Isso motivou a fazer o seguinte teste: colocar uma folha de papel na frente do carrinho, de modo a tentar acentuar a diferença de cor entre a pista e as bordas. As Figuras 40 e 41 demonstram o teste e seu resultado, respectivamente.

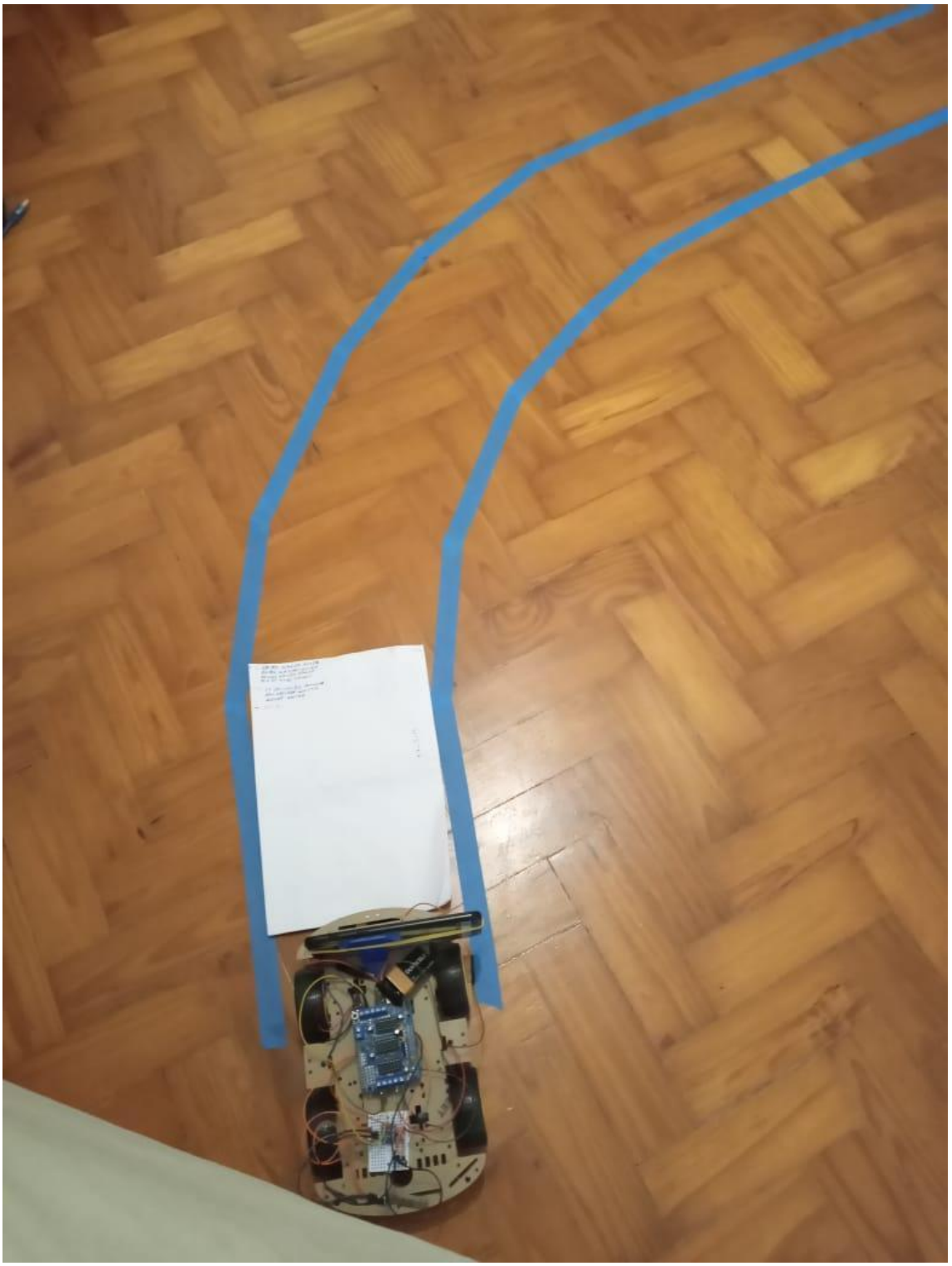


**Figura 40:** Ilustração do cenário de teste, com a folha de papel para atenuar a diferença na iluminação.

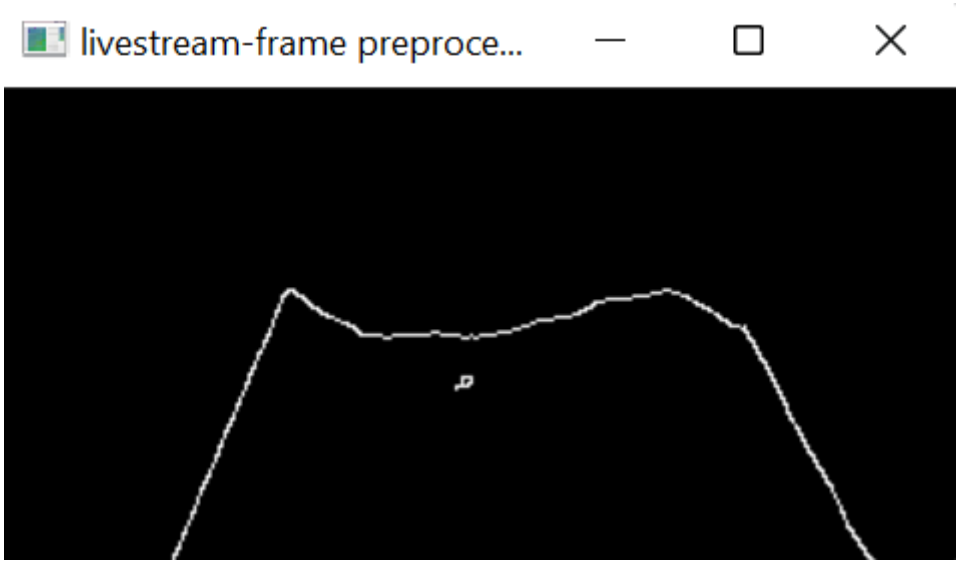



**Figura 41:** Resultado do pré-processamento da imagem da Figura 40.

A partir do teste, não foi possível concluir se a marcação se deu somente devido ao contraste do papel com o chão ou se a borda em azul também estava ajudando.

Outro teste feito foi colocar uma lanterna em cima da pista, dessa vez sem utilizar nenhum papel, para verificar se com auxílio de iluminação o contraste entre a pista e o chão poderia se tornar mais nítido na imagem pré-processada. O resultado pode ser visto nas Figuras 42 e 43.

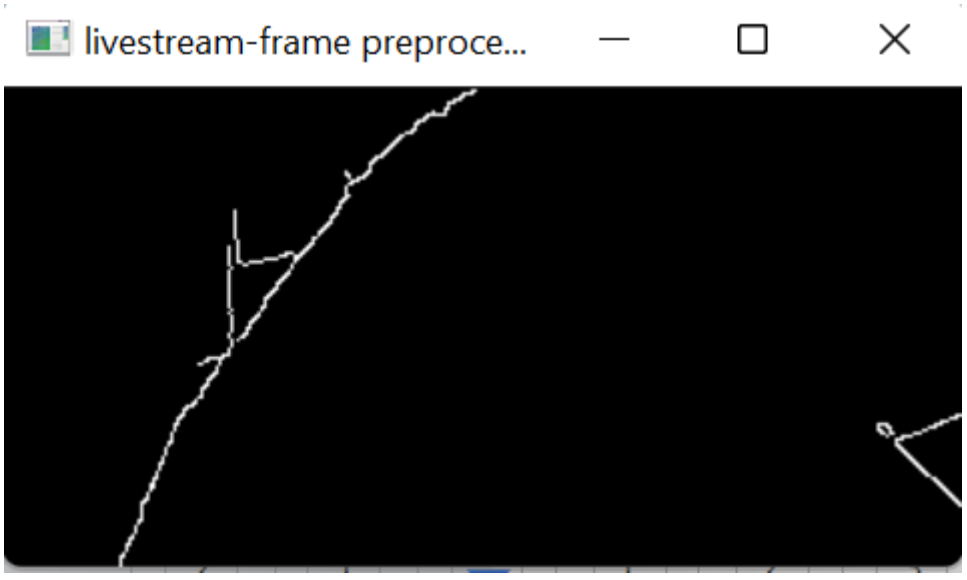



**Figura 42:** Resultado do pré-processamento de uma imagem com iluminação em cima.


**Figura 43:** Resultado de pré-processamento de uma imagem com iluminação em cima.

A partir das Figuras 42 e 43, é possível ver que a marcação da pista melhorou bastante, porém outro problema foi que as irregularidades do piso também estavam sendo detectadas e aparecendo na imagem, o que poderia acabar atrapalhando.

Neste ponto, em vez de marcar a pista diretamente no chão, se resolveu usar um tapete invertido como base e fazer a marcação em cima deste. Isso porque o tapete não possuía irregularidades tão marcantes como o piso e possuía uma cor mais clara, de modo que na teoria facilitaria identificar a marcação da pista em contraste com o chão. A Figura 44 demonstra como ficou a pista, a 45 mostra a visão da pista da perspectiva do carrinho e a 46 demonstra como ficou a imagem pré-processada.

**Figura 44:** Visão de cima da pista e do carrinho em cima do tapete.

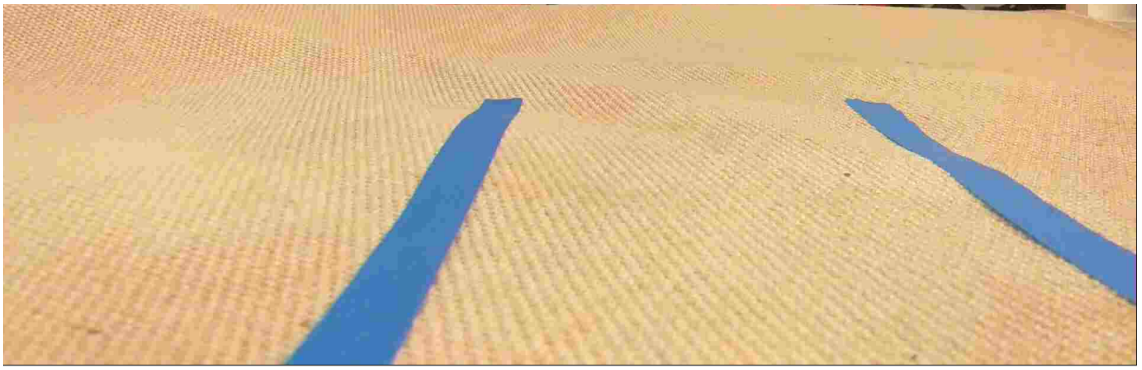


**Figura 45:** Imagem na perspectiva do carrinho.

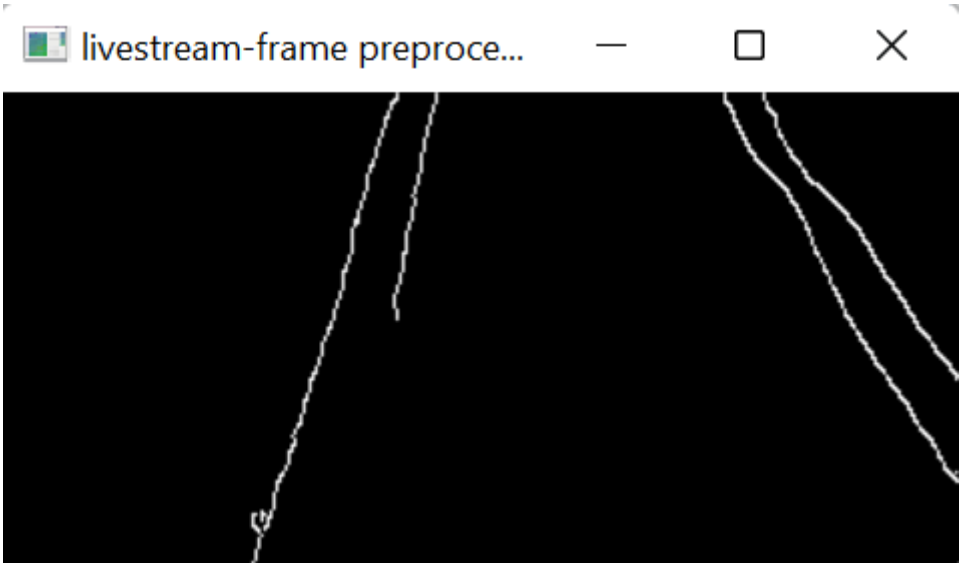



**Figura 46:** Resultado do pré-processamento da imagem da Figura 45.

É possível ver então que, dessa maneira, a identificação da pista na imagem pré-processada melhorou muito. A Figura 47 demonstra a pista que foi então criada para realizar o teste com o carrinho.

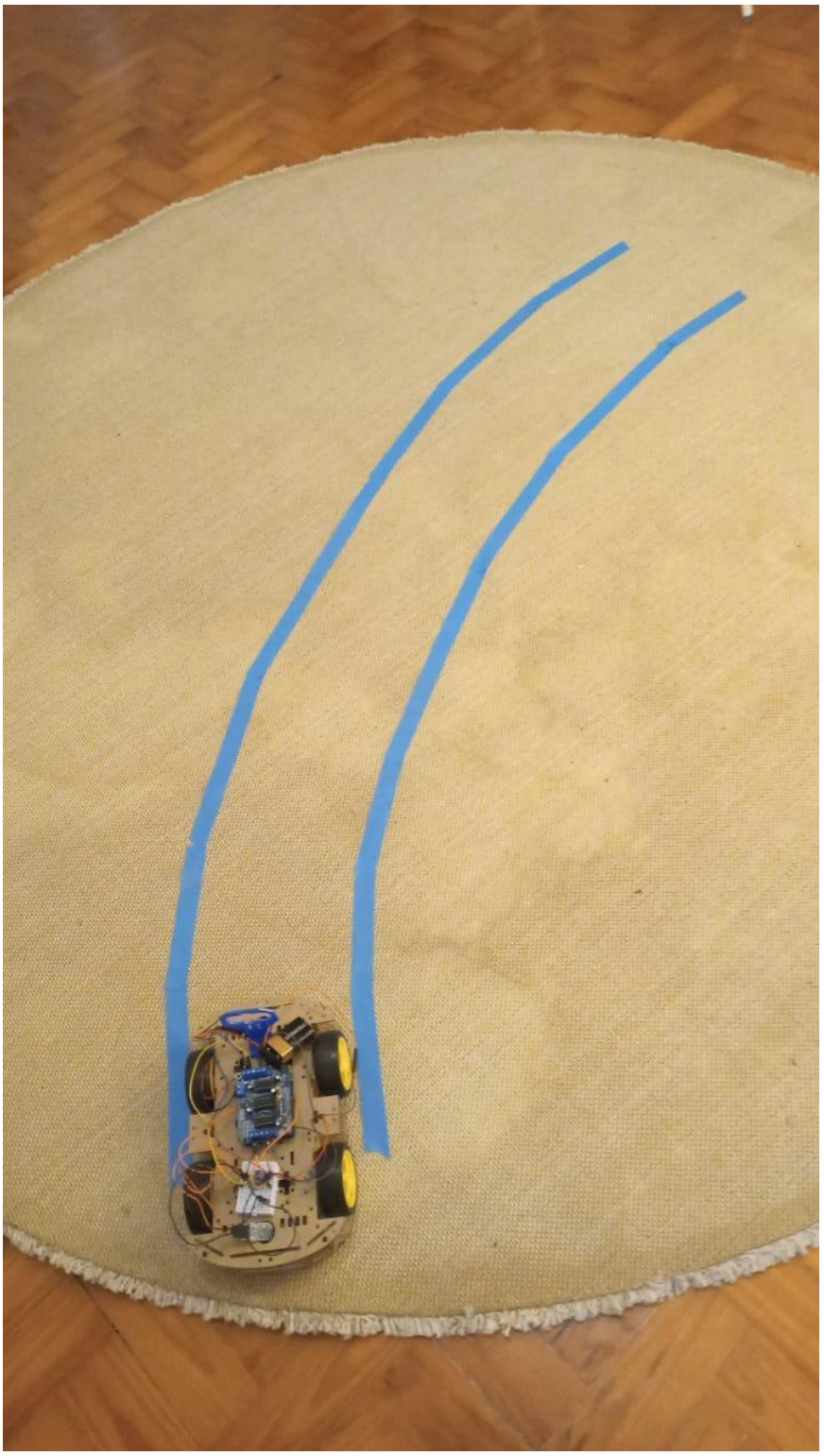


**Figura 47:** Visão de cima da pista criada sobre o tapete.

Ainda assim, os resultados obtidos durante este experimento com o carrinho não foram bons, o que levou a crer que o modelo precisava, mesmo assim, ficar ainda mais robusto.

### 5.1.6 Sexto Experimento

Neste sexto experimento, a quantidade de épocas no treinamento foi aumentada de 10 para 50. Além disso, passou a ser utilizado *Data Generator*, através da adição do sufixo DA em algumas imagens, cuja descrição foi feita na Seção 2.18, de modo que, ao serem montados os *batches* de treinamento, havia uma probabilidade de 25% de ser aplicado *data augmentation* em cima de uma imagem escolhida aleatoriamente. Foram aumentados então, artificialmente, o número de exemplos de treinamento. Como consequência disto, a quantidade de *steps per epoch* aumentou para 126. No Apêndice A, são mostradas as épocas em que a perda de validação (*validation loss*) diminuiu para esse experimento (melhorando assim a qualidade da rede).

Além disso, a função de pré-processamento foi levemente alterada. Além de fazer tudo que já estava sendo feito, passou a gerar linhas artificiais, de acordo com as bordas encontradas, e colocá-las sobre a imagem. A Figura 48 ilustra isso.

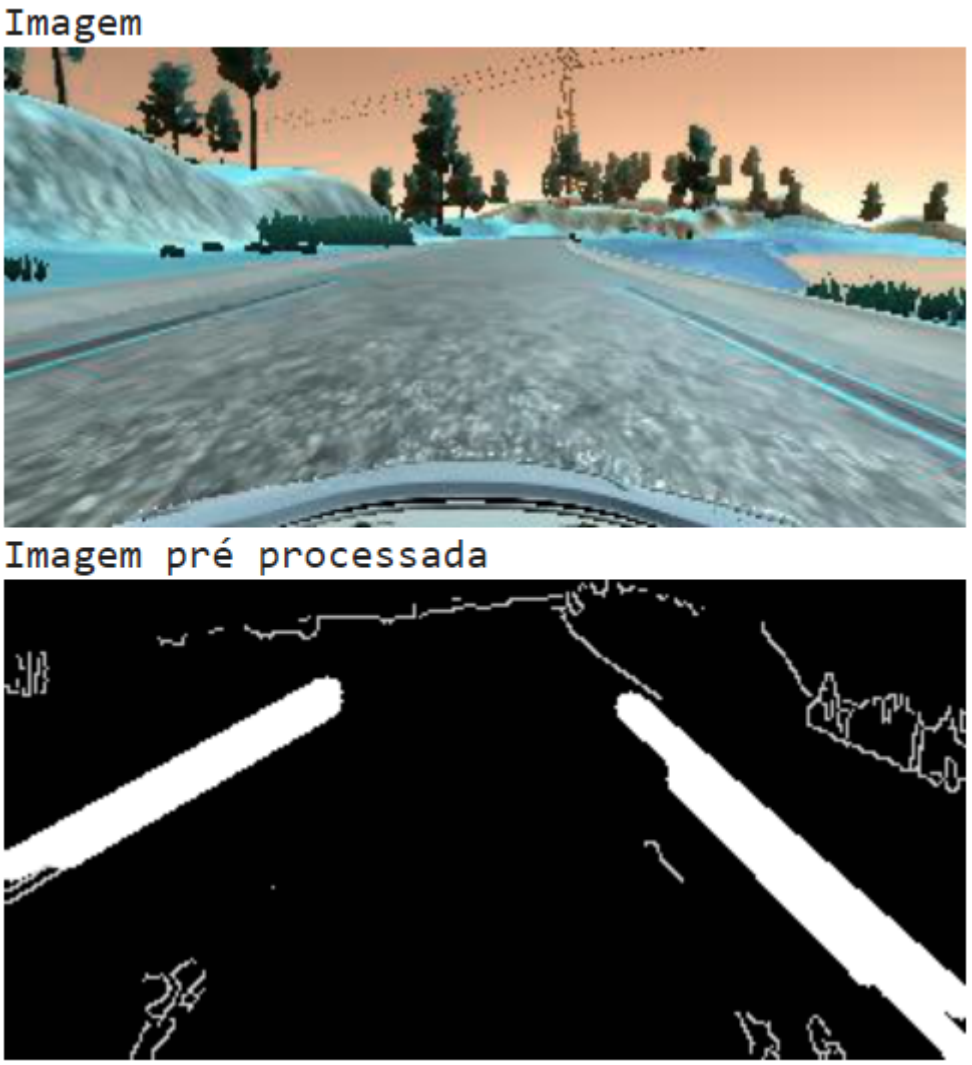



**Figura 48:** Exemplo de imagem obtida do simulador (cima) e a imagem obtida após o pré-processamento (baixo), em que as linhas artificiais correspondem às linhas brancas grossas à direita e à esquerda.

O objetivo dessa estratégia foi tentar aproximar o resultado do pré-processamento das imagens oriundas do simulador das imagens pré-processadas oriundas da pista do cenário real, buscando assim facilitar o processamento da rede.

Novamente foi utilizada a pista sobre o tapete. As Figuras 49 e 50 demonstram, respectivamente, a imagem do cenário real na perspectiva do carrinho e a imagem pré-processada correspondente. No Apêndice B, são apresentados 10 *outputs* dessa mesma imagem, assim como a média aritméticas deles.

**Figura 49:** Imagem na perspectiva do carrinho.

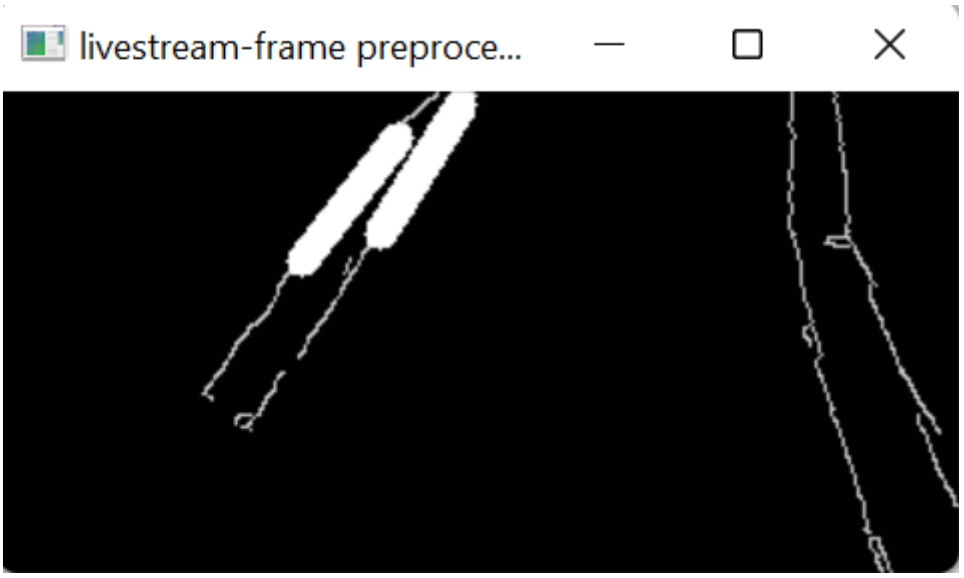



**Figura 50:** Resultado do pré-processamento da imagem da Figura 49.

É possível ver então que, no caso dessa imagem específica, o resultado de *output* do modelo estava sendo coerente, já que era positivo e alto, indicando que o carrinho deveria virar à direita, à medida que a curva da pista estava, de fato, indo para a direita.

O carrinho foi então colocado na outra ponta da pista e as Figuras 51 e 52 ilustram a imagem na perspectiva do carrinho e o resultado do seu pré-processamento.

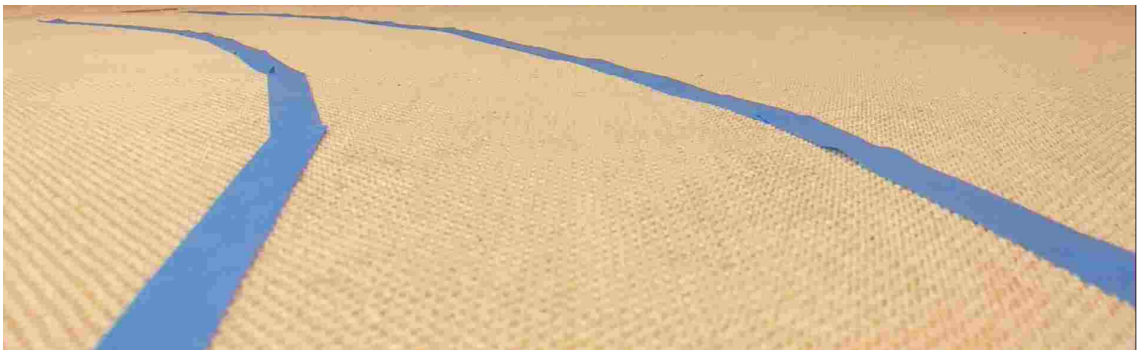


**Figura 51:** Imagem na perspectiva do carrinho.

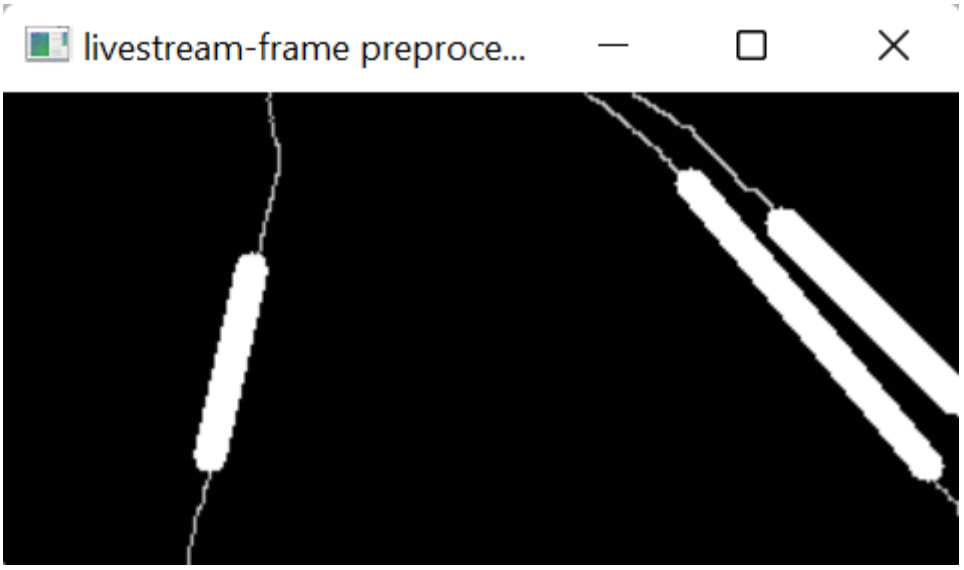



**Figura 52:** Resultado do pré-processamento da imagem da Figura 51.

Assim sendo, foram coletados 10 *outputs* da rede para a imagem da Figura 52 como entrada e o resultado pode ser visto no Apêndice C. É possível ver que, dentre os 10 valores de saída, há apenas um que possui valor negativo, que está associado à curva para a esquerda, e os demais estão positivos. Isso pode indicar que a curva para a esquerda estava tão suave que o modelo estava resultando uma saída próxima de zero, na verdade considerando que ele deveria seguir em frente.

Essa suspeita foi aparentemente confirmada ao mover o carrinho um pouco para frente, na mesma configuração, de modo que a curva para a esquerda passasse a ficar mais evidente. As Figuras 53 e 54 ilustram a imagem na perspectiva do carrinho e o resultado do seu pré-processamento.

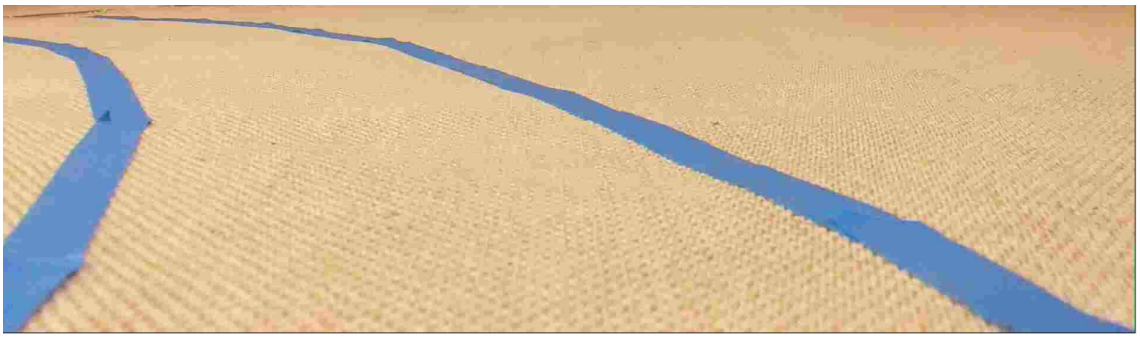


**Figura 53:** Imagem na perspectiva do carrinho.

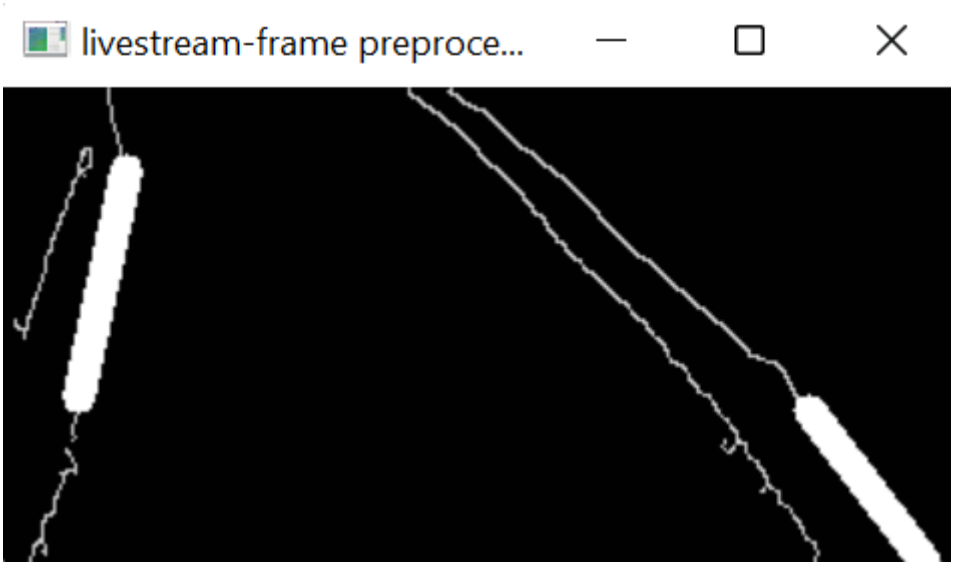



**Figura 54:** Resultado do pré-processamento da imagem da Figura 53.

No Apêndice D, são apresentados 10 *outputs* da rede para a imagem da Figura 54 como entrada e é possível perceber que, dessa vez, os resultados já foram todos negativos e bem mais distantes de zero, indicando que o carrinho deveria seguir para a esquerda.

O carrinho foi então colocado para rodar, testando-o tanto para a esquerda quanto para a direita. Quando foi realizar a curva para a esquerda ele se saiu bem, realizando quase perfeitamente a curva. Já para a direita, na primeira vez ele não se saiu muito bem. Começou fazendo corretamente a curva para a direita, porém um pouco depois acabou interpretando de maneira equivocada que deveria seguir para a esquerda e acabou perdendo a pista de vista. Analisando então as imagens que estavam chegando à rede, nessa parte em que o carrinho estava se perdendo, foi possível perceber que a iluminação não estava adequada, fazendo com que as bordas da pista não aparecessem direito em alguns pontos, como mostra a Figura 56.

**Figura 55:** Imagem na perspectiva do carrinho, ilustrando a baixa iluminação para essa configuração.

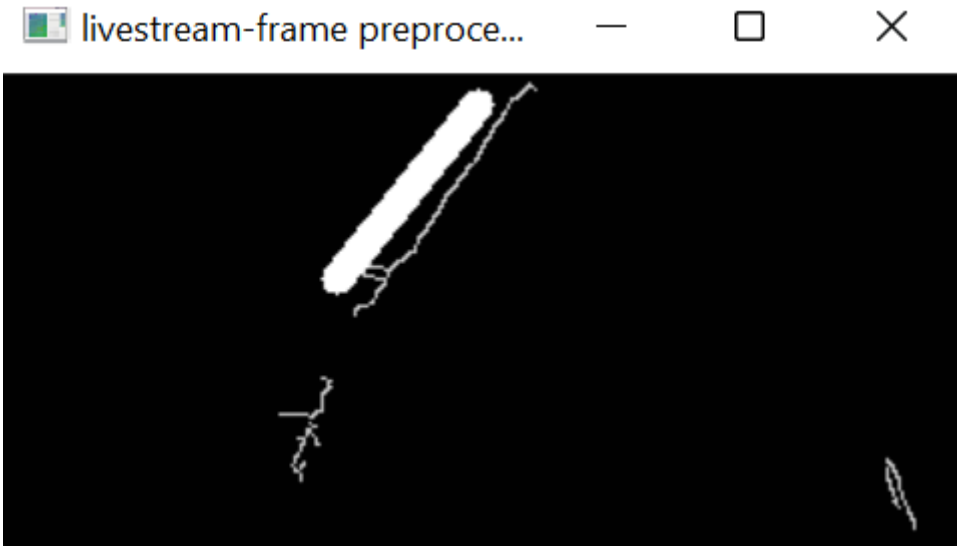



**Figura 56:** Resultado do pré-processamento da imagem da Figura 55.

Dessa maneira, sem as bordas precisamente detectadas pelo pré-processamento, que serve como entrada do modelo, esse ficava perdido, às vezes predizendo um valor negativo, às vezes predizendo um valor positivo.

Foi realizado um outro experimento e dessa vez foi obtido um resultado bem melhor. Pelo que foi possível perceber, em alguns pontos da pista as bordas eram melhor identificadas, em outros, pior. A iluminação do ambiente não era ideal, e a diferença de cores entre as bordas da pista e o tapete não era tão grande, ainda mais em um ambiente mais escuro em que as tonalidades do tapete e fita acabavam parecidas, de modo que dificultava por vezes a detecção das bordas. Isso acabava fazendo com que em algumas partes da pista o carrinho pudesse ficar “cego”, de maneira que as imagens que estavam alimentando a rede nesses momentos não eram boas o suficiente.

## 5.2 Segundo Conjunto de Experimentos

No segundo conjunto de experimentos, foram treinadas duas redes tratando o problema como sendo de classificação e seis redes abordando-o como um problema de regressão. O pré-processamento tanto das redes de classificação, quanto de regressão incluía o *crop* da imagem, com objetivo de remover a parte de cima que não continha a pista e a parte de baixo na qual aparecia à frente do carro, mantendo, dessa forma, somente a região de interesse. Após isso era realizado o *resize* da imagem, para ficar do tamanho certo para servir de *input* da rede (160 X 320).

Depois a imagem era convertida de RGB para *GrayScale* (tons de cinza), era feito o *threshold* da imagem para ajudar a remover ruídos, variações de iluminação, etc. e era utilizado o *Gaussian Blur*, o qual também ajudava na remoção de ruídos. Também era utilizado o método *canny* do *OpenCV*, o qual serve para marcar as bordas da imagem, como foi explicado anteriormente, o que ajudava a marcar as bordas da pista, deixando o resto da imagem toda preta (tirando outras bordas que porventura também fossem encontradas).

A imagem também passava por um método chamado *region_of_interest*, o qual passava um filtro pela imagem, removendo possíveis ruídos do centro, como foi mostrado anteriormente, e partes dos cantos superiores, as quais em geral não eram relevantes, pois não faziam parte da pista.

Por fim, em alguns casos, foi utilizado o método *HoughLinesP*, do *OpenCV*, o qual servia para detectar linhas retas presentes em uma imagem, armazenando-as em um vetor, o qual posteriormente foi usado para "desenhar" sobre a imagem, para marcar estas linhas com mais intensidade. Os Quadros informam os casos em que estas linhas foram adicionadas ao pré-processamento.

No caso do segundo grupo de experimentos, o método que iterava sobre o vetor de linhas e as desenhava sobre a imagem passou a utilizar somente uma linha da parte esquerda e uma da direita. A Figura 57 ilustra uma imagem do *dataset*, obtida a partir da pista do simulador, enquanto a 58 demonstra a imagem pré-processada sem a adição de linhas, ao passo que a 59 a exibe com a adição das linhas.

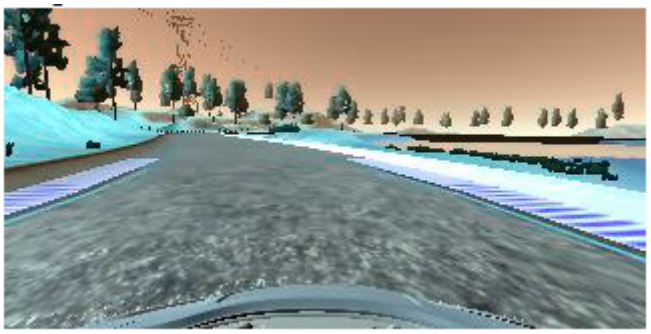


**Figura 57:** Imagem do *dataset* de treino.

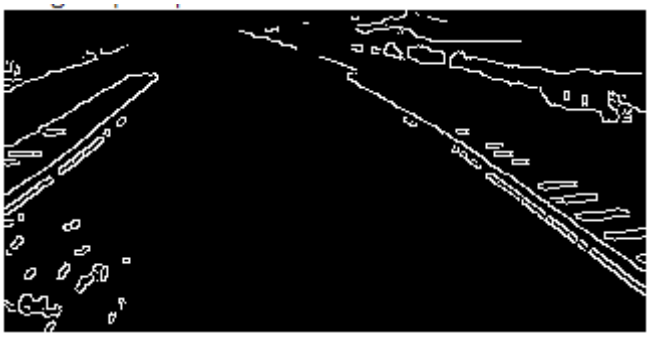


**Figura 58:** Imagem da Figura 57 pré-processada sem as linhas.

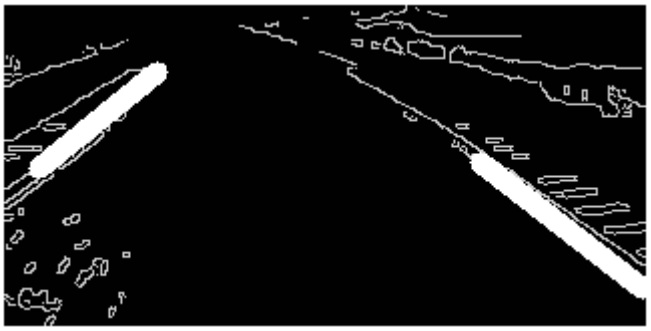


**Figura 59:** Imagem da Figura 57 pré-processada com as linhas.

Para todas as redes treinadas foi utilizado *data augmentation*, utilizando a funcionalidade do *data generator*, e o intervalo da operação de translação horizontal. Para realizar o *data augmentation*, são utilizadas 2 funções: uma para possivelmente (50% de chance) realizar um *flip* horizontal e outra para realizar uma translação horizontal. As outras funções que haviam sido utilizadas no primeiro grupo de experimentos foram removidas, pois foi notado que estavam, em alguns casos, atrapalhando muito na qualidade das imagens e, desta forma, prejudicando o treinamento. Somente era feito o *flip* e translação horizontal, pois o vertical,

principalmente para o caso do *flip*, faria a imagem perder o sentido. Afinal, não faz sentido trocar a pista de lugar com o céu. As variações da quantidade de DA's e do intervalo da translação horizontal serão apresentadas mais adiante.

Os Quadros 2 e 3 a seguir apresentam informações sobre as redes treinadas utilizando classificação e regressão, respectivamente. A coluna "N." representa o número da rede, "Épocas", a quantidade de épocas, "Função de Perda", a função de perda utilizada, "Otimizador", a função de otimização utilizada, "DA's", a quantidade de *strings* DA inseridas no *array* de *paths* do *dataset* de treinamento, "Pré-pro", se o pré-processamento incluía as linhas ou não, "*Data Aug*", qual o intervalo utilizado na etapa de translação horizontal do *data augmentation*, "*Batch Size*", o tamanho do batch, "U. E. M", a última época em que houve melhora da rede. As colunas "Perda T", "Perda V", "Acurácia T" e "Acurácia V" apresentavam 4 informações no seguinte formato: <perda/acurácia da primeira época> - <perda/acurácia da última época que a rede obteve melhora> <setinha para representar se a métrica abaixou ou aumentou>. O "T" significa treino e "V" validação.

A seguir é mostrado um exemplo de possível valor da coluna "Perda T": 1.073 - 0.63 ↓. Neste caso, isso deveria ser lido da seguinte forma: a perda em relação ao conjunto de validação de treinamento da primeira época foi 1.073, da última época que a rede obteve melhora foi 0.63 e portanto houve uma redução desta métrica. As colunas de acurácia só existem no quadro de redes de classificação, pois esta métrica só é usada neste caso.

**Quadro 2:** Principais informações das redes de classificação treinadas no segundo conjunto de experimentos.

| N. | Épocas | Função de perda | Otimizador | DA's | Pré-proc | Data Aug | Batch Size | U. E. M. | Perda T | Perda V | Acurácia T | Acurácia V |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 200 | categorical-crossentropy | ADAM | 5.000 | com linhas | range 50 | 64 | 2 | 1.073 - 0.63 ↓ | 1.039 - 1.73 ↑ | 0.42 - 0.41 ↓ | 0.41 - 0.49 ↑ |
| 2 | 200 | categorical-crossentropy | ADAM | 2.500 | sem linhas | range 30 | 64 | 1 | 1.115 | 1.067 | 0.37 | 0.44 |

**Quadro 3:** Principais informações das redes de regressão treinadas no segundo conjunto de experimentos.

| N. | Épocas | Função de perda | Otimizador | DA's | Pré-proc | Data Aug | Batch Size | U. E. M. | Perda T | Perda V |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 200 | mean squared error | ADAM | 5.000 | com linhas | range 50 | 64 | 48 | 0.197 - 0.022 ↓ | 0.089 - 0.035 ↓ |
| 2 | 200 | mean absolute error | ADAM | 5.000 | sem linhas | range 30 | 32 | 17 | 0.245 - 0.127 ↓ | 0.184 - 0.147 ↓ |
| 3 | 150 | mean squared error | ADAM | 5.000 | com linhas | range 30 | 32 | 33 | 0.215 - 0.023 ↓ | 0.081 - 0.035 ↓ |
| 4 | 100 | mean squared error | ADAM | 5.000 | sem linhas | range 30 | 32 | 7 | 0.122 - 0.039 ↓ | 0.099 - 0.036 ↓ |
| 5 | 100 | mean squared error | ADAM | 20.000 | com linhas | range 30 | 32 | 9 | 0.097 - 0.034 ↓ | 0.058 - 0.033 ↓ |
| 6 | 60 | mean squared error | ADAM | 20.000 | sem linhas | range 30 | 32 | 13 | 0.104 - 0.029↓ | 0.057 - 0.033 ↓ |

A partir do primeiro Quadro, é possível ver que as redes treinadas para a tarefa de classificação não se saíram muito bem. Ambas não passaram de 50% de acurácia e no caso da primeira, pode ainda ser analisado que provavelmente ocorreu um *overfitting*, já que a perda de treinamento diminuiu, porém a de validação aumentou.

No caso das redes de regressão, a rede de número 2 não obteve um resultado muito bom, pois, como é possível ver, sua perda tanto de treinamento quanto de validação não diminuíram muito. A principal diferença desta para as outras era que esta utilizava a função de perda de erro médio absoluto ao invés da de erro médio quadrático, o que pode nos levar a concluir que a primeira não é muito indicada para este tipo de tarefa.

Já as outras redes de regressão obtiveram bons resultados pois conseguiram diminuir a perda de validação de algo em torno de 0.1 à 0.03 na maioria das vezes. Os bons resultados de predição da rede 6 podem ser vistos no Apêndice E.

O pré-processamento da imagem que era recebida via *stream,* ou seja, das imagens referentes ao cenário real, na hora de testar a rede na prática com o carrinho, era praticamente igual ao utilizado para a respectiva rede treinada. Era feito o *crop*, *resize*, *Gaussian Blur*, *Canny* e, quando a estratégia de adicionar linhas havia sido utilizada na parte do treinamento da rede que estava sendo testada, também eram

utilizadas no teste e vice-versa. A grande diferença é que não era feita a conversão de cores para *GrayScale* e nem era usado o método *threshold*.

Além disso, ainda em relação às imagens que eram recebidas via *stream*, passou a ser utilizado um método para detectar somente o espectro de cor azul na imagem, motivo pelo qual a conversão para *Grayscale* deixou de ser feita. Isso porque a cor da fita utilizada para marcar as bordas da pista era azul e, desta forma, somente a pista passava a aparecer na imagem pré-processada, apagando totalmente possíveis ruídos e objetos indesejados.

Dessa forma, o filtro de cor azul era usado antes do método *canny*, de maneira que o *canny* passava a somente marcar as bordas da pista, já que o resto da imagem ficava preto. Isso resolvia o problema relatado na Seção 5.1, relacionado ao fato do *canny* por vezes marcar bordas do ambiente, como do piso, por exemplo, o que acabava por atrapalhar bastante na predição.

As Figuras 60 e 61 mostram uma foto de uma pista criada e a mesma imagem pré-processada, sem a utilização do filtro de cor azul.

**Figura 60:** Imagem da pista criada com a fita azul.

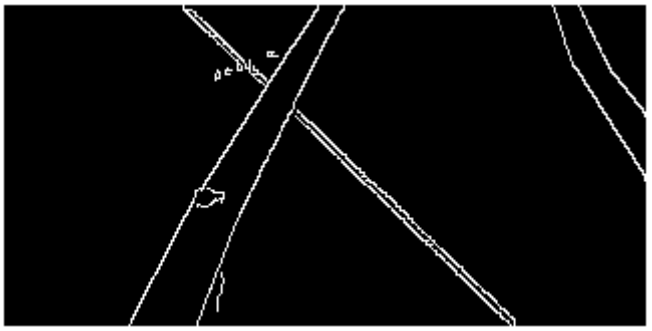


**Figura 61:** *Canny* da imagem sem o filtro de cor azul.

Como é possível ver, o método *canny* acaba identificando também a marcação

do piso, o que interfere na qualidade da imagem, afetando por fim a predição da rede.

Já as Figuras 62 e 63 demonstram a imagem da Figura 60 pré-processada utilizando o filtro de cor azul e o *canny* respectivamente.

**Figura 62:** Imagem parcialmente pré-processada, com filtro de cor azul.

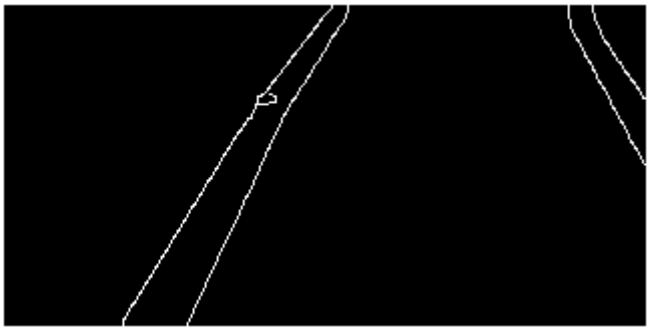


**Figura 63:** *Canny* da imagem com o filtro de cor azul.

Dessa maneira, é possível perceber que o filtro de luz azul ajudou muito a melhorar a qualidade da imagem que iria entrar na rede neural. O filtro mantém somente objetos de cor azul na imagem, apagando todo o resto, permitindo utilizar o método *canny* em cima desta nova imagem obtida, marcando assim somente a pista e ignorando outras marcações do piso ou objetos do local (contanto que estes não sejam da cor azul).

Após o treinamento, primeiramente foram realizados testes sem rodar o carrinho, utilizando imagens de uma pista criada. Todas as redes (classificação e regressão) foram treinadas, porém, apesar de muitas vezes apresentarem bons resultados, como foi demonstrado no primeiro grupo de testes, algumas vezes elas classificavam errado. Esse comportamento inconsistente prejudicava os testes, pois algumas predições erradas já eram capazes de tirar o carrinho da pista, de modo que a visão desta era perdida e, portanto, o carrinho ficava completamente "cego", já que as imagens streamadas já não continham mais nenhuma parte da pista. As duas redes de

classificação sempre forneciam valores parecidos para a posição que representava esquerda e para a que representava a direita, normalmente apresentando um valor um pouco menor para o centro, independentemente de pra qual lado fosse a curva, como demonstram as Figuras 64 e 65.

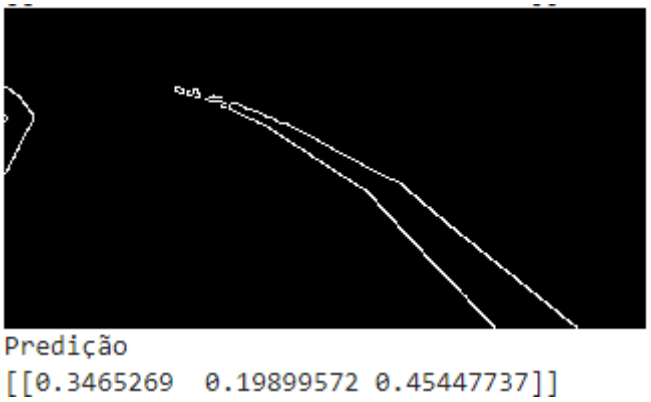



**Figura 64:** Imagem pré-processada que passou pela rede 1 de classificação e o resultado da predição.

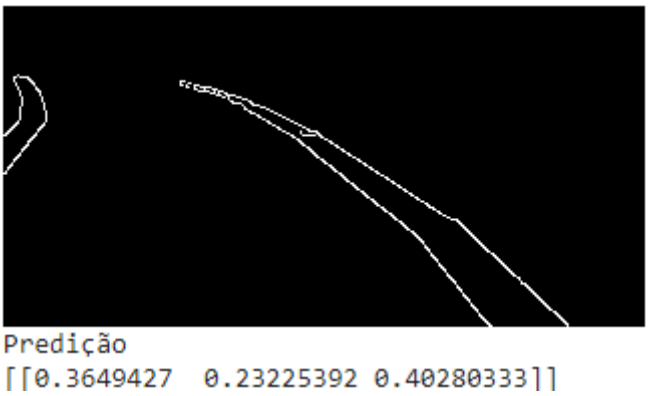



**Figura 65:** Imagem pré-processada que passou pela rede 2 de classificação e o resultado da predição.

Como é possível perceber, ambas as Figuras 64 e 65, as quais passaram pela rede 1 e 2 de classificação, claramente representam uma curva para a esquerda. Entretanto, a posição do *array* que possui maior valor é a terceira, a qual representa curvas para a direita. Lembrando que o *output* da rede é dado por esse vetor de 3 posições, as quais representam esquerda, centro e direita, respectivamente. O valor de

cada uma das posições significa a porcentagem de chance daquela posição ser o resultado correto para determinada entrada (imagem). Além disso, apesar de ambas as imagens representarem curvas acentuadas, o valor das posições que representam esquerda e direita não são assim tão diferentes, o que acaba por não fazer muito sentido.

Foi feito o teste também considerando a curva para o outro lado, utilizando as mesmas redes 1 e 2 de classificação e o resultado pode ser visto nas Figuras 66 e 67.

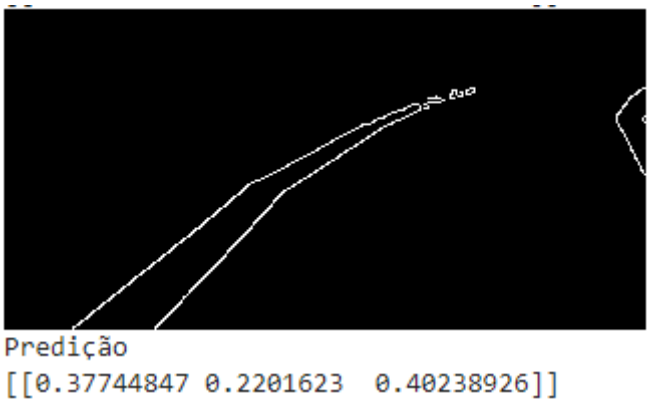



**Figura 66:** imagem pré-processada que passou pela rede 1 de classificação e o resultado da predição.

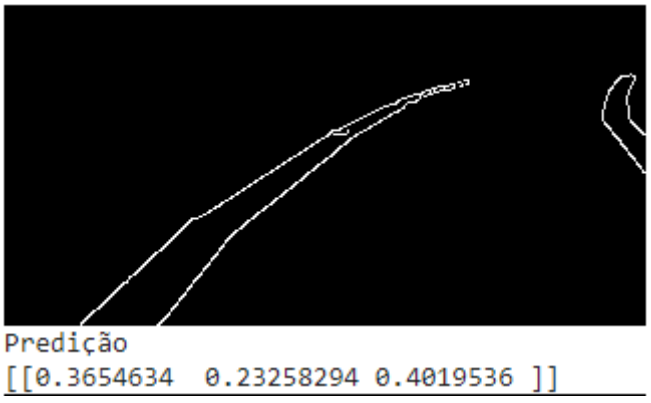



**Figura 67:** Imagem pré-processada que passou pela rede 2 de classificação e o resultado da predição.

Como pode ser observado através das Figuras 66 e 67, mesmo mudando o lado da curva, os resultados continuaram muito próximos. Esse resultado ruim não foi exatamente uma surpresa, pois como é possível ver no Quadro de redes de classificação, ambas as redes 1 e 2 obtiveram acurácia máxima abaixo de 50%. Isso significa que a rede estava fornecendo resultados apenas um pouco melhor do que se definissemos

aleatoriamente a classe de cada imagem, já que somente existiam 3 classes e, portanto, a chance de acertar aleatoriamente a classe seria de aproximadamente 33,3%. Somente por propósitos de confirmação, foram realizados testes utilizando o carrinho e o *stream* de imagens, porém os resultados continuaram seguindo esse padrão.

Após isso, foram realizados testes utilizando as redes de regressão, as quais haviam obtido resultados muito melhores no treinamento, já que a maioria havia conseguido reduzir a perda de validação para algo próximo de 0.033, a não ser no caso da rede 2, cuja principal diferença foi que utilizou a função de perda *mean absolute error,* ao invés da *mean squared error*. Neste caso chega-se à conclusão de que a última é melhor para este tipo de treinamento do que a primeira. A maior esperança estava nas redes de regressão, já que estas apresentavam resultados consideravelmente bons quando testados utilizando o *dataset* de validação, com as imagens do simulador, como é possível ver no Apêndice E. Nele é possível ver que a maioria das predições fornecidas resultaram em valores bem próximos dos labels destas imagens. Em alguns casos os resultados não eram tão próximos, mas sempre possuíam o mesmo sinal positivo ou negativo em comparação com o label.

As predições do Apêndice E foram fornecidas pela rede 6 de regressão. Esta, como é possível verificar no Quadro, utilizou 20.000 DA's, com o intuito de tornar a rede o mais genérica possível. Não foi utilizado um valor maior do que 20.000 pois, se não, o treinamento começaria a ficar demasiado demorado.

Entretanto, quando a rede foi testada com imagens reais da pista criada com a fita azul, o resultado não foi sempre satisfatório, mas inconsistente, como foi falado anteriormente. Os Apêndices F e G demonstram alguns casos de resultados ruins de predição da rede. O Apêndice F apresenta imagens pré-processadas da pista criada com a fita azul, com a curva sendo realizada para a esquerda, seguidas de suas predições realizadas pela rede 6 de regressão. Já é possível ver no Apêndice F que, apesar da curva estar sendo feita para a esquerda, o resultado para a maioria das imagens é positivo (o qual deveria ser bem negativo, dado que se tratavam de curvas acentuadas).

Já no caso das imagens do Apêndice G, a maioria das predições são valores positivos, como era de se esperar, com alguns valores bem negativos. Apesar de nesse caso serem de fato esperados valores positivos, como as imagens com curva para a esquerda também obtiveram resultados parecido, isso levou à conclusão de que a rede não estava conseguindo predizer corretamente, no caso destas imagens, mas somente resultando valores positivos na maioria das predições, o que no caso das curvas para a

direita calharam de coincidir com o resultado esperado.

Além disso, estes resultados extremos negativos nas imagens com curva para a direita também indicam a inconsistência dos resultados da rede.

Por outro lado, o vídeo presente no github do projeto demonstra 2 ótimos resultados obtidos pela rede, rodando o carrinho na prática. Nestes casos, foram realizadas curvas simples em S para esquerda e direita e o carrinho com auxílio da IA foi capaz de completá-las praticamente sem sair da pista. Portanto, chega-se à conclusão de que por algum motivo a rede conseguia predizer com certa precisão algumas imagens do cenário real, enquanto ao processar outras aparentava se perder, apresentando portanto, como dito anteriormente, um resultado de certa forma inconsistente.

# 6 Conclusão

## 6.1 *Considerações finais*

Como foi possível observar nos experimentos apresentados no capítulo anterior, a conclusão chegada é que a rede conseguiu ser treinada satisfatoriamente para predizer imagens do simulador, ou seja, imagens parecidas com as utilizadas no treinamento e validação. Entretanto, quando estas mesmas redes eram utilizadas para predizer imagens de fotos de pistas criadas no mundo real através de uma fita crepe azul, como mostrado ao longo do trabalho, os resultados eram inconsistentes, por vezes acertando e por vezes se perdendo.

Isso indica que as redes profundas treinadas não foram capazes de alcançar generalização a ponto de estender completamente sua capacidade de predição a imagens que não fossem aquelas do simulador, mesmo tentando aproximá-las às reais através dos pré-processamentos utilizados.

Além disso, foi possível ver que as redes treinadas na maioria das vezes paravam de melhorar, durante o treinamento, nas épocas iniciais. Poucas redes conseguiam melhorar além da época 50, mesmo treinando por mais de 100 épocas. Isso pode indicar que, na maioria dos treinamentos, as redes estavam caindo em um mínimo local da função de perda, sem conseguir sair desta região e, portanto, sem conseguir melhorar além disso.

Mesmo assim, foi possível chegar a um ótimo pré-processamento, principalmente para o caso das imagens do cenário real, de modo que a última versão de pré-processamento utilizada era capaz de demarcar somente a pista ignorando todos os outros objetos e "ruídos" em cena, de modo que este poderia continuar sendo utilizado em trabalhos futuros.

## 6.2 *Limitações e trabalhos futuros*

Para obter melhores resultados nas pistas criadas, provavelmente seria preciso usar uma rede mais robusta, com uma maior quantidade de imagens no *dataset*, ou

utilizar um *dataset* de treinamento formado a partir de imagens similares às da pista criada com a fita azul. Esta última opção possui mais probabilidade de ser efetiva e trazer melhorias na predição da rede nos experimentos com a pista real. Desta forma a limitação, neste caso, foi a falta de tempo para gerar um *dataset* manualmente, assim como de tentar mais combinações de hiperparâmetros, arquiteturas de rede etc. Portanto, trabalhos futuros poderiam, inspirados neste, continuar os testes realizados, partindo do ponto até onde este trabalho chegou e levando em consideração toda a estrutura de sistema distribuído construída, assim como as evoluções em relação ao *data augmentation*, hiperparâmetros e pré-processamento utilizados, testando outras possibilidades de otimizadores ou mesmo utilizando *transfer learning* através da rede VGG.19. Poderia também ser testado a utilização de redes treinadas a partir de aprendizado por reforço. Indo além, estes poderiam, após obter sucesso na etapa de seguir corretamente a pista a partir da visão das bordas, adicionar outras características ao treinamento da rede, como por exemplo:

- a identificação de objetos para conseguir desviar deles;
- identificação de sinais de trânsito;
- identificação de semáforos.

# Referências Bibliográficas


1. ALBERT, EDWARD. **AI in talent acquisition: a review of AI-applications used in recruitment and selection**. Strategic HR Review, Vol. 18 No. 5, 2019.

2. CALDERON, PATRICK; OKABE, KEN; MORENO, FERNANDA; YAFFAR, JORDI. Autonomous Driving on NVIDIA Dave-2. Document revision of "End to End Learning for Self-Driving Cars " and adaptation on Udacity Simulator, 2021 Selected Topics in Artificial Intelligence. Professor Gissel Velarde.

3. CUPERTINO . **Rapid Autonomous Car Control based on Spatial And Temporal Visual Cues**. In: 1st Surya Dantuluri, Monta Vista High School, 2018, USA.

4. DEVI, T.; SRIVATSAVA, AKSHAT; MUDGAL, KRITESH; JAYANTI, RANJNISH; KARTHICK, T.. **Behaviour Cloning for Autonomous Driving**. ISSN: 1735-188X DOI: 10.14704/WEB/V17I2/WEB17061, 2020.

5. KROHN, JHON; BEYLEVELD, GRANT; BLASSENS, AGLAÉ. **Deep learning illustrated**, 2020.

6. LEÓN-VERA, LEONARDO; MORENO-VERA, FELIPE; CASTRO-CRUZ, RENATO; JOSE, NAVÍO-TORRES; MARCO CAPCHA-MANSILLA. **Car Monitoring System in Apartment Garages by Small Autonomous Car using Deep Learning**. Universidad Tecnologica del Perú, 2019.

7. OLIVEIRA, NICOMAR. **Classificação de Sinais de Trânsito em Direção Autônoma por Redes Neurais Convolucionais.** 2020.

8. RUSSELL, STUART; NORVIG, PETER. **Artificial intelligence: a modern approach**, 2002.

9. SAJJAD, MUHAMMAD; IRFAN, MUHAMMAD; MUHAMMAD, KHAN; SER, JAVIER; SANCHEZ-MEDINA, JAVIER; ANDREEV, SERGEY; DING, WEIPING; LEE, JONG. **An Efficient and Scalable Simulation Model for Autonomous Vehicles With Economical Hardware**. IEEE transactions on intelligent transportation systems, VOL. 22, NO. 03, 2021.

10. Swaminathan, Vaibhav; Shrey, Arora; Ravi, Bansal; Rajalakshmi R*. **Autonomous Driving System with Road Sign Recognition using Convolutional Neural Networks**. In: **Second International Conference on Computational Intelligence in Data Science**, 2019, Tamilnadu, India.

11. YIQIN, YANG; ZHE, WU; QINGYANG, XU; FABAO, YAN. **Deep Learning Technique-Based Steering of Autonomous Car**. International Journal of Computational Intelligence and Applications Vol. 17, No. 2, School of Mechanical, Electrical & Information Engineering Shandong University, 180 Wenhua Xilu, Weihai, 2018.

12. Funções de Ativação. Disponível em: https://matheusfacure.github.io/2017/07/12/activ-func/. Acesso em 20 de Jul. 2022.

## APÊNDICES

**Apêndice A -** *Log* de treinamento da rede neural, mostrando somente as épocas em que houve melhoria do sexto experimento do primeiro grupo de experimentos.

```
Epoch 1/50
126/126 [==============================] - ETA: 0s - loss: 0.1180
Epoch 1: val_loss improved from inf to 0.07613, saving model to
drive/MyDrive/TCC/Trabalho
Pratico/6-ModelsNNAutoCar/bestmodel-001.h5
126/126 [==============================] - 915s 7s/step - loss:
0.1180 - val_loss: 0.0761
Epoch 2/50
126/126 [==============================] - ETA: 0s - loss: 0.0725
Epoch 2: val_loss improved from 0.07613 to 0.05431, saving model to
drive/MyDrive/TCC/Trabalho
Pratico/6-ModelsNNAutoCar/bestmodel-002.h5
126/126 [==============================] - 41s 324ms/step - loss:
0.0725 - val_loss: 0.0543
Epoch 3/50
126/126 [==============================] - ETA: 0s - loss: 0.0627
Epoch 3: val_loss improved from 0.05431 to 0.04667, saving model to
drive/MyDrive/TCC/Trabalho
Pratico/6-ModelsNNAutoCar/bestmodel-003.h5
126/126 [==============================] - 34s 270ms/step - loss:
0.0627 - val_loss: 0.0467
Epoch 5/50
126/126 [==============================] - ETA: 0s - loss: 0.0518
Epoch 5: val_loss improved from 0.04667 to 0.04359, saving model to
drive/MyDrive/TCC/Trabalho
Pratico/6-ModelsNNAutoCar/bestmodel-005.h5
126/126 [==============================] - 34s 268ms/step - loss:
0.0518 - val_loss: 0.0436
Epoch 9/50
126/126 [==============================] - ETA: 0s - loss: 0.0420
Epoch 9: val_loss improved from 0.04359 to 0.04335, saving model to
drive/MyDrive/TCC/Trabalho
Pratico/6-ModelsNNAutoCar/bestmodel-009.h5
```

```
126/126 [==============================] - 34s 269ms/step - loss:
0.0420 - val_loss: 0.0433
Epoch 17/50
126/126 [==============================] - ETA: 0s - loss: 0.0310
Epoch 17: val_loss improved from 0.04335 to 0.04287, saving model
to                                    drive/MyDrive/TCC/Trabalho
Pratico/6-ModelsNNAutoCar/bestmodel-017.h5
126/126 [==============================] - 34s 268ms/step - loss:
0.0310 - val_loss: 0.0429
Epoch 23/50
126/126 [==============================] - ETA: 0s - loss: 0.0274
Epoch 23: val_loss improved from 0.04287 to 0.04246, saving model
to                                    drive/MyDrive/TCC/Trabalho
Pratico/6-ModelsNNAutoCar/bestmodel-023.h5
126/126 [==============================] - 33s 264ms/step - loss:
0.0274 - val_loss: 0.0425
Epoch 29/50
126/126 [==============================] - ETA: 0s - loss: 0.0242
Epoch 29: val_loss improved from 0.04246 to 0.04204, saving model
to                                    drive/MyDrive/TCC/Trabalho
Pratico/6-ModelsNNAutoCar/bestmodel-029.h5
126/126 [==============================] - 34s 273ms/step - loss:
0.0242 - val_loss: 0.0420
Epoch 32/50
126/126 [==============================] - ETA: 0s - loss: 0.0236
Epoch 32: val_loss improved from 0.04204 to 0.04134, saving model
to                                    drive/MyDrive/TCC/Trabalho
Pratico/6-ModelsNNAutoCar/bestmodel-032.h5
126/126 [==============================] - 33s 264ms/step - loss:
0.0236 - val_loss: 0.0413
Epoch 36/50
126/126 [==============================] - ETA: 0s - loss: 0.0203
Epoch 36: val_loss improved from 0.04134 to 0.04128, saving model
to                                    drive/MyDrive/TCC/Trabalho
Pratico/6-ModelsNNAutoCar/bestmodel-036.h5
126/126 [==============================] - 34s 267ms/step - loss:
0.0203 - val_loss: 0.0413
Epoch 42/50
126/126 [==============================] - ETA: 0s - loss: 0.0208
```

```
Epoch 42: val_loss improved from 0.04128 to 0.04128, saving model
to                                    drive/MyDrive/TCC/Trabalho
Pratico/6-ModelsNNAutoCar/bestmodel-042.h5
126/126 [==============================] - 33s 264ms/step - loss:
0.0208 - val_loss: 0.0413
Epoch 43/50
126/126 [==============================] - ETA: 0s - loss: 0.0199
Epoch 43: val_loss improved from 0.04128 to 0.04123, saving model
to                                    drive/MyDrive/TCC/Trabalho
Pratico/6-ModelsNNAutoCar/bestmodel-043.h5
126/126 [==============================] - 33s 263ms/step - loss:
0.0199 - val_loss: 0.0412
Epoch 44/50
126/126 [==============================] - ETA: 0s - loss: 0.0181
Epoch 44: val_loss improved from 0.04123 to 0.04111, saving model
to                                    drive/MyDrive/TCC/Trabalho
Pratico/6-ModelsNNAutoCar/bestmodel-044.h5
126/126 [==============================] - 33s 261ms/step - loss:
0.0181 - val_loss: 0.0411
Epoch 46/50
126/126 [==============================] - ETA: 0s - loss: 0.0189
Epoch 46: val_loss improved from 0.04111 to 0.04110, saving model
to                                    drive/MyDrive/TCC/Trabalho
Pratico/6-ModelsNNAutoCar/bestmodel-046.h5
126/126 [==============================] - 33s 262ms/step - loss:
0.0189 - val_loss: 0.0411
Epoch 47/50
126/126 [==============================] - ETA: 0s - loss: 0.0193
Epoch 47: val_loss improved from 0.04110 to 0.04089, saving model
to                                    drive/MyDrive/TCC/Trabalho
Pratico/6-ModelsNNAutoCar/bestmodel-047.h5
126/126 [==============================] - 34s 269ms/step - loss:
0.0193 - val_loss: 0.0409
Epoch 50/50
126/126 [==============================] - ETA: 0s - loss: 0.0176
Epoch 50: val_loss improved from 0.04089 to 0.04017, saving model
to                                    drive/MyDrive/TCC/Trabalho
Pratico/6-ModelsNNAutoCar/bestmodel-050.h5
```

```
126/126 [==============================] - 33s 261ms/step - loss:
0.0176 - val_loss: 0.0402
```

**Apêndice B -** *Log* de *outputs* da rede neural do sexto experimento (Seção 6.1.8) da primeira etapa de treinamentos. Foram apresentadas 10 imagens do cenário da pista real para a rede neural e as saídas estimadas pela rede são expostas abaixo, assim como a média artimética destas.

1

0.33344445

2

0.2202781

3

0.28063187

4

0.16561353

5

0.27973175

6

0.2827734

7

0.2265096

8

0.19703527

9

0.21826497

10

0.26611826

Média

0.2468231

**Apêndice C -** Log de *outputs* da rede neural do sexto experimento da primeira etapa de treinamentos. São apresentados 10 *outputs* e ao final a média aritmética destes. Neste caso, o carrinho estava em uma posição diferente da posição do Apêndice B.

1

0.05023249

2

0.04526652

3

-0.00534536

4

0.03370613

5

0.03922284

6

0.05251731

7

0.09563781

8

-0.01910254

9

0.10568858

10

0.04992292

Média

0.04537845

**Apêndice D -** Log de *outputs* da rede neural referente ao Sexto Experimento (Seção  da primeira etapa de treinamentos. São apresentados 10 *outputs* e ao final a média aritmética destes. Neste caso, o carrinho estava em uma posição diferente da posição dos Apêndices A e B.

1

-0.312194

2

-0.09539973

3

-0.18263

4

-0.16737574

5

-0.08589181

6

-0.41635656

7

-0.25486207

8

-0.15955648

9

-0.10514191

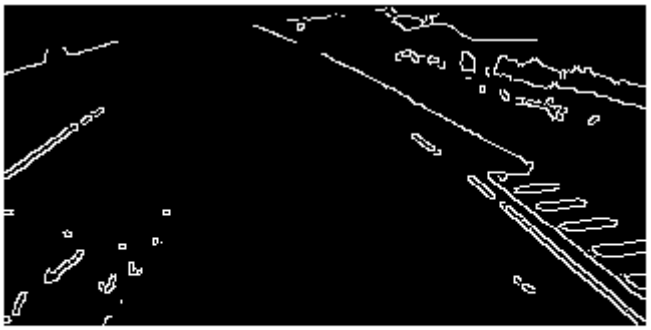
10

-0.02757783

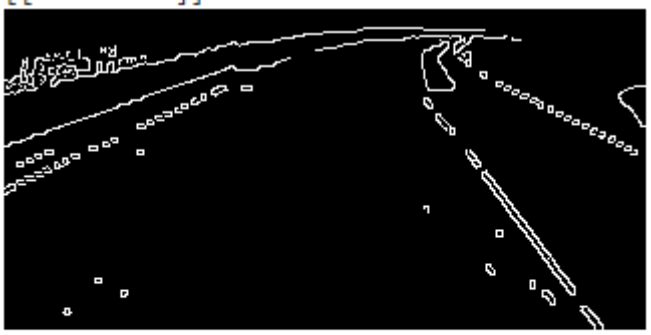
Média

-0.20471999

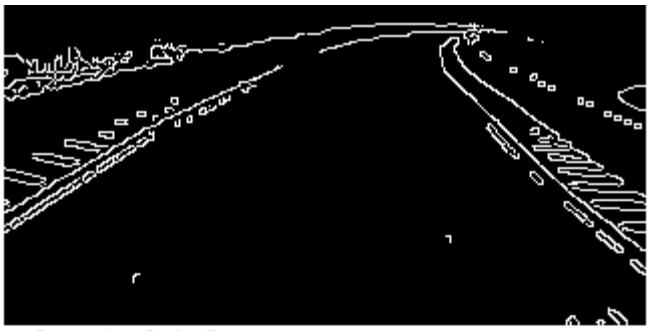
**Apêndice E -** imagens pré-processadas da pista do simulador *UDacity* do *dataset* de validação, seguidas de seus respectivos labels e da predição fornecida pela rede 6 de regressão.



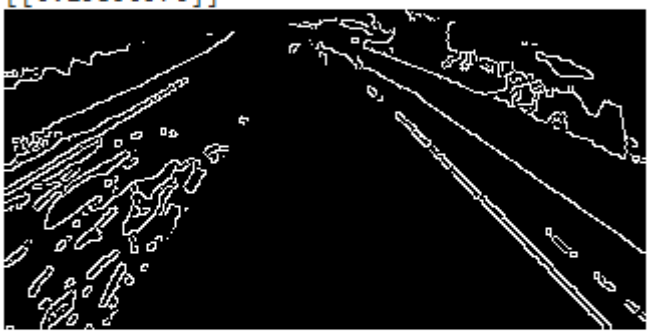
```
Valor do label
-0.6500001
Predição
[[-0.370129]]
```



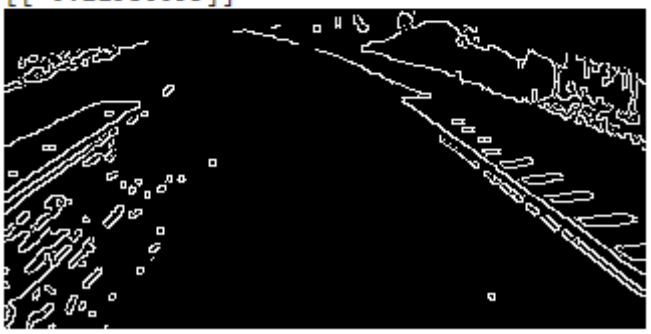
```
Valor do label
0.25
Predição
[[0.23627192]]
```



Valor do label
0.45
Predição
[[0.19390976]]

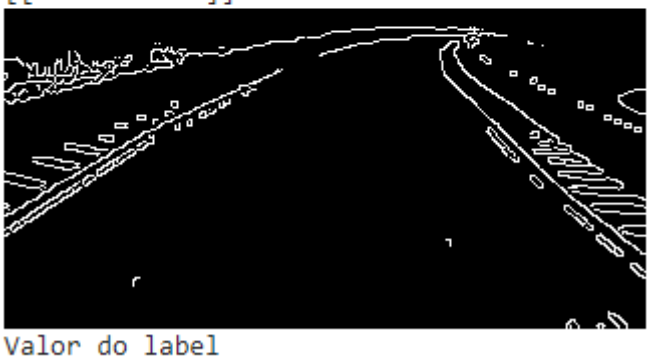


Valor do label
-0.25
Predição
[[-0.21980095]]



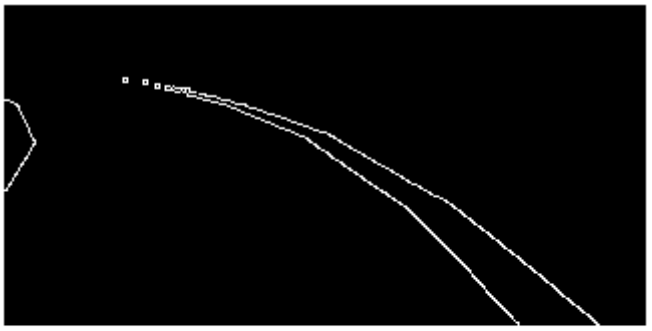
Valor do label
-0.2
Predição
[[-0.4750455]]

```
Valor do label
-0.15
Predição
[[-0.16702686]]
```

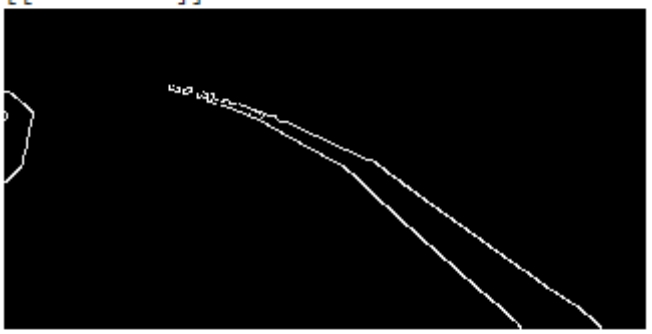


```
Valor do label
0.45
Predição
[[0.19390976]]
```

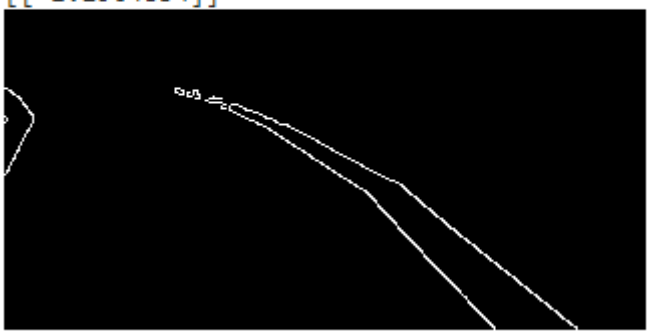
**Apêndice F -** imagens pré-processadas da pista criada, com a curva sendo feita para a esquerda, seguidas das predições fornecidas pela rede 6 de regressão.



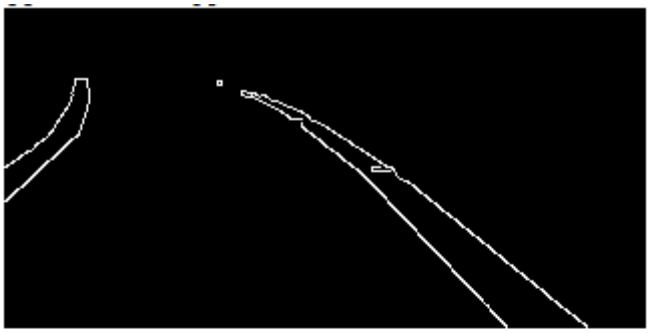
Predição
[[2.2464569]]



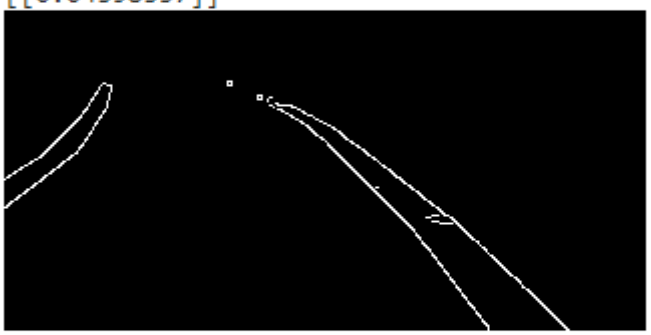
Predição
[[-1.1964834]]



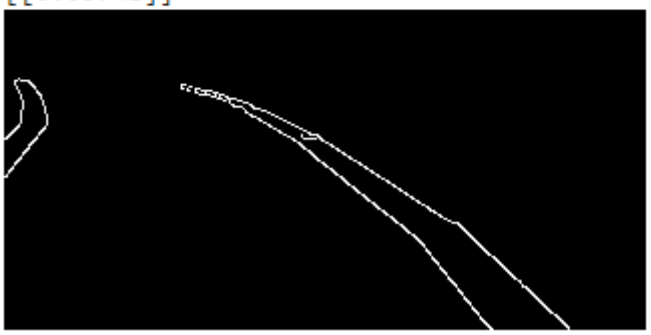
Predição
[[0.17522377]]



Predição
[[0.04398337]]

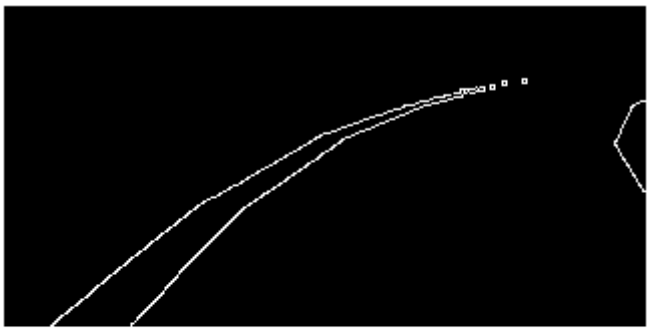


Predição
[[0.65741]]

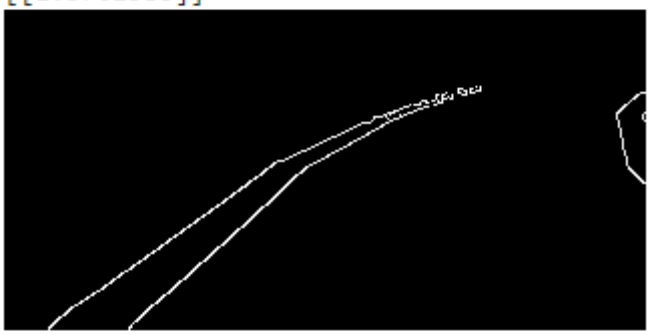


Predição
[[2.7021544]]

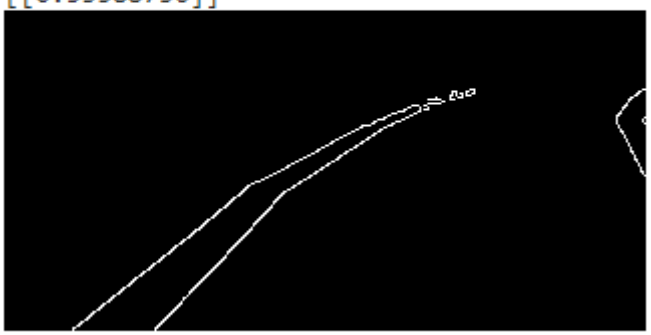
**Apêndice G -** imagens pré-processadas da pista criada, com a curva sendo feita para a direita, seguidas das predições fornecidas pela rede 6 de regressão.



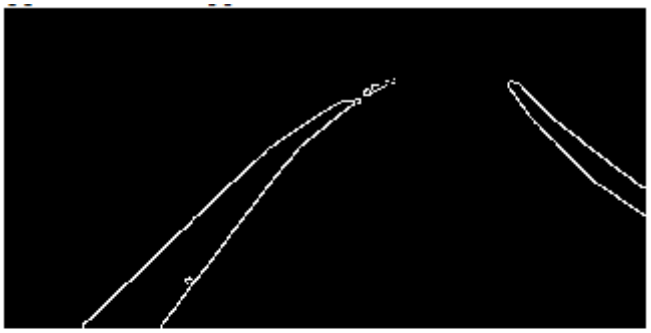
Predição
[[1.3702388]]



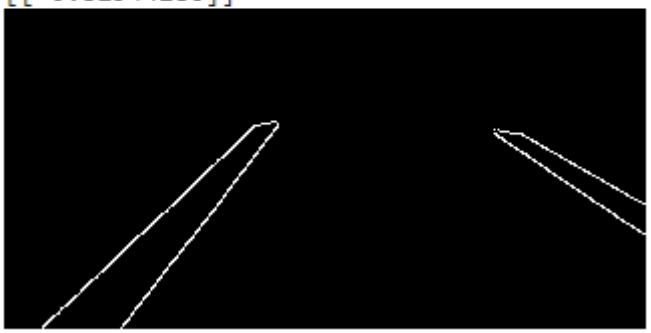
Predição
[[0.53588736]]



Predição
[[0.32399723]]



Predição
[[-0.82944286]]



Predição
[[-0.31936613]]