CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS TIMÓTEO

Diego Haji Carvalho Campos

CRIAÇÃO E VALIDAÇÃO DE UMA BASE DE DADOS COM ALGUNS ELEMENTOS DO TRÂNSITO BRASILEIRO PARA VEÍCULOS AUTÔNOMOS

Timóteo

2019

Diego Haji Carvalho Campos

CRIAÇÃO E VALIDAÇÃO DE UMA BASE DE DADOS COM ALGUNS ELEMENTOS DO TRÂNSITO BRASILEIRO PARA VEÍCULOS AUTÔNOMOS

Monografia apresentada à Coordenação de Engenharia de Computação do Campus Timóteo do Centro Federal de Educação Tecnológica de Minas Gerais para obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Elder de Oliveira Rodrigues

Timóteo

2019

Diego Haji Carvalho Campos

CRIAÇÃO E VALIDAÇÃO DE UMA BASE DE DADOS COM ALGUNS ELEMENTOS DO TRÂNSITO BRASILEIRO PARA VEÍCULOS AUTÔNOMOS

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais para a obtenção do título de Engenheiro de Computação.

Trabalho aprovado. Timóteo 17 de junho de 2019:

Prof. Dr. Elder de Oliveira Rodrigues
Orientador

Prof. Me. Odilon Corrêa Silva
Professor Convidado

Prof. Me. Adilson Mendes Ricardo
Professor Convidado

Timóteo
2019

Agradecimentos

Agradeço aos meus professores que me ajudaram a chegar até aqui e aos meus familiares que sempre me apoiaram.

Resumo

A popularização do aprendizado de máquina vem possibilitando o surgimento de novos produtos e recursos revolucionários, como veículos autônomos que são capazes de compreender e responder eficientemente à toda a diversidade de situações presentes no trânsito. Graças às redes neurais artificiais aplicadas à visão computacional, os sistemas tornaram-se aptos a reconhecer padrões de forma muito próxima à visão humana. Para o treinamento de redes neurais profundas (*Deep Learning*) é necessária uma base de dados numerosa para assegurar a compreensão de padrões pelos processadores de dados. Visando contribuir com as aplicações para veículos autônomos, este trabalho cria uma base de dados com alguns elementos de sinalização de trânsito brasileiro. Para a validação e treinamento da base de dados foi utilizado a rede Yolov3 e realizados testes em diversos cenários, avaliando critérios como a quantidade de objetos detectados e a capacidade de identificação correta do tipo de objeto. Como resultado, para o ambiente virtual houve uma localização de 17,9% de objetos, o qual 83% destes foram classificados corretamente. E para o cenário real houve uma localização de 94% de objetos, o qual 61,8% destes foram classificados corretamente.

Palavras-chave: Aprendizado de Maquina, Veículos Autônomos, Redes Neurais Artificiais, Deep Learning, Yolov3, Base de Dados.

Abstract

The popularization of machine learning has enabled the emergence of revolutionary new products and resources, such as autonomous vehicles that are capable of understanding and responding efficiently to all the diversity of situations present in traffic. Thanks to artificial neural networks applied to computer vision, systems have become able to recognize patterns very close to human vision. For deep neural network training (Deep Learning) a large database is required to ensure understanding of patterns by data processors. Aiming to contribute to the applications for autonomous vehicles, this work creates a database with some elements of Brazilian traffic signaling. For the validation and training of the database, the Yolov3 network was used and tests were performed in several scenarios, evaluating criteria such as the number of objects detected and the ability to correctly identify the type of object. As a result, for the virtual environment there was a location of 17.9% of objects, of which 83% were correctly classified. And for the real scenario there was a location of 94% of objects, which 61.8% of these were classified correctly.

Keywords: Machine Learning, Autonomous Vehicle, Artificial Neural Network, Deep Learning, Yolov3, Database.

Lista de ilustrações

Figura 1 – Exemplo de uma camada de convolução. 19
Figura 2 – Legendas geradas por uma Rede Neural Recorrente(RNN), utilizando uma uma CNN. 20
Figura 3 – Funcionamento do sistema de detecção de YOLO de acordo com seu criador. 21
Figura 4 – Fluxograma descrevendo os passos necessários para a execução do projeto. 28
Figura 5 – Fluxograma descrevendo os passos para a implementação de uma CNN para detecção de objetos. 30
Figura 6 – Funcionamento do sistema de caixas delimitadores de YOLOv3 de acordo com seu criador. 32
Figura 7 – Figura ilustrando o arquivo obj.data. 39
Figura 8 – Figura ilustrando o arquivo obj.names. 40
Figura 9 – Figura ilustrando uma marcação manual. 41
Figura 10 – Figura ilustrando as coordenadas contidas em um arquivo com a marcação manual, referente a figura 9. 41
Figura 11 – Figura ilustrando a posição das ancoras. 41
Figura 12 – Figura ilustrando a região das ancoras, referente a figura 11 42
Figura 13 – Figura ilustrando o funcionamento do mAP e do IoU. 45
Figura 14 – Figura ilustrando o mAP e o IoU conforme o treinamento da rede. 46
Figura 15 – Figura ilustrando a posição das ancoras. 47
Figura 16 – Figura ilustrando a região das ancoras, referente a figura 15 48
Figura 17 – Figura ilustrando o mAP e o IoU conforme o treinamento da rede com a adição de novas imagens. 49
Figura 18 – Detecção de objetos no ambiente virtual do cenário 1. 51
Figura 19 – Detecção de objetos no ambiente virtual do cenário 2. 51
Figura 20 – Detecção de objetos no ambiente virtual do cenário 3. 52
Figura 21 – Detecção de objetos no ambiente virtual do cenário 4. 52
Figura 22 – Detecção de objetos no ambiente virtual do cenário 5. 53
Figura 23 – Detecção de objetos no ambiente virtual do cenário 6. 53
Figura 24 – Detecção de objetos de cenários reais com imagens do *Google Images*. . . 56
Figura 25 – Detecção de objetos de cenários reais com imagens do *Google Images*. . . 56
Figura 26 – Detecção de objetos de cenários reais com imagens do *Google Images*. . . 57
Figura 27 – Detecção de objetos de cenários reais com imagens do *Google Images*. . . 57
Figura 28 – Detecção de objetos de cenários reais com imagens do *Google Images*. . . 58
Figura 29 – Detecção de objetos de cenários reais com imagens do *Google Images*. . . 58
Figura 30 – Imagens de cada classe respectivamente. 67

Lista de tabelas

Tabela 1 – Tabela demonstrando o tempo gasto por cada implementação da rede Yolov3 para detecção de objetos em imagens. 33
Tabela 2 – Tabela demonstrando o tempo gasto por cada implementação da rede Yolov3-tiny para detecção de objetos em imagens. 34
Tabela 3 – Tabela demonstrando o tempo gasto por cada implementação da rede Yolov3 e para detecção de objetos em vídeos. 35
Tabela 4 – Tabela demonstrando o tempo gasto por cada implementação da rede Yolov3-tiny para detecção de objetos em vídeos. 36
Tabela 5 – Tabela mostrando os objetos que foram detectados pela rede distribuídos em 47 classes. 37
Tabela 6 – Continuação da tabela 5. 38
Tabela 7 – Tabela demonstrando o tempo gasto por iteração e quantidade de classes durante o treinamento com 2 classes para a rede *Yolov3-tiny*. 44
Tabela 8 – Tabela demonstrando o tempo gasto por iteração e quantidade de classes durante o treinamento com 2 classes para a rede *Yolov3*. 44
Tabela 9 – Tabela demonstrando o tempo gasto por iteração e quantidade de classes durante o treinamento com 47 classes para rede *Yolov3*. 45
Tabela 10 – Tabela demonstrando o tempo gasto por iteração e quantidade de classes durante o treinamento com 47 classes para rede *Yolov3* com a adição de novas imagens. 48
Tabela 11 – Tabela demonstrando a quantidade de objetos presentes em cada cenário. . 54
Tabela 12 – Tabela demonstrando a quantidade de objetos encontrados pela rede treinada, e quantos desses estão corretos, no ambiente virtual. 54
Tabela 13 – Tabela demonstrando a precisão relativa à quantidade de objetos detectados pela rede durante os testes no ambiente virtual. 55
Tabela 14 – Tabela demonstrando a precisão relativa classificação de objetos detectados pela rede durante os testes no ambiente virtual. 55
Tabela 15 – Tabela demonstrando os resultados dos cenários reais. 59

Lista de abreviaturas e siglas

ANN	Artificial Neural Network - (RNA) Rede Neural Artificial
CNN	Convolutional Neural Network - (RNC) Rede Neural Convolucional
DL	Deep Learning - (AP) Aprendizado Profundo
AI	Artificial Inteligence - (IA) Inteligência Artificial
AV	Automated Vehicle - (VA) Veículo Autônomo
ML	Machine Learning - (AM) Aprendizado de Maquina
R-CNN	Regional Based Convolutional Neural Network - (RNC-BR) Rede Neural Convolucional Baseada em Regiões
FPS	Frames per Second - (QPS) Quadros por Segundo
mAP	Mean Average Precision - (PM) Precisão Média
IoU	Intersection over Union - (IsU) Interseção sobre União
UE4	Unreal Engine 4

Sumário

1 INTRODUÇÃO 12
1.1 Justificativa e Problema 13
1.2 Objetivos 13

2 REVISÃO BIBLIOGRÁFICA 14
2.1 Fundamentos teóricos 14
2.1.1 Redes Neurais Artificiais 14
2.1.2 OpenCV 15
2.1.3 CUDA 15
2.1.4 cuDNN 16
2.1.5 Blender 3D 16
2.1.6 Unreal Engine 4 16
2.1.7 Deep Learning 17
2.1.8 Redes Neurais Convolucionais 18
2.1.9 Yolo - You Only Look Once 20
2.2 Fundamentos históricos 21
2.2.1 Carros autônomos 22
2.2.2 Deep Learning 23
2.2.3 Técnicas de Deep Learning 25
2.2.4 Ambientes de simulação 27

3 MÉTODOS E MATERIAIS 28
3.1 Projeto e implementação de uma CNN para detecção de objetos 29

4 BASE DE DADOS E TREINAMENTO 31
4.1 Yolov3 31
4.1.1 Testes preliminares 32
4.2 Base de dados 36
4.3 Treinamento 39
4.3.1 Arquivos 39
4.3.2 Rotulação Manual 40
4.3.3 Extras 41
4.3.4 Pesos 42
4.3.5 Configuração da rede 43
4.3.6 Execução 43
4.3.7 Avaliação inicial 45
4.3.8 Avaliação secundária 46

5 TESTES E RESULTADOS 50

5.1 Testes 50
5.1.1 Testes do ambiente virtual 51
5.1.2 Testes de cenários reais 55
5.2 Resultados 59
5.2.1 Ambiente virtual 59
5.2.2 Cenários reais 60
5.3 Comparação com trabalhos relacionados 60

6 CONCLUSÃO 62
6.1 Considerações e limitações 62
6.2 Trabalhos futuros 63

REFERÊNCIAS 64

APÊNDICE A – APÊNDICES 67
A.1 Objetos 67

1 Introdução

Uma das tecnologias que vem ganhando destaque na computação é Machine Learning (ML), que é uma aplicação de inteligência artificial (IA) que fornece aos sistemas a capacidade de aprender e melhorar automaticamente a partir da experiência sem ser explicitamente programado (SYSTEM, 2017). O ML é utilizado por grandes empresas na criação, por exemplo, de *chatbots*, reconhecimento e processamento de linguagem natural e reconhecimentos faciais.

Em relação ao reconhecimento de imagens, o ML também é amplamente utilizado, como em Bullock, Cuesta-Lazaro e Quera-Bofarull (2018), que o ML foi treinado para avaliar imagens de raio-x e obteve uma precisão de 92%.

Uma técnica que geralmente é empregada em conjunto com a área de reconhecimento de imagem é o *Deep Learning* (DL). O DL é uma abordagem específica usada para construir e treinar redes neurais, que são considerados nós de decisão altamente promissores. Um algoritmo é considerado *"Deep"* se os dados de entrada forem passados por uma série de não-linearidades ou transformações não-lineares antes de se tornarem saídas. Em contraste, a maioria dos algoritmos modernos de aprendizado de máquina são considerados "superficiais" porque a entrada só pode levar apenas alguns níveis de chamada de sub-rotina (TECHOPEDIA, 2019b).

O *Deep Learning* também é amplamente utilizado em conjunto com a Visão Computacional (VC). A visão computacional é um campo da ciência da computação que trabalha para permitir que os computadores vejam, identifiquem e processem imagens da mesma maneira que a visão humana, e depois forneçam resultados apropriados (TECHOPEDIA, 2019a). O trabalho de Krizhevsky, Sutskever e Hinton (2012), demonstra como o DL acompanha a VC, uma vez que os autores utilizaram o DL para vencer uma competição de Visão Computacional.

Para o treinamento das máquinas que utilizam redes DL é necessário uma grande quantidade de dados, de centenas de milhares até bilhões. Quando os dados são imagens há uma grande dificuldade em sua obtenção devido a necessidade de que cada uma das imagens seja previamente inspecionada e que sejam delimitados todos os elementos de interesse em cada imagem. Esse conjunto de imagens é utilizado para fazer com que a máquina consiga reconhecer padrões nas imagens de treinamento, e abstrair tais padrões para cenários similares posteriormente.

Um mecanismo atualmente empregado por grandes empresas de tecnologia para a etapa de curadoria das imagens (onde é feita a identificação dos elementos e sua delimitação espacial) é o projeto *recaptcha*, que requer que o usuário clique em imagens que contenham o objeto desejado, como é feito pelo Google, que utiliza o *recaptcha* como ferramenta padrão para autenticação em duas etapas (GOOGLE, 2019).

Atualmente não existe uma base de dados com placas de trânsito brasileiras, como existe, por exemplo, a base de dados belga da Visics (2009) ou a alemã da Stallkamp et al. (2011), que são utilizadas geralmente para testes e validações de reconhecimento de objetos, mas não possuem o mesmo valor para desenvolvimento no nosso país, uma vez que as placas são muito diferentes. Existe também a base de dados COCO, que segundo COCO (2018) possui mais de 120000 imagens para treinos, pertencendo a 80 classes, todas com marcações manuais, mas nem todas as imagens são referentes ao trânsito em geral, apenas algumas classes.

A partir dos dados apresentados surge a questão de pesquisa: É possível criar uma base de dados com alguns elementos característicos do Brasil, para que futuramente possa contribuir e/ou ser utilizada para o treinamento e detecção de veículos autônomos?

1.1 Justificativa e Problema

Este trabalho justifica-se pela necessidade de uma base de treinamento com alguns elementos de transito mais comuns no Brasil. Uma das formas de avaliação é o *Deep Learning*. As redes neurais que utilizam o DL reduzem o tempo dos algoritmos de processamento de imagens, que são algoritmos com um alto custo computacional, de acordo com Lecun et al. (1998) e LeCun, Bengio e Hinton (2015).

Este trabalho investiga se é possível criar uma base de dados com alguns elementos do trânsito brasileiro, para que possa contribuir e/ou ser utilizada para o treinamento e detecção de veículos autônomos no futuro.

1.2 Objetivos

Para resolver o problema proposto, o objetivo geral deste trabalho é adquirir e criar imagens, além de catalogá-las, para a criação de uma base de dados e validá-las com técnicas de *Deep Learning*.

Também, objetivam-se mais especificamente:

1. Criar uma base de dados diversificada com os objetos a serem detectados;
2. Treinar uma Rede Neural Artificial(RNA), baseada na Rede Neural Convolucional ou *Convolutional Neural Network* (CNN);
3. Testar o treinamento com imagens e vídeos para que seja possível avaliar a precisão e o desempenho;
4. Validar o projeto através de analises quantitativas.

2 Revisão bibliográfica

O principal objetivo deste capítulo é apresentar os fundamentos teóricos e históricos sobre os quais este trabalho é executado, além de expor os principais e mais recentes trabalhos em Deep Learning e carros autônomos.

2.1 Fundamentos teóricos

O objetivo deste trabalho é criar uma base de dados com elementos característicos do trânsito brasileiro, que futuramente possa ser utilizada e/ou contribuir para treinamento de veículos autônomos. Ou seja, o objetivo é adquirir e/ou criar imagens para catalogação e validação, que sejam relevantes para o contexto do trânsito brasileiro. Para isso, foram detalhados nos próximos sub-tópicos conceitos teóricos, com o intuito de embasar e nortear o desenvolvimento do mesmo.

2.1.1 Redes Neurais Artificiais

O final da década de 80 marcou o ressurgimento da área de Redes Neurais Artificiais (RNAs), também conhecida como conexionismo ou sistemas de processamento paralelo e distribuído. Esta forma de computação não-algorítmica é caracterizada por sistemas que em algum nível, relembram a estrutura do cérebro humano. Por não ser baseada em regras ou programas, a computação neural se constitui em uma alternativa à computação algorítmica convencional. RNAs são sistemas paralelos distribuídos compostos por unidades de processamento simples (nodos) que calculam determinadas funções matemáticas (normalmente não-lineares). Tais unidades são dispostas em unia ou mais camadas e interligadas por um grande número de conexões, geralmente unidirecionais. Na maioria dos modelos estas conexões estão associadas a pesos, os quais armazenam o conhecimento representado no modelo e servem para ponderar a entrada recebida por cada neurônio da rede. O funcionamento destas redes é inspirado em uma estrutura física concebida pela natureza: o cérebro humano. A solução de problemas através de RNAs é bastante atrativa, já que a forma conto estes são representados internamente pela rede e o paralelismo natural inerente à arquitetura das RNAs criam a possibilidade de um desempenho superior ao dos modelos convencionais. Em RNAs, o procedimento usual na solução de problemas passa inicialmente por uma fase de aprendizagem, em que um conjunto de exemplos é apresentado para a rede, a qual extrai automaticamente as características necessárias para representar a informação fornecida. Estas características são utilizadas posteriormente para gerar respostas para o problema. As RNA's possuem uma grande capacidade de aprender através de exemplos e de generalizar a informação aprendida é, sem dúvida , o atrativo principal da solução de problemas através de RNAs. A generalização , que está associada à capacidade de a rede aprender através de um conjunto reduzido de exemplos e posteriormente dar respostas coerentes para dados não-conhecidos, é uma demonstração de que a capacidade das RNAs vai muito além do que

simplesmente mapear relações de entrada e saída. As RNAs são capazes de extrair informações não-apresentadas de forma explícita através dos exemplos. Não obstante, as RNAs são capazes de atuar como mapeadores universais de funções multivariáveis, com custo computacional que cresce apenas linearmente com o número de variáveis. Outra característica importante é a capacidade de auto-organização e de processamento temporal, que, aliada àquelas citadas anteriormente, faz das RNAs uma ferramenta computacional extremamente poderosa e atrativa para a solução de problemas complexos (BRAGA; CARVALHO; LUDERMIR, 2000).

2.1.2 OpenCV

OpenCV é uma biblioteca de visão computacional de código aberto. A biblioteca é escrita em C e C ++ e é executada no Linux, *Windows*, *Mac OS X*, *iOS*, e *Android*. O *OpenCV* foi projetado para eficiência computacional com um foco em aplicações de tempo real: otimizações foram feitas em todos os níveis, desde algoritmos até multi-núcleos e instruções de *CPU*. Por exemplo, O *OpenCV* suporta otimizações para *SSE*, *MMX*, *AVX*, *NEON*, *OpenMP* e *TBB*. Também é possível adquirir Primitivas de desempenho (IPP), que fazem parte de um pacote de otimização das arquiteturas Intel [Intel] para processamento básico de imagens, que consistem em rotinas otimizadas de baixo nível em muitos diferentes áreas algorítmicas. O *OpenCV* usa automaticamente as instruções apropriadas do IPP no tempo de execução. O módulo *GPU* também fornece versões aceleradas por *CUDA* de muitas rotinas (para *GPU's Nvidia*) e as otimizações para *OpenCL* (para *GPU's* genéricas). Um dos objetivos da *OpenCV* é fornecer uma infraestrutura de visão computacional simples de usar que ajude as pessoas a construir aplicações de visão razoavelmente sofisticadas rapidamente. A biblioteca *OpenCV* contém mais de 500 funções que abrangem muitas áreas, incluindo inspeção de produtos de fábricas, imagens médicas, segurança, interface de usuário, câmera calibração, visão estereoscópica e robótica. Como a visão computacional e o aprendizado de máquina costumam estar juntos, o *OpenCV* também contém uma biblioteca de Aprendizado de Máquina (*MLL*) completa e de uso geral. Esta sub-biblioteca é focado no reconhecimento de padrões estatísticos e agrupamento. O *MLL* é altamente útil para as tarefas de visão que fazem parte da ideia geral da *OpenCV*, mas ela é geral o suficiente para ser usada em qualquer problema de aprendizado de máquina (BRADSKI; KAEHLER, 2013).

2.1.3 CUDA

CUDA é uma plataforma de computação paralela e modelo de programação desenvolvido pela *NVIDIA* para computação geral em unidades de processamento gráfico (*GPUs*). Com o *CUDA*, os desenvolvedores podem acelerar drasticamente os aplicativos de computação, aproveitando o poder das *GPU's*. Em aplicativos acelerados por *GPU*, a parte sequencial da carga de trabalho é executada na *CPU* - que é otimizada para desempenho de encadeamento único - enquanto a parte de computação intensiva do aplicativo é executada em milhares de núcleos de *GPU* em paralelo. Ao usar o CUDA, os desenvolvedores programam em linguagens populares como C, C ++, Fortran, Python e MATLAB e expressam paralelismo através de extensões na forma de algumas palavras-chave básicas. O *CUDA Toolkit* da *NVIDIA* fornece tudo o que você precisa para desenvolver aplicativos acelerados por *GPU*. O *CUDA* Toolkit

inclui bibliotecas aceleradas por *GPU*, um compilador, ferramentas de desenvolvimento e o tempo de execução *CUDA* (NVIDIA, 2018a).

2.1.4 cuDNN

A biblioteca *NVIDIA CUDA Deep Neural Network* (cuDNN) é uma biblioteca de primitivos acelerada por *GPU* para redes neurais profundas (DNN). O *cuDNN* fornece implementações altamente ajustadas para rotinas padrão, como camadas de convolução para frente e para trás (*forward and backward*), agrupamento (*pooling*), normalização e ativação. Pesquisadores de *Deep Learning* e desenvolvedores de *framework* utilizam o *cuDNN* para aceleração de *GPU* de alto desempenho. Isso permite que eles se concentrem no treinamento de redes neurais e no desenvolvimento de aplicativos de software, em vez de gastar tempo com o ajuste de desempenho da *GPU* de baixo nível. O *cuDNN* acelera as estruturas de aprendizagem profunda amplamente usadas, incluindo o Caffe, o Caffe2, o Chainer, o Keras, o MATLAB, o MxNet, o TensorFlow e o PyTorch (NVIDIA, 2018b).

2.1.5 Blender 3D

Blender é um pacote de criação 3D gratuito e de código aberto. Ele suporta modelagem, manipulação, animação, simulação, renderização, composição e rastreamento de movimento, até mesmo edição de vídeo e criação de jogos. Usuários mais avançados utilizam a *API* do *Blender* para criação de *scripts* em Python para personalizar o aplicativo e escrever ferramentas especializadas. O *Blender* é bem adequado para pessoas e pequenos estúdios que se beneficiam de seu pipeline unificado e processo de desenvolvimento responsivo. *Blender* é multi-plataforma e funciona em computadores *Linux*, *Windows* e *Macintosh*. Sua interface usa o *OpenGL* para fornecer uma experiência consistente. Como um projeto conduzido pela comunidade sob a Licença Pública Geral GNU (GPL), os usuários tem o poder de fazer pequenas e grandes mudanças na base de código, o que leva a novos recursos, correções de erros responsivas e melhor usabilidade (BLENDER.ORG, 2019).

2.1.6 Unreal Engine 4

O Unreal Engine 4 é um conjunto completo de ferramentas de desenvolvimento criadas para qualquer pessoa que trabalhe com tecnologia em tempo real. Desde aplicativos corporativos e experiências cinematográficas até jogos de alta qualidade em PC, console, dispositivos móveis, VR e AR, o Unreal Engine 4 oferece tudo o que você precisa para começar. Um conjunto de ferramentas de classe mundial e fluxos de trabalho acessíveis capacitam os desenvolvedores a interagir rapidamente em ideias e ver resultados imediatos sem tocar em uma linha de código, enquanto o acesso ao código-fonte completo dá a todos na comunidade do Unreal Engine 4 a liberdade de modificar e estender os recursos do mecanismo (EPIC, 2019).

2.1.7 Deep Learning

Como descrito em 2.1.1, as redes neurais artificiais possuem a capacidade de aprender através de exemplos e generalizar as informações aprendidas. A partir das RNA's foi desenvolvidos estudos e recentemente chegou-se ao *Deep Learning*, conhecido como aprendizado profundo e uma parte desses estudos estão descritos abaixo.

> "Nos últimos anos, as técnicas de aprendizado de máquina, particularmente quando aplicadas às RNA's, desempenharam um papel cada vez mais importante no projeto de sistemas de reconhecimento de padrões. De fato, pode-se argumentar que a disponibilidade de técnicas de aprendizado tem sido um fator crucial no sucesso recente de aplicativos de reconhecimento de padrões, como reconhecimento de fala e reconhecimento de caligrafia contínuos". (LECUN et al., 1998, p.2278)

De acordo com Krizhevsky, Sutskever e Hinton (2012), as abordagens atuais para o reconhecimento de objetos fazem uso essencial dos métodos de aprendizado de máquina. Para melhorar seu desempenho, podemos coletar conjuntos de dados maiores, aprender modelos mais poderosos e usar técnicas melhores para evitar o *sobreajuste*. Até recentemente, os conjuntos de dados de imagens rotuladas eram relativamente pequenos - na ordem de dezenas de milhares de imagens (por exemplo, NORB, Caltech-101/256 e CIFAR-10/100). Tarefas de reconhecimento simples podem ser resolvidas muito bem com conjuntos de dados desse tamanho, especialmente se eles são aumentados com transformações de preservação de rótulo. Por exemplo, a taxa de erro atual mais alta na tarefa de reconhecimento de dígitos MNIST (<0.3%) se aproxima do desempenho humano. Mas objetos em ambientes realistas exibem considerável variabilidade, então, para aprender a reconhecê-los, é necessário usar conjuntos de treinamento muito maiores. E, de fato, as deficiências de pequenos conjuntos de dados de imagens têm sido amplamente reconhecidos, mas somente recentemente foi possível coletar conjuntos de dados rotulados com milhões de imagens. Os novos conjuntos de dados maiores incluem o LabelMe, que consiste em centenas de milhares de imagens totalmente segmentadas, e ImageNet, que consiste em mais de 15 milhões de imagens de alta resolução rotuladas em mais de 22.000 categorias.

O Deep Learning permite que modelos computacionais compostos de múltiplas camadas de processamento, possam aprender representações de dados com múltiplos níveis de abstração. Esses métodos melhoraram drasticamente o estado da arte em reconhecimento de fala, objetos visuais, detecção de objetos e muitos outros domínios, como descoberta de drogas e genômica. Deep Learning revela estruturas complexas em grandes conjuntos de dados usando o algoritmo backpropagation para indicar como uma máquina deve alterar seus parâmetros internos que são usados para calcular a representação em cada camada de representação, e na camada anterior. Redes convolucionais profundas trouxeram avanços no processamento de imagens, vídeo, fala e áudio, enquanto as redes recorrentes esclareceram dados sequenciais, como texto e fala. (LECUN; BENGIO; HINTON, 2015).

2.1.8 Redes Neurais Convolucionais

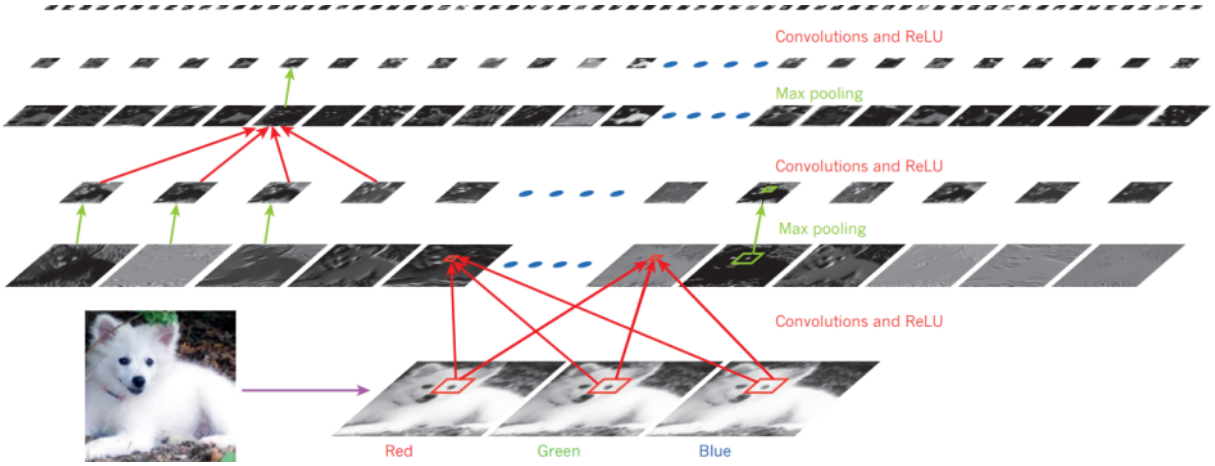
A capacidade de redes multicamadas treinadas com gradiente descendente para aprender mapeamentos complexos, não lineares e de alta dimensão a partir de grandes conjuntos de exemplos torna-os candidatos óbvios para tarefas de reconhecimento de imagem. No modelo tradicional de reconhecimento de padrões, um extrator de recursos desenhado à mão reúne informações relevantes da entrada e elimina as variabilidades irrelevantes. Um classificador treinável então categoriza os vetores de característica resultantes em classes. Neste esquema, as redes multicamadas padrão totalmente conectadas podem ser usadas como classificadores (LECUN et al., 1998).

> "As *ConvNets* são projetadas para processar dados que vêm na forma de múltiplos *arrays*, por exemplo, uma imagem colorida composta por três matrizes contendo intensidades de pixel nos três canais de cores. Muitas modalidades de dados estão na forma de múltiplos *arrays*: 1D para sinais e sequências, incluindo a linguagem; 2D para imagens ou espectrogramas de áudio;e 3D para vídeo ou imagens volumétricas. Existem quatro ideias-chave atrás de *ConvNets* que aproveitam as propriedades dos recursos naturais sinais: conexões locais, pesos compartilhados, *pooling* e o uso de muitas camadas."(LECUN; BENGIO; HINTON, 2015, p.439-440)

A arquitetura de uma *Rede Convolucional* típica, como demonstrado na figura 1, é estruturada como um série de etapas. Os primeiros estágios são compostos por dois tipos de camadas: camadas convolucionais e camadas de agrupamento. Unidades em uma camada convolucional são organizadas em mapas de características, dentro do qual cada unidade está conectado a fragmentos locais nos mapas de recursos da camada anterior, através de um conjunto de pesos chamado um banco de filtros. O resultado dessa soma ponderada local é então passada através de uma não-linearidade, como um *ReLU*. Todas as unidades em um mapa de recursos compartilham o mesmo banco de filtros. Mapas de recursos diferentes em uma camada usam diferentes bancos de filtros. O motivo dessa arquitetura é duplicado. Primeiro, em dados de matriz, como imagens, locais grupos de valores são frequentemente altamente correlacionados, formando motivos que são facilmente detectados. Em segundo lugar, as estatísticas locais de imagens e outros sinais são invariantes à localização. Em outras palavras, se um motivo pode aparecer em uma parte da imagem, ela pode aparecer em qualquer lugar, a ideia de unidades em diferentes locais compartilhando os mesmos pesos e detectando o mesmo padrão em diferentes partes da matriz. Matematicamente, a operação de filtragem executada por um mapa de recursos é uma convolução, daí o nome. Embora o papel da camada convolucional seja detectar conjunções locais de características da camada anterior, o papel do conjunto camada é mesclar recursos semanticamente semelhantes em um. Porque a posição relativa das características que formam um motivo podem variar um pouco, a detecção confiável do motivo pode ser feita grosseiramente a posição de cada característica. Uma unidade típica de *pool* calcula o máximo de um fragmento local de unidades em um mapa de recursos (ou em alguns mapas de recursos). Unidades de *pool* vizinhas recebem entradas de fragmentos que são deslocados por mais de uma linha ou coluna, reduzindo assim a dimensão da representação e criando uma invariância a pequenas mudanças e distorções. Dois ou três estágios de convolução, não linearidade e agrupamento são empilhados, seguidos por mais

convolucional e camadas totalmente-conectadas. Gradientes de *backpropagating* através de um *ConvNet* é tão simples quanto através de uma *Deep Net* regular, permitindo que todos os pesos em todos os bancos de filtros sejam treinados. (LECUN; BENGIO; HINTON, 2015).

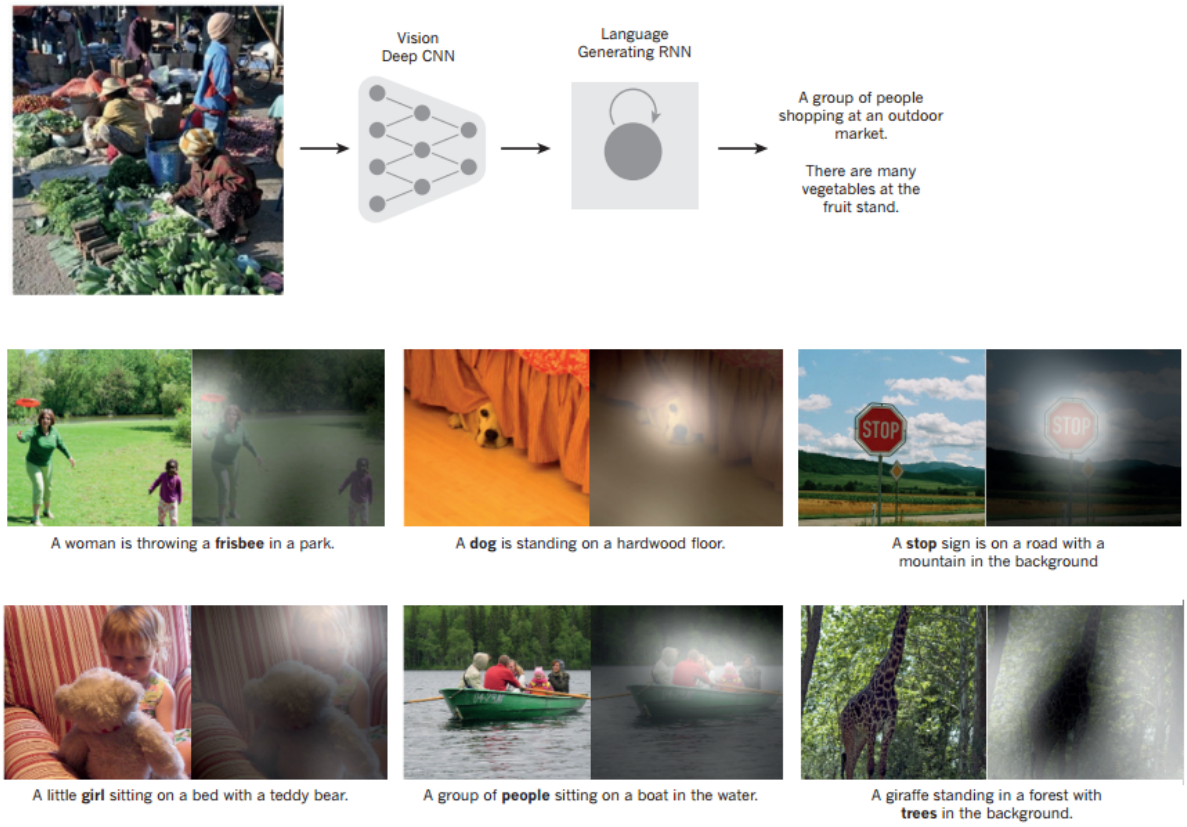
Figura 1 – Exemplo de uma camada de convolução.



Fonte: (LECUN; BENGIO; HINTON, 2015).

Ainda de acordo com LeCun, Bengio e Hinton (2015), desde o início dos anos 2000, as *ConvNets* têm sido aplicadas com grande sucesso a detecção, segmentação e reconhecimento de objetos e regiões em imagens. Estas foram todas tarefas em que os dados rotulados eram relativamente abundantes, como reconhecimento de sinais de trânsito, segmentação de imagens especialmente para *conectomics*, e a detecção de faces, texto, pedestres e corpos humanos em imagens naturais. Um importante sucesso prático recente das *ConvNets* é o reconhecimento facial. Importante, as imagens podem ser rotuladas no nível de pixel, que terá aplicações em tecnologia, incluindo robôs móveis autônomos e carros autônomos. Empresas como Mobileye e NVIDIA estão usando esses métodos baseados em *ConvNet* em seus futuros sistemas de visão para carros. Outras aplicações que ganham importância envolvem compreensão da linguagem e reconhecimento de fala. Apesar destes sucessos, os*ConvNets* ConvNets foram largamente abandonados pela comunidades tradicionais de visão computacional e aprendizagem de máquina até a competição ImageNet em 2012. Quando CNN's profundas foram aplicadas a um conjunto de dados de cerca de um milhão de imagens da web que continha 1.000 classes diferentes, eles alcançaram resultados espetaculares, quase reduzindo pela metade as taxas de erro das melhores abordagens concorrentes. Esse sucesso veio do uso eficiente de GPUs, ReLUs, uma nova técnica de regularização chamada *dropout*, e técnicas para gerar mais exemplos de treinamento, deformando os já existentes. Este sucesso trouxe uma revolução na visão computacional; *ConvNets* são agora a abordagem dominante para quase todo o reconhecimento tarefas de detecção e abordagem e desempenho humano em algumas tarefas. Uma recente demonstração impressionante combina *ConvNets* e módulos líquidos recorrentes para a geração de legendas de imagens como demonstrado na figura 2.

Figura 2 – Legendas geradas por uma Rede Neural Recorrente(RNN), utilizando uma uma CNN.



Fonte: (LECUN; BENGIO; HINTON, 2015).

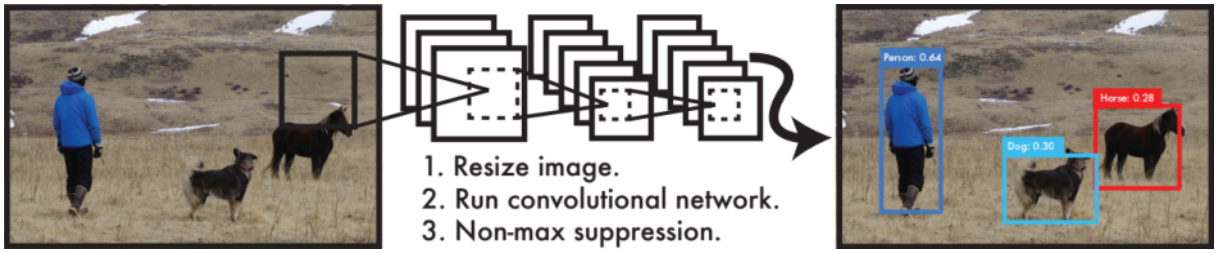
2.1.9 Yolo - You Only Look Once

De acordo com Redmon et al. (2015), o desenvolvimento de algoritmos rápidos e precisos para a detecção de objetos permitiriam que os computadores conduzissem carros sem sensores especializados, permitiriam que dispositivos auxiliares transmitissem informações de cena em tempo real para usuários humanos e liberassem o potencial para sistemas robóticos responsivos e de uso geral. Os sistemas de detecções atuais redirecionam os classificadores para realizar a detecção. Para detectar um objeto, esses sistemas pegam um classificador para esse objeto e o avaliam em vários locais e escalas em uma imagem de teste. Sistemas como modelos de peças deformáveis (DPM) usam uma abordagem de janela deslizante em que o classificador é executado em locais com espaçamento uniforme em toda a imagem como, por exemplo, o trabalho de Sermanet et al. (2013).

Ainda de acordo com Redmon et al. (2015), abordagens mais recentes, como a Regional Based Convolutional Neural Network (R-CNN), usam métodos de proposta de região para gerar primeiro caixas delimitadoras potenciais em uma imagem e, em seguida, executar um classificador nessas caixas propostas. Após a classificação, o pós-processamento é usado para refinar as caixas delimitadoras, eliminar detecções duplicadas e recodificar as caixas com base em outros objetos na cena. Esses pipelines complexos são lentos e difíceis de otimizar,

porque cada componente individual deve ser treinado separadamente. Baseado nestes fatos, a detecção de objetos foi reformulada como um problema de regressão único, desde os pixels da imagem até as coordenadas das caixas delimitadoras e as probabilidades das classes. Utilizando o sistema *you only look once* (YOLO) em uma imagem para prever quais objetos estão presentes e onde estão localizados. O funcionamento do sistema pode ser visto na figura 3, onde uma única rede convolucional prevê simultaneamente várias caixas delimitadoras e probabilidades de classe para essas caixas. A YOLO treina em imagens completas e otimiza diretamente o desempenho da detecção. Este modelo unificado tem vários benefícios sobre os métodos tradicionais de detecção de objetos.

Figura 3 – Funcionamento do sistema de detecção de YOLO de acordo com seu criador.



Fonte: (REDMON et al., 2015).

Como YOLO teve moldada sua detecção como um problema de regressão, não foi preciso um pipeline complexo. Simplesmente executando a rede neural em uma nova imagem em tempo de teste para prever detecções. A rede básica é executada a 45 quadros por segundo, sem processamento em lote em uma GPU Titan X, e uma versão rápida é executada em mais de 150 fps. Isso significa que é possível processar o streaming de vídeo em tempo real com menos de 25 milissegundos de latência (mais de 30 fps). Além disso, a YOLO alcança mais que o dobro da precisão média de outros sistemas em tempo real. A YOLO raciocina globalmente sobre a imagem ao fazer previsões. Ao contrário das técnicas baseadas em proposta de janela deslizante e região, a YOLO vê a imagem inteira durante o treinamento e o tempo de teste para codificar implicitamente informações contextuais sobre as classes, bem como sua aparência. O Fast R-CNN, um método de detecção superior, confunde partes plano de fundo em uma imagem para objetos porque não consegue ver o contexto maior. A YOLO faz menos da metade do número de erros de segundo plano em comparação com o Fast R-CNN. A YOLO aprende representações generalizáveis de objetos. Quando treinada em imagens naturais e testada em obras de arte, a YOLO supera em muito os melhores métodos de detecção, como DPM e R-CNN. Como o YOLO é altamente generalizável, é menos provável que ocorra um erro quando aplicado a novos domínios ou a entradas inesperadas.

2.2 Fundamentos históricos

Diversos trabalhos foram encontrados durante a etapa de levantamento bibliográfico, através de um curso online, feito pelo autor, sobre *Deep Learning* oferecido pela empresa

Coursera[1]. Outros trabalhos foram encontrados através de buscas no portal de periódicos da CAPES.

2.2.1 Carros autônomos

1. **Driverseat: Crowdstrapping Learning Tasks for Autonomous Driving:** O artigo de Rajpurkar et al. (2015) apresenta o *Driverseat*, uma tecnologia para incorporar multidões em torno de sistemas de aprendizagem para condução autônoma. *Driverseat* utiliza as contribuições do *crowdstrapping* para (a) coleta de etiquetas 3D complexas e (b) marcação diversos cenários para pronta avaliação de sistemas de aprendizagem. Foi demonstrado como o *Driverseat* pode fazer um *crowdstrap* em uma rede neural convolucional para a tarefa de detecção de faixa. De uma forma mais geral, o *crowdstrapping* introduz um paradigma valioso para qualquer tecnologia que possa se beneficiar da poderosa combinação do ser humano e a inteligência computacional. Como resultado foi avaliado o desempenho do sistema na tarefa de detecção de faixa em uma matriz de diferente cenários marcados com a interface *TagEval* da *Driverseat*. A avaliação qualitativa do desempenho da rede nestes cenários revelam pontos fortes e dificuldades para a rede. Por um lado, as rampas de acesso são desafiadoras para a rede, e as pistas emergentes são muitas vezes perdidas, sombras e mudanças de pavimento são modestamente difíceis também, e a rede torna-se suscetível a interpretar incorretamente manchas refletivas na estrada com marcas de faixa. Por outro lado, a rede é robusta às curvas nas rodovias e é capaz de detectar os limites das faixas mesmo presença de veículos ocultando-as. A avaliação quantitativa fornece mais detalhes. Para a detecção da fronteira da *ego-lane*, notamos que o desempenho nas curvas da estrada, embora comparável com as desempenho, diminui rapidamente com a distância sombras e mudanças no pavimento são as condições mais desafiadoras para a detecção da *ego-lane*. Representando graficamente os desempenhos nas pistas adjacente à *ego-lane* em ambos os lados, é identificado que as saídas de acesso prejudicam o desempenho de detecção na faixa de divisão, e as sombras continuam sendo um cenário desafiador.

2. **An Empirical Evaluation of Deep Learning on Highway Driving:** O artigo de Huval et al. (2015), apresenta uma série de avaliações empíricas de avanços recentes em *Deep Learning*. A visão computacional, combinada com o *Deep Learning*, tem o potencial de trazer uma solução relativamente barata e robusta para a direção autônoma. Para preparar o *Deep Learning* para o uso da indústria e aplicações práticas, as redes neurais exigirão grandes conjuntos de dados que representam todos os possíveis ambientes e cenários de direção. Foi coletado um grande conjunto de dados de dados de rodovias, e aplicados à algoritmos de *Deep Learning* e de visão computacional, como problemas de detecção de carros e pistas. Por fim é mostrado como as redes neurais convolucionais (CNN's) existentes podem ser usadas para realizar a detecção de pista e veículo durante a execução em taxas de quadros necessárias para um sistema em tempo real.

[1] Link disponível para o curso https://www.coursera.org/specializations/deep-learning

Os resultados dão credibilidade à hipótese de que a *Deep Learning* é promissora para a condução autônoma.

2.2.2 Deep Learning

1. **Gradient-Based Learning Applied to Document Recognition:** O artigo de Lecun et al. (1998), um dos pioneiros na área, descreve o funcionamento redes neurais multicamadas treinadas com o algoritmo de propagação reversa, que constituem o melhor exemplo de uma técnica de aprendizado baseada em gradiente bem-sucedida. Dada uma arquitetura de rede apropriada, os algoritmos de aprendizado baseados em gradiente podem ser usados para sintetizar uma superfície de decisão complexa que pode classificar padrões de alta dimensionalidade, como caracteres manuscritos, com pré-processamento mínimo. É feita uma analise de vários métodos aplicados ao reconhecimento de caracteres manuscritos e compara-os em uma tarefa padrão de reconhecimento de dígitos manuscritos. Redes neurais convolucionais, que são especificamente projetadas para lidar com a variabilidade de formas bidimensionais (2-D), são utilizadas para superar todas as outras técnicas. Os sistemas de reconhecimento de documentos da vida real são compostos de vários módulos, incluindo extração de campo, segmentação, reconhecimento e modelagem de linguagem. Um novo paradigma de aprendizado, chamado de redes de transformadores de grafos (GTNs), permite que esses sistemas *multimodulares* sejam treinados globalmente usando métodos baseados em gradiente, de modo a minimizar uma medida de desempenho global. Dois sistemas para reconhecimento de manuscrito online são descritos e as experiências demonstram a vantagem do treinamento global e a flexibilidade das redes de transformadores de grafos. E finalmente, uma *GTN* também é utilizada para demonstrar como é possível ler um cheque bancário, ela usa reconhecedores de caracteres de rede neural convolucional combinados com técnicas de treinamento global para fornecer precisão de registro em cheques corporativos e pessoais, ela é implantada comercialmente e lê vários milhões de cheques por dia.

2. **ImageNet Classification with Deep Convolutional Neural Networks:** No artigo de Krizhevsky, Sutskever e Hinton (2012), os autores treinaram uma rede neural convolucional grande e profunda para classificar as 1,2 milhão de imagens de alta resolução no concurso *ImageNet LSVRC-2010* nas 1000 classes diferentes. Nos dados de teste, conseguiram atingir as taxas de erro top-1 e top-5 de 37,5% e 17,0%, o que é consideravelmente melhor do que o estado-da-arte anterior. A rede neural, que tem 60 milhões de parâmetros e 650.000 neurônios, consiste em cinco camadas convolucionais, algumas das quais são seguidas por camadas *max-pooling* e três camadas totalmente conectadas com um *softmax* final de 1000 vias. Para tornar o treinamento mais rápido, foram utilizados neurônios não saturantes e uma implementação de GPU muito eficiente da operação de convoluções. Para reduzir o *overfitting* nas camadas totalmente conectadas, foi empregado um método de regularização recentemente desenvolvido, chamado *"dropout"*, que provou ser muito eficaz. Como resultados, também foi inserimos uma variante deste modelo na competição *ILSVRC-2012* e foi obtido uma taxa de erro de teste

top-5 vencedora de 15,3%, em comparação com 26,2% alcançados pela segunda melhor entrada.

3. **Deep learning for class-generic object detection:** O artigo de Huval, Coates e Ng (2013) investigou o uso de *Deep Neural Networks* para a nova tarefa de detecção de objetos de classes genéricas. É demonstrado que as redes neurais originalmente projetadas para o reconhecimento de imagens podem ser treinadas para detectar objetos dentro de imagens, independentemente de sua classe, incluindo objetos para os quais nenhum rótulo de caixa delimitadora foi fornecido. Além disso, é também demonstrado que os rótulos de caixa delimitadora geram um aumento de desempenho de 1% no desafio de reconhecimento do *ImageNet*.

4. **Visualizing and Understanding Convolutional Networks:** No artigo de Zeiler e Fergus (2013) é visto que, grandes modelos da Convolutional Network demonstraram recentemente um impressionante desempenho de classificação no benchmark *ImageNet* (KRIZHEVSKY; SUTSKEVER; HINTON, 2012). No entanto, não há uma compreensão clara de por que eles funcionam tão bem ou como podem ser melhorados. Foi abordado os dois problemas e também foi introduzido uma nova técnica de visualização que fornece informações sobre a função das camadas intermediárias de recursos e a operação do classificador. As mesmas foram utilizadas em uma função de diagnóstico, essas visualizações os permitiram encontrar arquiteturas de modelos que superam Krizhevsky, Sutskever e Hinton (2012) no benchmark de classificação do *ImageNet*. Também foi realizado um estudo de separação para descobrir a contribuição de desempenho de diferentes camadas de modelo. E por ultimo é mostrado que o modelo *ImageNet* é generalizado para outros conjuntos de dados: quando o classificador *softmax* é treinado novamente, ele bate de forma convincente os resultados atuais de última geração nos conjuntos de dados *Caltech-101* e *Caltech-256*.

5. **OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks:** No presente artigo, de Sermanet et al. (2013), é apresentada uma estrutura integrada para o uso de Redes Convolucionais para classificação, localização e detecção. A estrutura é mostrada como uma abordagem de multi-escalar e de janela deslizante pode ser implementada de maneira eficiente em uma rede neural convolucional. Também é introduzido uma nova abordagem de *Deep Learning* para localização, aprendendo a prever limites de objetos. Caixas delimitadoras são acumuladas ao invés de serem suprimidas para aumentar a confiança na detecção. É demonstrado que diferentes tarefas podem ser aprendidas simultaneamente usando uma única rede compartilhada. Esta estrutura integrada é a vencedora da tarefa de localização do Desafio de Reconhecimento Visual de Grande Escala *ImageNet* 2013 (ILSVRC2013) e obteve resultados muito competitivos para as tarefas de detecção e classificação. No trabalho pós-competição, foi estabelecido um novo estado da arte para a tarefa de detecção. E finalmente, foi lançado um extrator de recursos do nosso melhor modelo chamado *OverFeat*.

6. **A Style-Based Generator Architecture for Generative Adversarial Networks:** No artigo de Karras, Laine e Aila (2018) é proposta uma arquitetura de geradores alternativos

para redes geradoras de adversários (RGA), pegando emprestado da literatura de transferência de estilo. A nova arquitetura gera um aprendizado automático de uma separação não-supervisionada, de atributos de alto nível (por exemplo, pose e identidade quando treinados em rostos humanos) e variação estocástica nas imagens geradas (por exemplo, sardas, cabelos) e possibilita o controle específico da síntese com escalabilidade intuitiva. O novo gerador melhora o estado da arte em termos de métricas tradicionais de qualidade de distribuição, demonstra também melhores propriedades de interpolação e também decifra melhor os fatores latentes de variação. Para quantificar a qualidade de decifrar e interpolar, é proposto dois novos métodos automatizados que são aplicáveis a qualquer arquitetura de gerador. Por fim, é apresentado um novo conjunto de dados de faces humanas altamente variado e de alta qualidade.

2.2.3 Técnicas de Deep Learning

1. **Deep Residual Learning for Image Recognition:** O artigo de He et al. (2015) apresenta redes neurais mais profundas, que são mais difíceis de treinar. É Apresentado uma estrutura de aprendizagem residual para facilitar o treinamento de redes que são substancialmente mais profundas do que as usadas anteriormente. Nas redes, foram reformuladas explicitamente as camadas para aprender funções residuais com referência às camadas de entrada, em vez de aprender funções não referenciadas. Foram fornecidas evidências empíricas abrangentes, mostrando que essas redes residuais são mais fáceis de otimizar e podem ganhar precisão a partir de uma profundidade consideravelmente maior. No banco de dados do *ImageNet*, foram avaliadas redes residuais com uma profundidade de até 152 camadas, que é 8 vezes mais profundo que as redes *VGG*, mas ainda com menor complexidade. Um conjunto dessas redes residuais atinge um erro de 3,57% no conjunto de testes do *ImageNet*. Este resultado conquistou o 1º lugar na tarefa de classificação do *ILSVRC 2015*. Também foi apresentado uma análise do*CIFAR-10* com 100 e 1000 camadas. A profundidade das representações foi de importância central para muitas tarefas de reconhecimento visual. Apenas devido a essas representações extremamente profundas, foram obtidas melhorias relativas de 28% no conjunto de dados de detecção de objetos *COCO*. *Deep Residual Networks* foram as bases das submissões para as competições *ILSVRC* & *COCO 2015*, onde também conseguiram ganhar os primeiros lugares nas tarefas de detecção de *ImageNet*, localização de *ImageNet*, detecção de *COCO* e segmentação de *COCO*.

2. **You Only Look Once: Unified, Real-Time Object Detection:** O artigo de Redmon et al. (2015) apresenta a *YOLO*, uma nova abordagem para detecção de objetos. O autor, em trabalho anterior na detecção de objetos, reaproveita os classificadores para realizar a detecção. No atual, em vez disso, é enquadrado a detecção de objetos como um problema de regressão para caixas delimitadoras espacialmente separadas e probabilidades de classe associadas. Uma única rede neural prevê caixas delimitadoras e probabilidades de classe diretamente a partir de imagens completas em uma avaliação. Como todo o pipeline de detecção é uma única rede, ele pode ser otimizado de ponta a ponta

diretamente no desempenho da detecção. A arquitetura unificada desenvolvida é extremamente rápida. O modelo base *YOLO* processa imagens em tempo real a 45 quadros por segundo. Uma versão menor da rede, a *Fast YOLO*, processa impressionantes 155 quadros por segundo enquanto ainda alcança o dobro do *mAP* de outros detectores em tempo real. Em comparação com sistemas de detecção de última geração, a *YOLO* faz mais erros de localização, mas é menos provável que ele preveja os falsos positivos em segundo plano. Finalmente, a *YOLO* aprende representações muito gerais de objetos. Ele supera outros métodos de detecção, incluindo *DPM* e *Region-Covolutional Neural Network*, ao generalizar de imagens naturais para outros domínios, como obras de arte.

3. **YOLO9000: Better, Faster, Stronger:** O artigo de Redmon e Farhadi (2016) apresenta a *YOLO9000*, um sistema de detecção de objetos em tempo real e avançado que pode detectar mais de 9000 categorias de objetos. Foram propostas diversas melhorias para o método de detecção *YOLO*, ambas novas e extraídas de trabalhos anteriores. O modelo aprimorado, *YOLOv2*, é o estado da arte em tarefas de detecção padrão, como *PASCAL VOC* e *COCO*. Usando um novo método de treinamento em escala múltipla, o mesmo modelo *YOLOv2* pode ser executado em tamanhos variados, oferecendo uma troca fácil entre velocidade e precisão. Em 67 quadros por segundos, a *YOLOv2* recebe 76,8 *mAP* no *VOC* 2007. Em 40 quadros por segundos, a *YOLOv2* obtém 78,6 *mAP*, superando os métodos de última geração, como *R-CNN* mais rápido com *ResNet* e *SSD*, enquanto ainda está operando de maneira significativamente mais rápida. Finalmente, é proposto um método para treinar conjuntamente na detecção e classificação de objetos. Usando esse método, *YOLO9000* foi treinada simultaneamente no conjunto de dados de detecção de *COCO* e no conjunto de dados de classificação do *ImageNet*. O treinamento conjunto permite que o *YOLO9000* preveja detecções para classes de objetos que não tenham dados de detecção rotulados. Foi validada a abordagem na tarefa de detecção do *ImageNet*. A *YOLO9000* obtém 19.7 *mAP* no conjunto de validação de detecção do *ImageNet*, apesar de ter apenas dados de detecção para 44 das 200 classes. Nas 156 classes que não estão no *COCO*, o *YOLO9000* recebe 16.0 *mAP*. Mas a *YOLO* pode detectar mais de 200 classes; prevê detecções para mais de 9000 categorias de objetos diferentes e ainda funciona em tempo real.

4. **YOLOv3: An Incremental Improvement:** No artigo de Redmon e Farhadi (2018) são apresentadas algumas atualizações para a *YOLO*. Foram feitas diversas mudanças de pequeno porte no design para torná-lo melhor. Também treinaram essa nova rede que é descrita pelo próprio autor como "muito boa". É um pouco maior que última, mas mais precisa e mais rápida. Em 320 por 320, a *YOLOv3* é executado em 22 ms a 28,2 *mAP*, tão preciso quanto o *SSD*, mas três vezes mais rápido. Se comparada à antiga métrica de detecção de mP de 0,5 *IOU*, a *YOLOv3* é "muito bom". Ele atinge 57: 9 AP50 em 51 ms em uma *GPU Titan X*, em comparação com 57: 5 AP50 em 198 ms por *RetinaNet*, desempenho semelhante, mas 3,8 vezes mais rápido.

2.2.4 Ambientes de simulação

1. **AirSim - High-Fidelity Visual and Physical Simulation for Autonomous Vehicles:** No artigo de Shah et al. (2017) acredita-se que desenvolver e testar algoritmos para veículos autônomos no mundo real é um processo caro e demorado. Além disso, para utilizar os avanços recentes em inteligência de máquina e *Deep Learning*, é preciso coletar uma grande quantidade de dados de treinamento anotados em uma variedade de condições e ambientes. Então foi criado um novo simulador na *Unreal Engine 4* que oferece simulações físicas e visualmente realistas para esses dois objetivos. O simulador inclui um mecanismo de física que pode operar em alta frequência para simulações de hardware em loop (HITL) em tempo real com suporte para protocolos populares (por exemplo, MavLink). O simulador foi projetado desde o início para ser extensível para acomodar novos tipos de veículos, plataformas de hardware e protocolos de software. Além disso, o design modular permite que vários componentes sejam facilmente utilizados de forma independente em outros projetos. É Demonstrado que o simulador implementando primeiro um quadro-rotor como veículo autônomo e comparando experimentalmente os componentes de software com voos do mundo real.

2. **UE4Sim - A Photo-Realistic Simulator for Computer Vision Applications:** No artigo de Mueller et al. (2017) é apresentado um simulador de treinamento e avaliação foto-realista (UE4Sim) com extensas aplicações em vários campos da visão computacional. Construído com a *Unreal Engine 4*, o simulador integra carros baseados em física completos, veículos aéreos não tripulados (*VANT's*) e atores humanos animados em diversos ambientes urbanos e suburbanos em 3D. É Demonstrado a versatilidade do simulador com dois estudos de caso: rastreamento autônomo baseado em *VANT* de objetos em movimento e condução autônoma usando aprendizado supervisionado. O simulador integra os dois algoritmos de rastreamento de última geração com uma ferramenta de avaliação de benchmark e uma arquitetura de rede neural profunda (DNN) para treinamento de veículos para dirigir de forma autônoma. Ele gera conjuntos de dados foto-realistas sintéticos com anotações automáticas de *ground truth* para ampliar facilmente os conjuntos de dados existentes no mundo real e fornece extensa variedade de dados sintéticos por meio de sua capacidade de reconfigurar cenários virtuais em tempo real usando uma ferramenta automática de geração de cenários.

3 Métodos e materiais

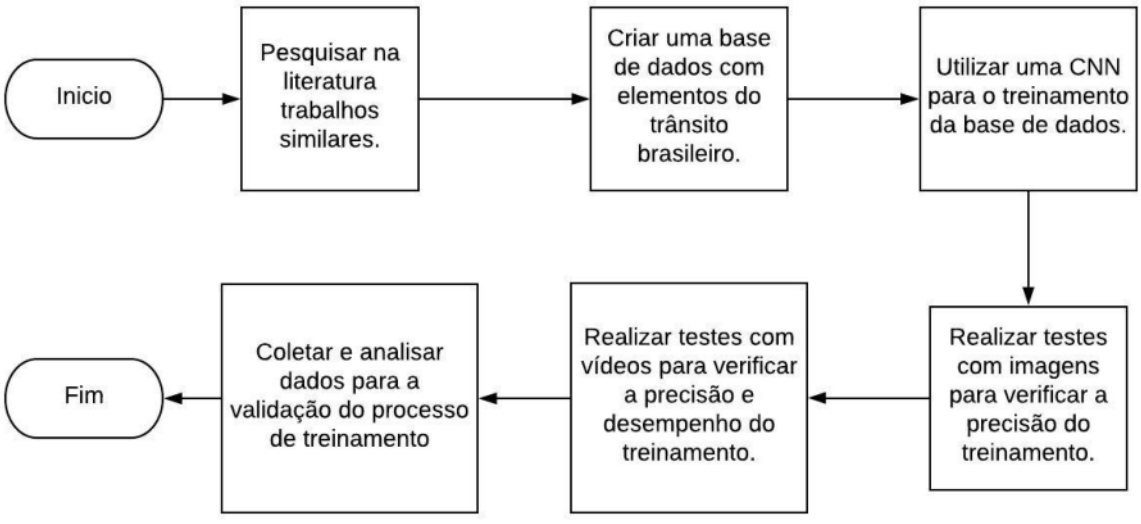
De acordo com Gerhardt e Silveira (2009, p.31-42), a presente pesquisa é descritiva e exploratória do ponto de vista dos objetivos, é descritiva pois é necessário entender e descrever o fenômeno estudado, e exploratória, uma vez que foi fundamental um estudo de caso para explicar o fenômeno; aplicada do ponto de vista de sua natureza, pois busca gerar conhecimento para aplicação prática; quantitativa quanto à abordagem ao problema, uma vez que procura resultados baseados em quantidades para determinar seu êxito; e pode ser classificada como pesquisa experimental quanto os procedimentos, tendo em perspectiva a criação de uma base de dados para usos futuros.

Os métodos são organizados nas seguintes etapas:

1. reunir e estudar os principais trabalhos envolvendo *Deep Learning*;
2. criar a base de dados a ser utilizada;
3. utilizar uma CNN para treinamento da base de dados;
4. realizar testes isolados para verificar a precisão do treinamento;
5. realizar testes com vídeos para verificar a precisão e desempenho do treinamento;
6. coletar e analisar dados para a validação do processo de treinamento.

Para melhor compreensão foi feito também um fluxograma, na figura 4, para exemplificar os passos.

Figura 4 – Fluxograma descrevendo os passos necessários para a execução do projeto.



Fonte: O autor

Os materiais utilizados serão:

- Processador Intel Core I5 6600K 4x @3.50 GHz;
- 32 GB de memória RAM DDR4 @2166 MHz;
- Placa gráfica Nvidia RTX 2070 Gigabyte Gaming OC 8 GB GDDR6;
- SSD Samsung EVO 860 250 GB;
- Ubuntu 18.04 LTS;
- CUDA 10;
- cuDNN 7.4.2;
- OpenCV 3.4.0.

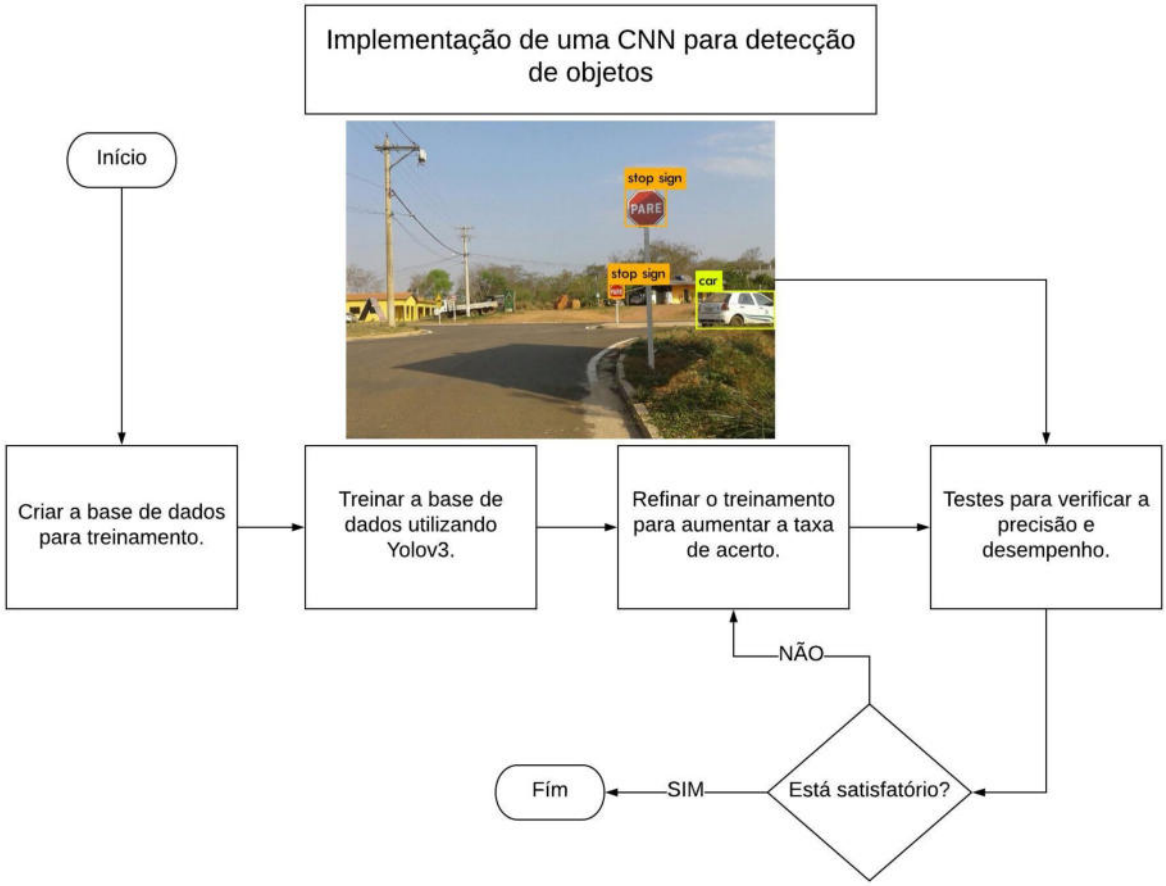
3.1 Projeto e implementação de uma CNN para detecção de objetos

A CNN escolhida para este projeto foi a *Yolov3*. Em conjunto com a *Yolov3* escolheu-se a *Darknet* como *API*, que foi desenvolvida pelo criador da Yolov3 para ser uma ferramenta rápida na detecção de objetos em tempo real. O principal diferencial do Darknet em relação ao Deep Neural Network (DNN) do OpenCV (biblioteca de DL do OpenCV), é sua compatibilidade com GPUs com Cuda Cores (NVIDIA, 2018a) e Tensor Cores (NVIDIA, 2018b) visto que o uso de GPUs geram ganhos significativos de desempenho, enquanto o DNN da OpenCV se limita ao uso de CPUs. É necessário a utilização da linguagem de programação C para fazer alterações na *API*. Algumas técnicas de *Deep Learning* serão necessárias, como, por exemplo, o *IoU* e o *mAP* para avaliação de precisão. Também foi necessário utilizar o *OpenCV* como um recurso extra à *Darknet*, para permitir que a *Darknet* possa executar vídeos.

Para utilizar a *Yolov3*, primeiro foi necessário criar uma base de dados contendo os objetos a serem detectados. foi necessário utilizar o *Yolo_mark* para a marcação de imagens para o treinamento e o *OpenCV* para que seja possível utilizar vídeos durante os testes.

Para melhor visualização da implementação foi feito o fluxograma descrito na figura 5.

Figura 5 – Fluxograma descrevendo os passos para a implementação de uma CNN para detecção de objetos.



Fonte: O autor

4 Base de dados e treinamento

4.1 Yolov3

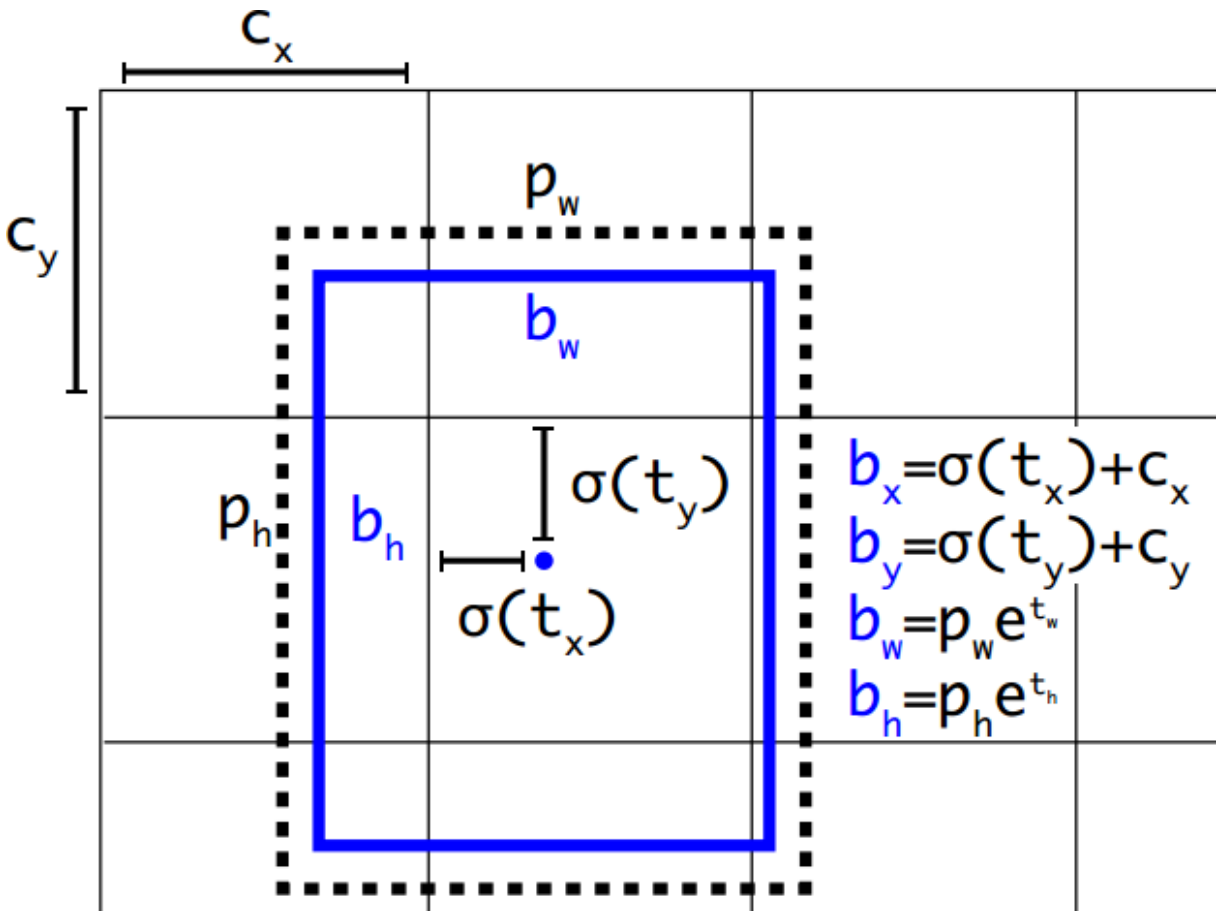
Para o desenvolvimento deste trabalho foi utilizada a *Yolov3*, uma versão mais recente da Yolo.

Segundo Redmon e Farhadi (2018) , o *Yolov3* segue os mesmos moldes de YOLO9000, onde o sistema prevê a limitação das caixas usando *clusters* de dimensão como caixas de ancoragem. A rede prevê 4 coordenadas para cada caixa delimitadora, tx, ty, tw, th. Se a célula estiver deslocada do canto superior esquerdo da imagem por (cx; cy) e a caixa delimitadora anterior tiver largura e altura pw, ph, então as previsões correspondem a:

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^t_w \\
b_h &= p_w e^t_h
\end{aligned}
$$

Durante o treinamento, é utilizada a soma do quadrado do erro da perda. Se a "*ground truth*"para alguma coordenada prevista é $\hat{t}*$, o gradiente é o valor de "*ground truth*"(calculado a partir da caixa de "*ground truth*") menos a previsão: $\hat{t} * - t*$. Este valor de "*ground truth*"pode ser calculado invertendo as equações acima. Um exemplo pode ser visto na figura 6. *Yolov3* prevê uma pontuação de objetividade para cada caixa delimitadora usando regressão logística. Deve ser 1 se a caixa delimitadora se sobrepuser a um objeto de "*ground truth*"por mais de qualquer outra caixa delimitadora anterior. Se a caixa delimitadora anterior não é a melhor, mas sobrepõe um objeto de verdade de mais do que alguns limiares, a previsão deve ser ignorada, como o *Faster R-CNN*. É utilizado o *threshold* de: 0.5. Diferente do *Faster R-CNN*, o *Yolov3* atribui apenas uma caixa delimitadora antes de cada "*ground truth*"objeto. Se uma caixa delimitadora anterior não for atribuída a um objeto "*ground truth*", não há uma queda em perda para previsões de coordenadas ou de classes, apenas objetividade.

Figura 6 – Funcionamento do sistema de caixas delimitadores de YOLOv3 de acordo com seu criador.



Fonte: (REDMON; FARHADI, 2018).

Cada caixa prevê as classes que a caixa delimitadora pode conter usando a classificação multi-rótulo. Não é usado um *softmax*, pois foi descoberto que é desnecessário para um bom desempenho, em vez disso, é usado classificadores logísticos independentes. Durante o treinamento, foi utilizado a perda de entropia cruzada binária para as previsões de classe. Esta formulação ajuda quando o *Yolov3* é utilizado em domínios mais complexo como o *Open Image Dataset*. Neste dataset existem muitos rótulos sobrepostos como, por exemplo, mulher e pessoa). Usar um *softmax* impõe a suposição de que cada caixa tem exatamente uma classe, o que geralmente não é o caso. Uma abordagem multi-rótulos modela melhor os dados.

4.1.1 Testes preliminares

Para avaliar o desempenho da detecção de objetos em imagens e vídeos do *Yolov3* e *Yolov3-tiny* foram feitos alguns testes com duas *GPU's* diferentes e uma *CPU*. Primeiramente foi testado na *Daknet* de Redmon e Farhadi (2018) que é a *Darknet* original. Logo após foi testada na *Darknet* de AlexeyAB (2017a) que é uma versão melhorada da *Darknet* e que possui mais recurso. E por fim foi testado no *DNN* do OpenCV. Em todos os casos foram utilizadas as mesmas redes com as mesmas configurações. Os resultados podem ser conferidos nas tabe-

las 1, 2, 3 e 4 . As configuração das redes podem ser encontradas online em Configurações Yolov3 e Configurações Yolov3-tiny.

Tabela 1 – Tabela demonstrando o tempo gasto por cada implementação da rede Yolov3 para detecção de objetos em imagens.

Darknet original			
Yolov3	Nvidia GT 640	Nvidia RTX 2070	Resolução da rede
FPS / Tempo (ms)	600 ms	33 ms	416x416
FPS / Tempo (ms)	Não disponível*	60 ms	608x608
FPS / Tempo (ms)	Não disponível*	88 ms	832x832
Classes	80	80	
Darknet AlexeyAB			
Yolov3	Nvidia GT 640	Nvidia RTX 2070	Resolução da rede
FPS / Tempo (ms)	Não testado	25 ms	416x416
FPS / Tempo (ms)	Não disponível*	50 ms	608x608
FPS / Tempo (ms)	Não disponível*	80 ms	832x832
Classes	80	80	
OpenCV**			
Yolov3	CPU	OPENCL	Resolução da rede
FPS / Tempo (ms)	250 ms	250 ms	416x416
FPS / Tempo (ms)	600 ms	600 ms	608x608
FPS / Tempo (ms)	1000 ms	1000 ms	832x832
Classes	80	80	

Fonte: O Autor

*Devido a falta de poder de processamento e memoria, não foi possível testar resoluções acima de 416x416.
**Por não existir ainda uma implementação cuda da biblioteca *DNN* do *OpenCV*, não foi possível utilizar a *GPU*, somente a *CPU*.

Tabela 2 – Tabela demonstrando o tempo gasto por cada implementação da rede Yolov3-tiny para detecção de objetos em imagens.

Darknet original			
Yolov3-tiny	Nvidia GT 640	Nvidia RTX 2070	Resolução da rede
FPS / Tempo (ms)	Não testado	4 ms	416x416
FPS / Tempo (ms)	Não disponível*	7 ms	608x608
FPS / Tempo (ms)	Não disponível*	12 ms	832x832
Classes	80	80	
Darknet AlexeyAB			
Yolov3-tiny	Nvidia GT 640	Nvidia RTX 2070	Resolução da rede
FPS / Tempo (ms)	Não testado	5 ms	416x416
FPS / Tempo (ms)	Não disponível*	8 ms	608x608
FPS / Tempo (ms)	Não disponível*	12 ms	832x832
Classes	80	80	
OpenCV**			
Yolov3-tiny	CPU	OPENCL	Resolução da rede
FPS / Tempo (ms)	127 ms	127 ms	416x416
FPS / Tempo (ms)	250 ms	250 ms	608x608
FPS / Tempo (ms)	600 ms	600 ms	832x832
Classes	80	80	

Fonte: O Autor

*Devido a falta de poder de processamento e memoria, não foi possível testar resoluções acima de 416x416.
**Por não existir ainda uma implementação cuda da biblioteca *DNN* do *OpenCV*, não foi possível utilizar a *GPU*, somente a *CPU*.

Tabela 3 – Tabela demonstrando o tempo gasto por cada implementação da rede Yolov3 e para detecção de objetos em vídeos.

Darknet original			
Yolov3	Nvidia GT 640	Nvidia RTX 2070	Resolução da rede
FPS / Tempo (ms)	1.8 fps / 555 ms	32 fps / 31 ms	416x416
FPS / Tempo (ms)	Não disponível*	20 fps / 50 ms	608x608
FPS / Tempo (ms)	Não disponível*	11 fps / 90 ms	832x832
Classes	80	80	
Darknet AlexeyAB			
Yolov3	Nvidia GT 640	Nvidia RTX 2070	Resolução da rede
FPS / Tempo (ms)	Não testado	45 fps 22 ms	416x416
FPS / Tempo (ms)	Não disponível*	27 fps 37 ms	608x608
FPS / Tempo (ms)	Não disponível*	16 fps 63 ms	832x832
Classes	80	80	
OpenCV**			
Yolov3	CPU	OPENCL	Resolução da rede
FPS / Tempo (ms)	4 fps / 250 ms	4 fps / 250 ms	416x416
FPS / Tempo (ms)	1.6 fps / 600 ms	1.6 fps / 600 ms	608x608
FPS / Tempo (ms)	1 fps / 1000 ms	1 fps / 1000 ms	832x832
Classes	80	80	

Fonte: O Autor

*Devido a falta de poder de processamento e memoria, não foi possível testar resoluções acima de 416x416.
**Por não existir ainda uma implementação cuda da biblioteca *DNN* do *OpenCV*, não foi possível utilizar a *GPU*, somente a *CPU*.

Tabela 4 – Tabela demonstrando o tempo gasto por cada implementação da rede Yolov3-tiny para detecção de objetos em vídeos.

Darknet original			
Yolov3-tiny	Nvidia GT 640	Nvidia RTX 2070	Resolução da rede
FPS / Tempo (ms)	Não testado	200 fps / 5 ms	416x416
FPS / Tempo (ms)	Não disponível*	120 fps / 8 ms	608x608
FPS / Tempo (ms)	Não disponível*	80 fps / 12 ms	832x832
Classes	80	80	
Darknet AlexeyAB			
Yolov3-tiny	Nvidia GT 640	Nvidia RTX 2070	Resolução da rede
FPS / Tempo (ms)	Não testado	210 fps / 4 ms	416x416
FPS / Tempo (ms)	Não disponível*	130 fps / 7 ms	608x608
FPS / Tempo (ms)	Não disponível*	85 fps / 11 ms	832x832
Classes	80	80	
OpenCV**			
Yolov3-tiny	CPU	OPENCL	Resolução da rede
FPS / Tempo (ms)	30 fps / 33 ms	30 fps / 33 ms	416x416
FPS / Tempo (ms)	5 fps / 200 ms	5 fps / 200 ms	608x608
FPS / Tempo (ms)	1.6 fps / 600 ms	1.6 fps / 600 ms	832x832
Classes	80	80	

Fonte: O Autor

*Devido a falta de poder de processamento e memoria, não foi possível testar resoluções acima de 416x416.
**Por não existir ainda uma implementação cuda da biblioteca *DNN* do *OpenCV*, não foi possível utilizar a *GPU*, somente a *CPU*.

Em todos os testes a resolução máxima foi 832x832, pois a rede não foi treinada em resoluções superiores. Foi observado durante os testes que o *DNN* do *OpenCV* possui um desempenho inferior devido à utilização apenas da *CPU* e também possui uma precisão inferior detectando menos objetos durante os vídeos. Por este fato, foi decidido utilizar apenas a *Darknet*, que por utilizar a *GPU* e o *CUDA*, obtêm um desempenho e uma precisão superior ao *DNN* do *OpenCV*. Em ambos os testes, para o *Darknet* foi utilizado o *OpenCV 3.4.0*, e para o *DNN* o *OpenCV 3.4.3*.

4.2 Base de dados

Neste trabalho os objetos a serem identificados encontram-se nas tabelas 5 e 6. As ilustrações dos objetos encontram-se no apêndice A.1.

Tabela 5 – Tabela mostrando os objetos que foram detectados pela rede distribuídos em 47 classes.

Objeto	Classe
Cone	0
Pedestre	1
Bicicleta	2
Ciclista	3
Veiculo	4
Moto	5
Motociclista	6
Pare	7
Semáforo	8
Proibido_virar_direita	9
Proibido_virar_esquerda	10
Virar_direita	11
Virar_esquerda	12
De_preferencia	13
Sentido_proibido	14
Proibido_retornar	15
Estacionamento_regulamentado	16
Proibido_estacionar	17
Proibido_parar_e_estacionar	18
Área_escolar	19
Saliência_ou_lombada	20
Estreitamento_de_pista_ao_centro	21
Estreitamento_de_pista_a_esquerda	22

Fonte: O Autor

Tabela 6 – Continuação da tabela 5.

Estreitamento_de_pista_a_direita	23
Ponte_estreita	24
Curva_acentuada_a_esquerda	25
Curva_acentuada_a_direita	26
Curva_a_esquerda	27
Curva_a_direita	28
Pista_sinuosa_a_esquerda	29
Pista_sinuosa_a_direita	30
Curva_acentuada_em_S_a_esquerda	31
Curva_acentuada_em_S_a_direita	32
Curva_em_S_a_Esquerda	33
Curva_em_S_a_direita	34
Pista_irregular	35
Depressão	36
Declive_acentuado	37
Aclive_acentuado	38
Pronto_socorro	39
Abastecimento	40
Restaurante	41
Aeroporto	42
Hotel	43
Ponto_de_parada	44
Identificação	45
Orientação_de_destino	46

Fonte: O Autor

Tendo escolhido os objetos a serem detectados, o próximo passo foi criar uma base de dados com diversas imagens desses objetos, tendo em mente que a dimensão ideal das imagens é um valor superior a dimensão utilizada na rede. Para a criação foram utilizados os seguintes métodos:

- Inicialmente utilizou-se o *Google Imagens* para procurar imagens relativas aos objetos selecionados.
- Apesar de alguns itens como veículos e pedestres serem encontrados mais facilmente, itens como placas de trânsito, por exemplo, por não serem comuns são mais difíceis de encontrar com maior diversidade.
- A primeira estratégia encontrada para solucionar o problema foi utilizar o *Google Street-view* para adquirir as imagens das placas fotografadas pelas rotas percorrida pelo carro da *Google*.
- Uma segunda estratégia foi utilizar o *software Blender 3D* para renderizar as placas para aumentar a diversidade da base de dados.

- Uma outra alternativa foi utilizar bases mais difundidas pela comunidade como, por exemplo, Krause et al. (2013), que possui diversas imagens de carros.
- A mesma estrategia foi adotada com semáforos e pedestres que foram retiradas de Worcester Polytechnic (2018).
- Com aproximadamente 10000 imagens na base de dados, foram utilizados filtros de programas de edição de imagens, no caso GIMP 2.10, para aumentar o quantidade de variações de iluminação, brilho, contraste e coloração das imagens da base de dados.

4.3 Treinamento

4.3.1 Arquivos

Uma vez que a base está pronta é necessário criar dois arquivos[1]:

- *"obj.names"*;
- *"obj.data"*.

Um exemplo dos arquivos pode ser visto nas figuras 7 e 8. O arquivo *"obj.data"* é um arquivo que possui respectivamente o numero de classes, o diretório das imagens com os rótulos para treino, diretório para validação e o diretório do arquivo de backup para continuar o treinamento caso deseje pausar. O arquivo *"obj.names"* contém o nome de cada um dos objetos a serem reconhecidos e são separados por linha.

Figura 7 – Figura ilustrando o arquivo obj.data.

```
classes= 47
train = /home/diego/treino_placas/train_oficial.txt
valid = /home/diego/treino_placas/train_oficial.txt
names = /home/diego/darknet_treino/cfg/treino/oficial.names
backup = /home/diego/darknet_treino/backup/oficial
```

Fonte: O autor.

[1] O nome dos arquivos podem ser diferentes, de acordo com a vontade do usuário, sendo necessário manter apenas a extensão.

Figura 8 – Figura ilustrando o arquivo obj.names.

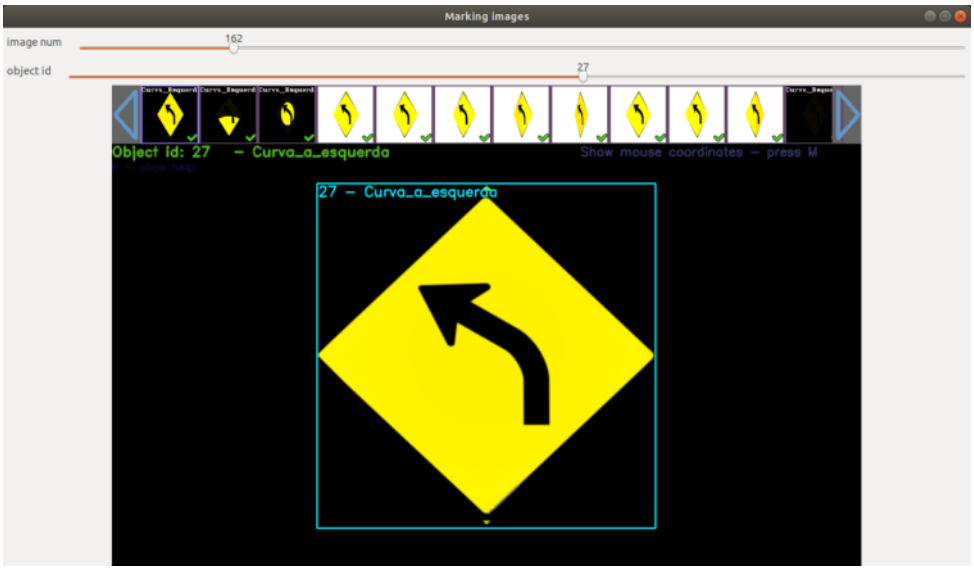
```
Cone
Pedestre
Bicicleta
Ciclista
Veiculo
Moto
Motociclista
Pare
Semaforo
Proibido_virar_direita
Proibido_virar_esquerda
Virar_direita
Virar_esquerda
De_preferencia
Sentido_proibido
Proibido_retornar
Estacionamento_regulamentado
Proibido_estacionar
Proibido_parar_e_estacionar
Area_escolar
Saliencia_ou_lombada
Estreitamento_de_pista_ao_centro
Estreitamento_de_pista_a_esquerda
Estreitamento_de_pista_a_direita
Ponte_estreita
Curva_acentuada_a_esquerda
Curva_acentuada_a_direita
```

Fonte: O autor.

4.3.2 Rotulação Manual

Caso a base de dados não possua uma rotulação, é preciso rotular manualmente as imagens para que a rede possa identificar onde encontra-se cada objeto em cada uma das imagens, e é necessário utilizar o *Yolo_mark* desenvolvido por AlexeyAB (2017b). O *"Yolo_mark"* é uma ferramenta criada para rotular manualmente cada objeto e transformar este rotulo em coordenadas que são lidas pela rede para identificação do objeto. É preciso informar o diretório das imagens, o arquivo onde serão salvos os diretórios das imagens e dos arquivos com as coordenadas e *"obj.names"* para que possa-se rotular as imagens com as classes corretas, conforme as figuras 9 e 10.

Figura 9 – Figura ilustrando uma marcação manual.



Fonte: O autor.

Figura 10 – Figura ilustrando as coordenadas contidas em um arquivo com a marcação manual, referente a figura 9.

```
27 0.499609 0.502083 0.452344 0.818056
```

Fonte: O autor.

4.3.3 Extras

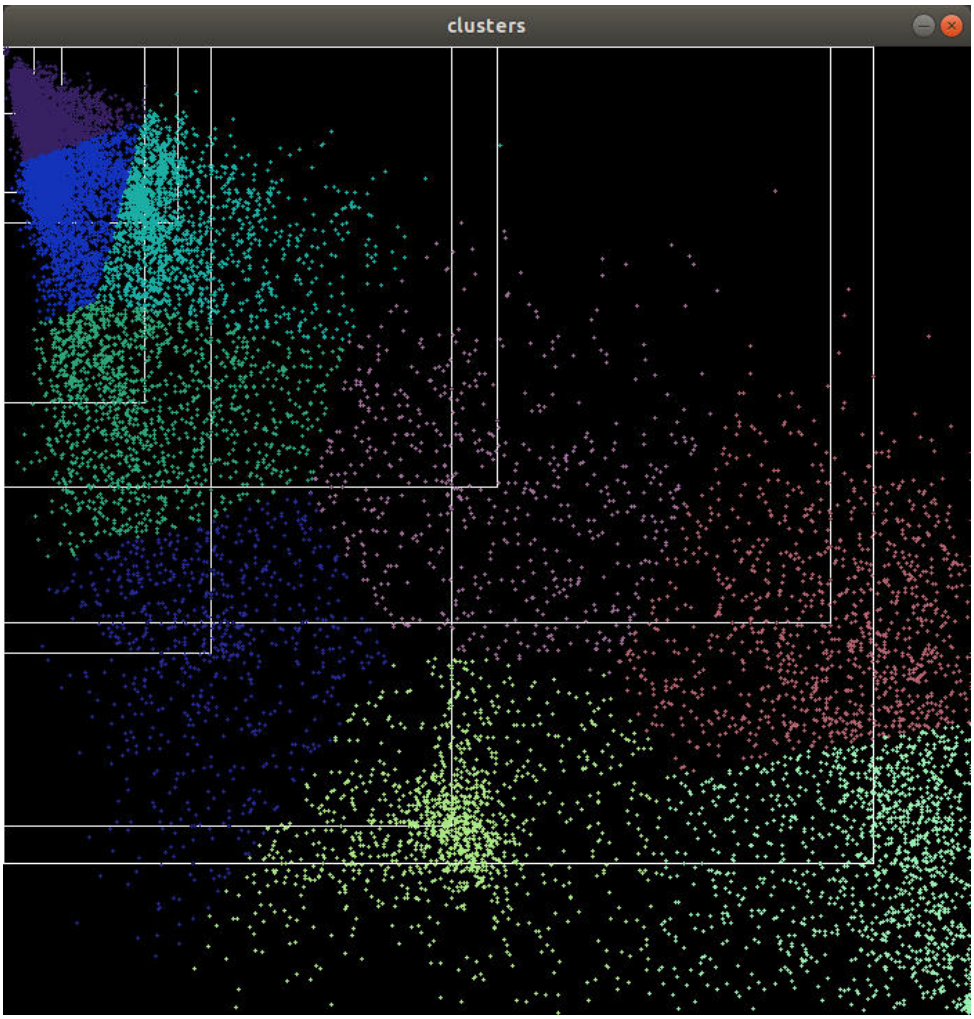
É recomendável, mas não necessário, utilizar a função *"calc_anchors"* de AlexeyAB (2017a) para recalcular as ancoras, para que possa-se aumentar a precisão do treinamento, e esse calculo é feito com base nas imagens e nos posicionamentos dos objetos nas imagens, podendo variar de acordo com o numero de imagens disponíveis na base de dados.

Figura 11 – Figura ilustrando a posição das ancoras.

```
num_of_clusters = 9, width = 608, height = 608
read labels from 79627 images
loaded          image: 79627    box: 158281
all loaded.

calculating k-means++ ...

avg IoU = 64.02 %

Saving anchors to the file: anchors.txt
anchors = 19.6289,42.4559, 37.2444,91.7812, 110.1220,110.3282, 88.8667,223.9557, 131.0294,380.4851, 310.2348,276.2910, 281.8358,489.0187, 519.9384,361
.4251, 546.6531,513.3019
```

Fonte: O autor.

Figura 12 – Figura ilustrando a região das ancoras, referente a figura 11



Fonte: O autor.

4.3.4 Pesos

É necessário um arquivo de pesos convolucionais para que o treinamento possa ser iniciado. Esse arquivo pode ser o *"darknet53.conv.74"*, que pode ser adquirido no site de seu criador <https://pjreddie.com/media/files/darknet53.conv.74>, caso a rede utilizada seja a *Yolov3*, ou pode ser adquirido através do comando de AlexeyAB (2017a): *"./darknet partial cfg/yolov3.cfg yolov3.weights yolov3.conv.81 81"* que extrai o peso convolucional da 81ª convolução da rede treinada *Yolov3*, na qual foi utilizada a base de dados COCO (2018) do desafio do ano de 2014.

É possivel utilizar o mesmo comando para extrair um treinamento parcial do *Yolov3-tiny* utilizando o arquivo comando *"./darknet partial cfg/yolov3-tiny.cfg yolov3-tiny.weights yolov3-tiny.conv.15 15"*. Utilizar a extração do peso permite um ajuste fino, ao contrário do *darknet53.conv.74*, que permite transferir o conhecimento. Neste trabalho foi utilizado o ajuste fino ao invés da transferência de conhecimento devido ao desempenho superior na tarefa de detecção de objetos.

4.3.5 Configuração da rede

- Primeiramente é necessário ter o arquivo yolov3.cfg, que pode ser encontrado em <https://github.com/pjreddie/darknet/blob/master/cfg/yolov3.cfg>.
- Depois é preciso encontrar as camadas *[yolo]* no arquivo de configurações, pois é nela que define-se o numero de classes.
- Em seguida é necessário encontrar a camada *[convolutional]* imediatamente acima de cada camada *[yolo]* e alterar o valor de *filters* utilizando a seguinte equação:

$$(numerodeclasses + 5) * 3 \tag{4.1}$$

- Caso as ancoras tenham sido recalculadas é necessário alterar o valor das mesmas na camada *[yolo] anchors*.
- Caso seja necessário acelerar o treinamento, pode utilizar adicionar a linha *stopbackward=1* antes da marcação ********** presente no arquivo. Essa linha impede que o conhecimento seja *retropropagado* aumentando a velocidade do treinamento, mas diminuindo a precisão.
- Dependendo da quantidade de memória *RAM* disponível na placa gráfica, é necessário alterar as linhas *batch* e *subdivisions*, no inicio do arquivo, para valores maiores, mas esse aumento diminui a velocidade de treinamento.
- Caso seja necessário uma precisão maior é recomendável alterar as linhas *width* e *height* para valores maiores, que sejam múltiplos de 32, mas aumentar os valores diminuem a velocidade de treinamento e consomem mais memória *RAM* da GPU.
- Caso seja necessário a diferenciação entre direita e esquerda e afins, é necessário adicionar a linha *flip=0* logo abaixo de *hue=.3*.
- A linha *max_batches* define o número de iterações totais do treinamento. Por padrão o valor é *500200*, podendo ser alterado de acordo com a necessidade.
- As linhas *width* e *height* definem a resolução de treinamento da rede, o que significa que, quanto maior a resolução mais precisa é a detecção de objetos e quanto menor a resolução menos precisa é a detecção. Deve-se levar em conta que o aumento de resoluções aumenta também o consumo de memória da placa gráfica e a carga de trabalho, diminuindo o desempenho. Neste trabalho foi escolhida a resolução 608x608, pois é o limite máximo que o hardware utilizado consegue treinar a rede, sem que haja estouro de memória por parte da placa gráfica durante o treinamento.

4.3.6 Execução

O próximo passo é treinar a rede e o recomendado é 2000 iterações por classe, de acordo com (ALEXEYAB, 2017a), nesse caso seriam necessárias pelo menos 94000 iterações. Inicialmente havia sido escolhida a rede *Yolov3-tiny* para este trabalho por desempenho de

processamento, mas devido a questões relativas a precisão, foi escolhida a rede *Yolov3*. Para avaliar o desempenho do treinamento do *Yolov3-tiny* e *Yolov3* foram feitos alguns testes com duas *GPU's* diferentes. Os resultados podem ser conferidos nas tabelas 7 e 8. Neste teste, a limitação de memória da GPU Nvidia GT 640 impediu que o treinamento fosse realizado no *Yolov3*.

Tabela 7 – Tabela demonstrando o tempo gasto por iteração e quantidade de classes durante o treinamento com 2 classes para a rede *Yolov3-tiny*.

GPU	Tempo por iteração	Tempo por iteração	Iterações por minuto	Rede
Nvidia GT 640	9h / 3900	12h / 5100	6,85 its/min	Yolov3-tiny
Nvidia RTX 2070	43 min / 3900	56 min / 5100	90 its/min	Yolov3-tiny
Classes	2	2		
Qtd de imagens	220	220		

Fonte: O Autor

Tabela 8 – Tabela demonstrando o tempo gasto por iteração e quantidade de classes durante o treinamento com 2 classes para a rede *Yolov3*.

GPU	Tempo por iteração	Tempo por iteração	Iterações por minuto	Rede
Nvidia GT 640	Não disponível*	Não disponível*	Não disponível*	Yolov3
Nvidia RTX 2070	2,3 h / 3900	3,07 h / 5100	27,6 its/min	Yolov3
Classes	2	2		
Qtd de imagens	220	220		

Fonte: O Autor

*Devido a falta de poder de processamento e memoria, não foi possível testar resoluções utilizando a GPU Nvidia GT 640.

Tais diferenças nos tempos e nos processamentos ocorrem devido à arquiteturas diferentes disponíveis em cada placa gráfica, principalmente pelo ano de lançamento de cada uma, sendo a Nvidia GT 640 de 2012, e a Nvidia RTX 2070 de 2018.

Uma vez executadas as 94000 iterações necessárias para o treinamento, através de testes, foi constatado que ainda não era o suficiente para obter-se uma boa precisão e era necessário um tempo maior de treinamento. Foi decidido deixar o treinamento ocorrer por 500200 iterações, conforme a tabela 9.

Tabela 9 – Tabela demonstrando o tempo gasto por iteração e quantidade de classes durante o treinamento com 47 classes para rede *Yolov3*.

GPU	Iterações por minuto	Tempo total	Rede
Nvidia RTX 2070	27,6 its/min	302,05 h - 15 dias	Yolov3
Classes	47		
Qtd de imagens	79627		
Iterações	500200		

Fonte: O Autor

4.3.7 Avaliação inicial

Para verificar a taxa de acerto do treinamento da rede é necessário executar o comando *"map"* de (ALEXEYAB, 2017a), que faz os cálculos com base nas imagens e nos rótulos para determinar quantos desses estão corretos e quantos desses a rede consegue detectar por conta própria. O comando *"map"* utiliza como uma das métricas o *mAP*, que é o *mean averege precision* que é o valor médio da "precisão média"de cada classe, onde a "precisão média"uma média de valores de 11 pontos de uma curva *"Precision-Recall"* para cada possibilidade de detecção para uma mesma classe, onde:

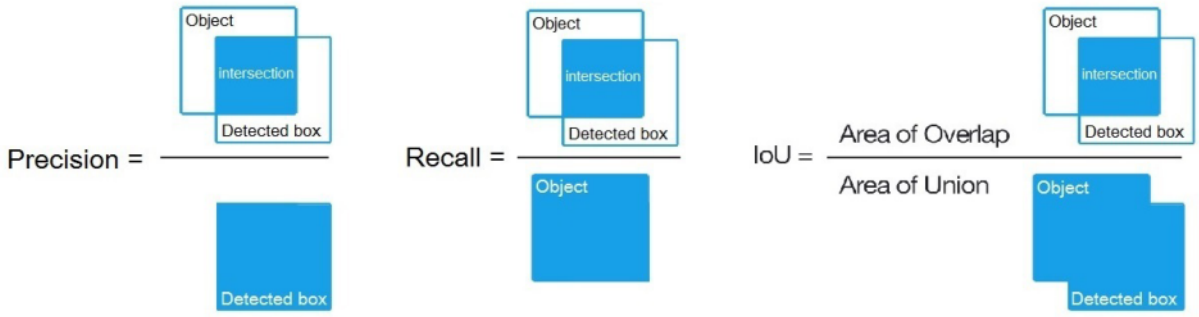
$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

sendo TP *True positive*, FP *False positive* e FN *False negative*.

O *"map"* também traz como outra métrica o *IoU*, que é a interseção sobre a união (*Intersection over union*), que representa a média de interseções sobre uniões de objetos detectados para um determinado *threshold*, no caso da *Yolov3*, 0.24 (ALEXEYAB, 2017a). A figura 13 mostra um pouco o funcionamento do *mAP* e do *IoU*.

Figura 13 – Figura ilustrando o funcionamento do mAP e do IoU.



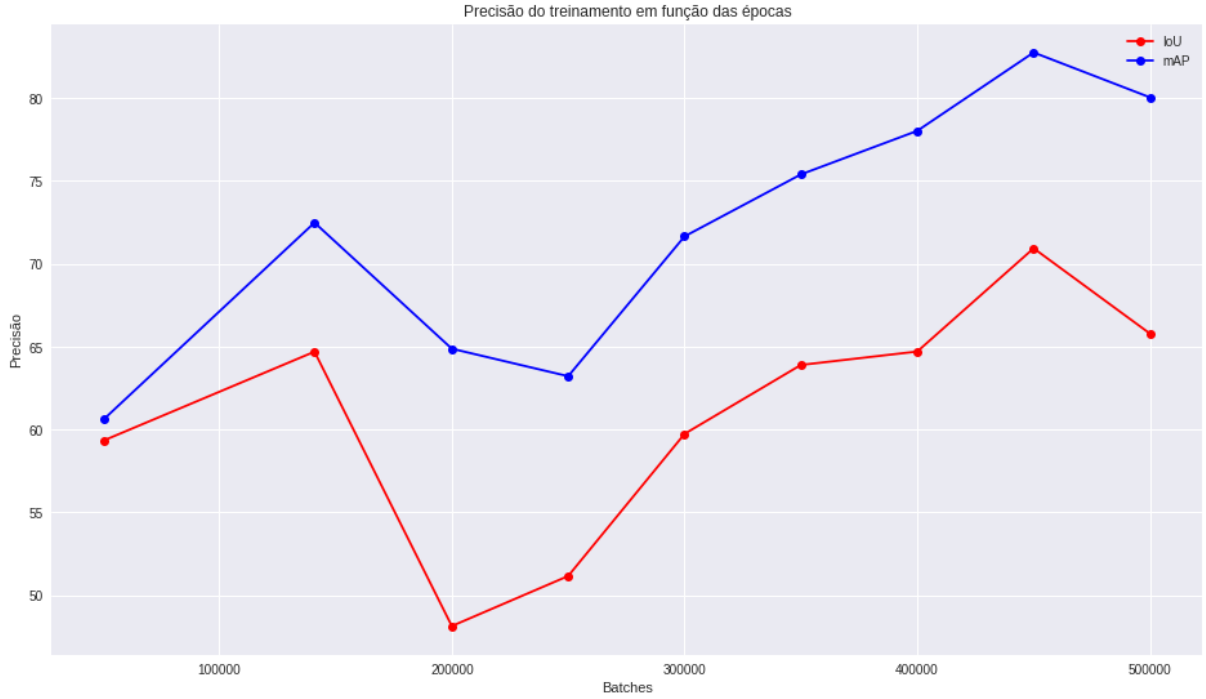
Fonte: (ALEXEYAB, 2017a).

A figura 13 mostra que a precisão são todas as detecções corretas divididas por todas as detecções feitas, certas ou erradas. O *recall* são todas as detecções corretas divididas por

todas as marcações manuais. O *IoU* é a área de interseção entre a marcação manual e a detecção feita dividida pela área de união entre as duas.

A figura 14 demonstra o *mAP* e o *IoU* durante o treinamento. O gráfico indica quando seria o momento ideal para parar o treinamento e quando é necessário uma quantidade maior de dados para que ela consiga continuar evoluindo. Com base apenas no gráfico, o melhor momento para parar o treinamento seria na iteração 450000, pois foi onde obteve-se o melhor *mAP* e *IoU*. E é também por causa da queda após este ponto que percebeu-se a necessidade de mais dados para continuar o treinamento, uma vez que, através de testes, constatou-se que a quantidade de objetos detectados estava baixa.

Figura 14 – Figura ilustrando o mAP e o IoU conforme o treinamento da rede.



Fonte: O autor.

4.3.8 Avaliação secundária

Após o termino do treinamento os resultados não foram como os desejados, por isso decidiu-se continuar um pouco mais o treinamento, até 901800 iterações. Junto com a continuação decidiu-se adicionar também novas imagens para tentar aumentar ainda mais o desempenho durante a execução. Foi adicionada a base de dados COCO (2018) do desafio de 2017, mas utilizando apenas as imagens relativas a base de dados original, ou seja, as imagens que possuem marcações são as seguintes:

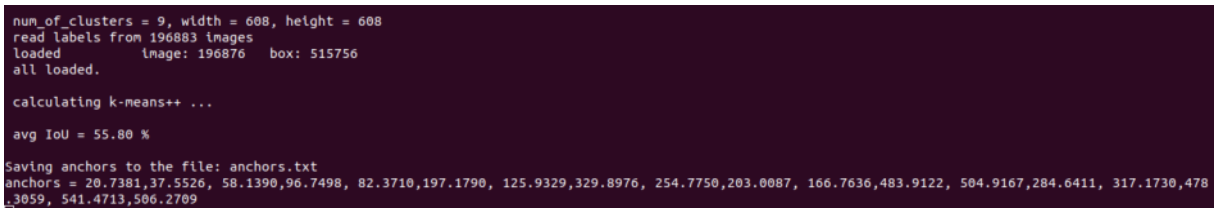
- pessoas;
- bicicletas;
- veículos;

- motocicletas;
- semáforos;
- placas de pare;

As outras imagens foram utilizadas como amostras negativas, e de acordo com AlexeyAB (2017a), essas amostras negativas são úteis para demonstrar para a rede objetos que ela não deve detectar.

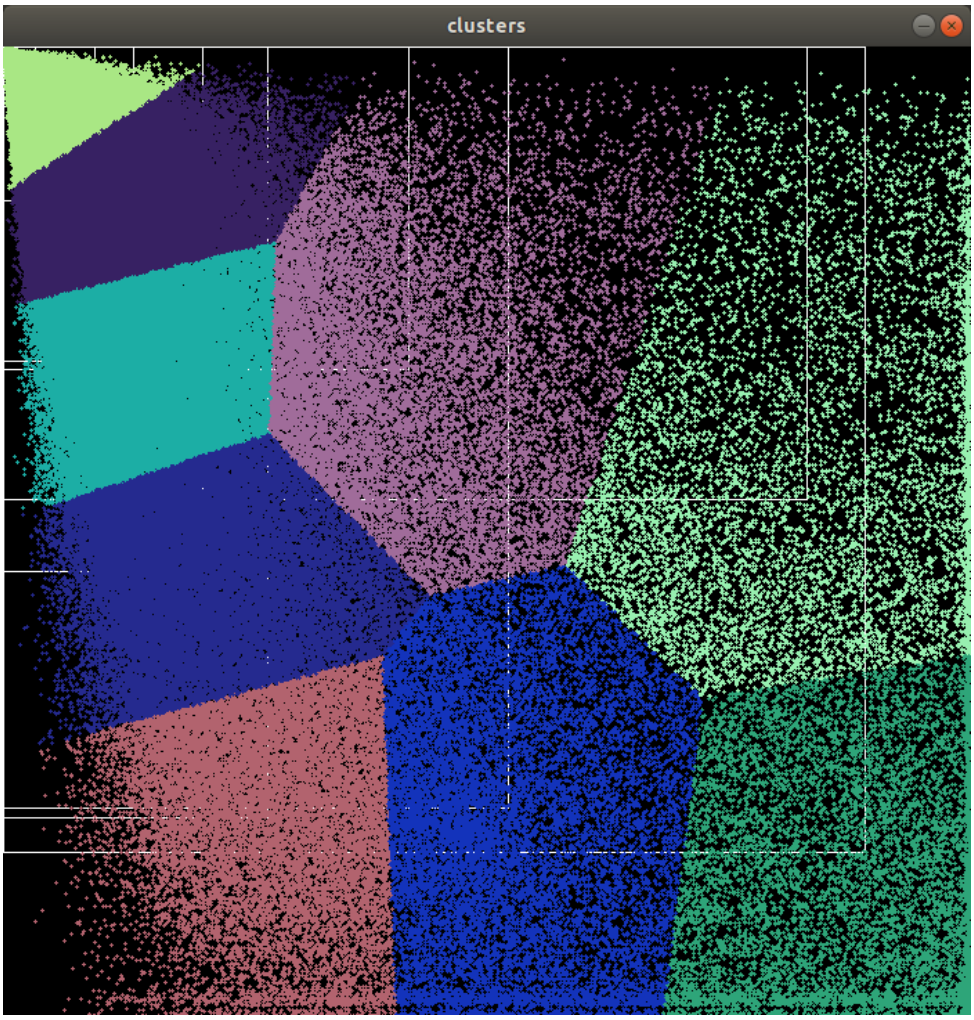
Junto das imagens, as ancoras também foram recalculadas, e estão dispostas nas figuras 15 e 16.

Figura 15 – Figura ilustrando a posição das ancoras.



Fonte: O autor.

Figura 16 – Figura ilustrando a região das ancoras, referente a figura 15



Fonte: O autor.

É esperado que com essas novas adições o *mAP* e *IoU* tenham uma queda, mas durante as detecção, mais objetos consigam ser detectados e corretamente, diminuindo o número de falsos positivos.

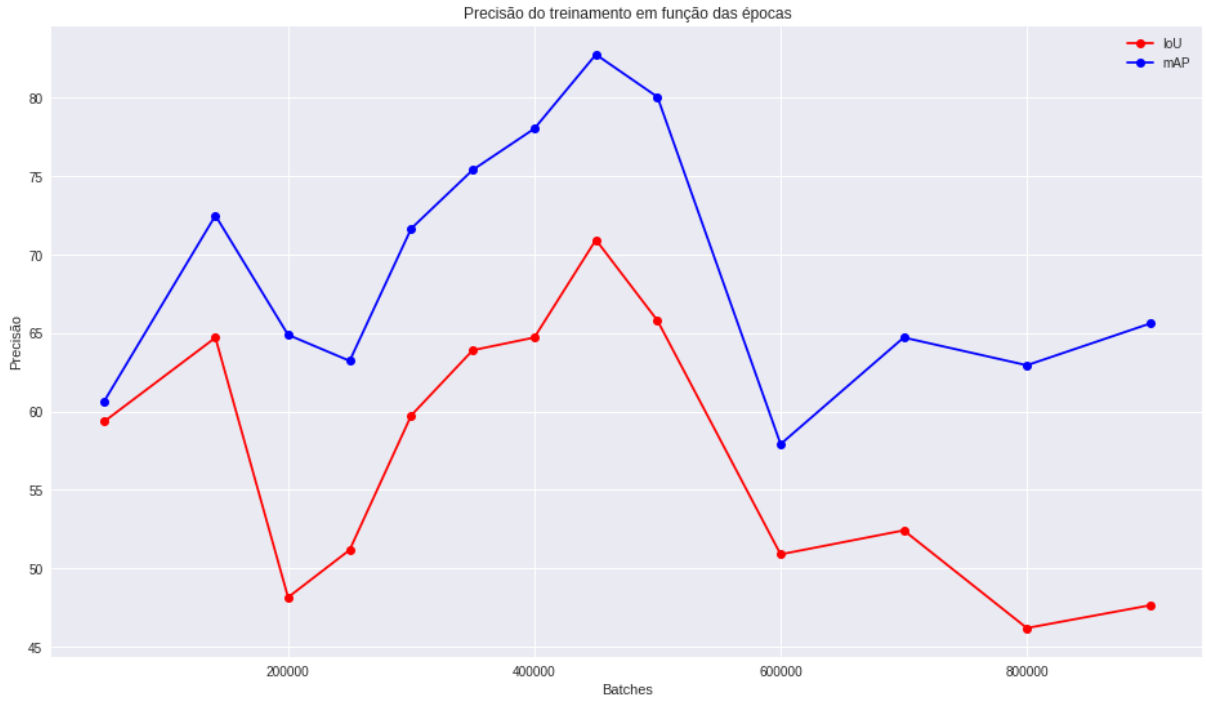
Com essas adições e modificações, o tempo total e a quantidade de imagens aumentou e estão dispostas na tabela 10. O *mAP* e o *IoU* estão na figura 17.

Tabela 10 – Tabela demonstrando o tempo gasto por iteração e quantidade de classes durante o treinamento com 47 classes para rede *Yolov3* com a adição de novas imagens.

GPU	Iterações por minuto	Tempo total	Rede
Nvidia RTX 2070	27,6 its/min	543,47 h - 22,6 dias	Yolov3
Classes	47		
Qtd de imagens	196876		
Iterações	901800		

Fonte: O Autor

Figura 17 – Figura ilustrando o mAP e o IoU conforme o treinamento da rede com a adição de novas imagens.



Fonte: O autor.

Todos os arquivos gerados por este trabalho podem ser encontrados no link: <https://drive.google.com/open?id=1g8xhIGA2TflhiG5NTiAYhv9QPnAP9yTe>.

5 Testes e resultados

5.1 Testes

Shah et al. (2017) desenvolveram o AirSim, uma plataforma cujo o objetivo é a pesquisa e o treinamento de Deep Learning, visão computacional e algoritmos de aprendizado reforçado para veículos autônomos, utilizando a Unreal Engine 4 (UE4), para tal treinamento. Segundo Epic (2018), a UE4 permite o treinamento de inteligência artificial de veículos autônomos com seus ambientes fotorrealistas. Seguindo a mesma linha de pensamento, para a realização dos testes foi escolhido utilizar a *Unreal Engine 4* (UE4), para testar o limite teórico da rede *Yolov3*. Então foi criado um ambiente virtual composto dos objetos das tabelas 5 e 6. Dado o fotorrealismo que é possível atingir dentro deste ambiente virtual, ele torna-se a melhor ferramenta para avaliar a capacidade da rede no quesito de detecção de objetos, uma vez que não há disponível um protótipo de veículo autônomo para ser utilizado em conjunto com a rede. Para construir o ambiente foram utilizados diversos *assets* retirados do site Turbosquid, com exceção das placas, que foram criadas como parte da base de treinamento e importadas para a UE4. No ambiente, a rede foi testada em diversas situações:

- situações simples com poucos objetos;
- situações estressantes com diversos objetos pertencentes a diferentes classes.

foi testada em diferentes cenários:

- manhã;
- tarde;
- noite;
- pouca iluminação;
- muita iluminação.

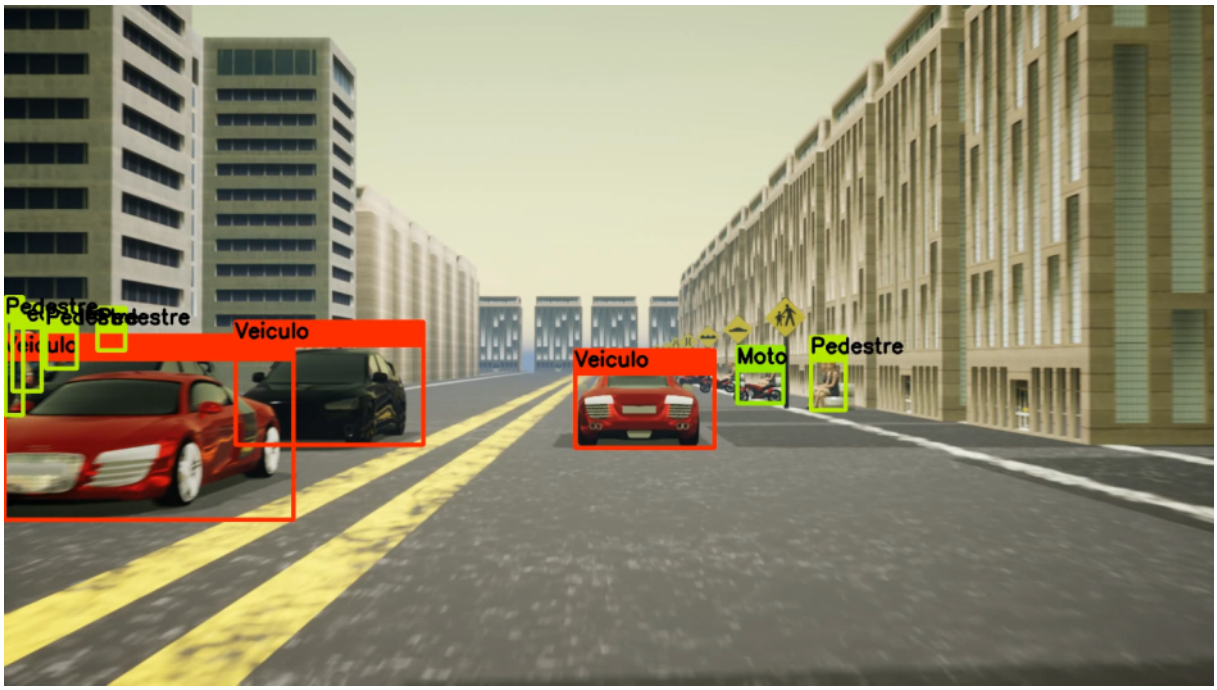
Todos estes testes servem para poder determinar não só a precisão, mas também os casos em que a detecção é falha e como isso pode impactar a base de dados para treinamentos futuros.

Como testes, foram feitos vídeos curtos utilizando funções da própria UE4 chamadas de *cinematics*. Foi avaliado em cada vídeo quantos objetos foram detectados por quadro e quantos desses objetos estão corretos. Também foram utilizados exemplos de cenários reais, através de imagens retiradas do *Google Images*.

5.1.1 Testes do ambiente virtual

Um exemplo de cada cenário pode ser observado a partir da figura 18 até 23.

Figura 18 – Detecção de objetos no ambiente virtual do cenário 1.



Fonte: O Autor.

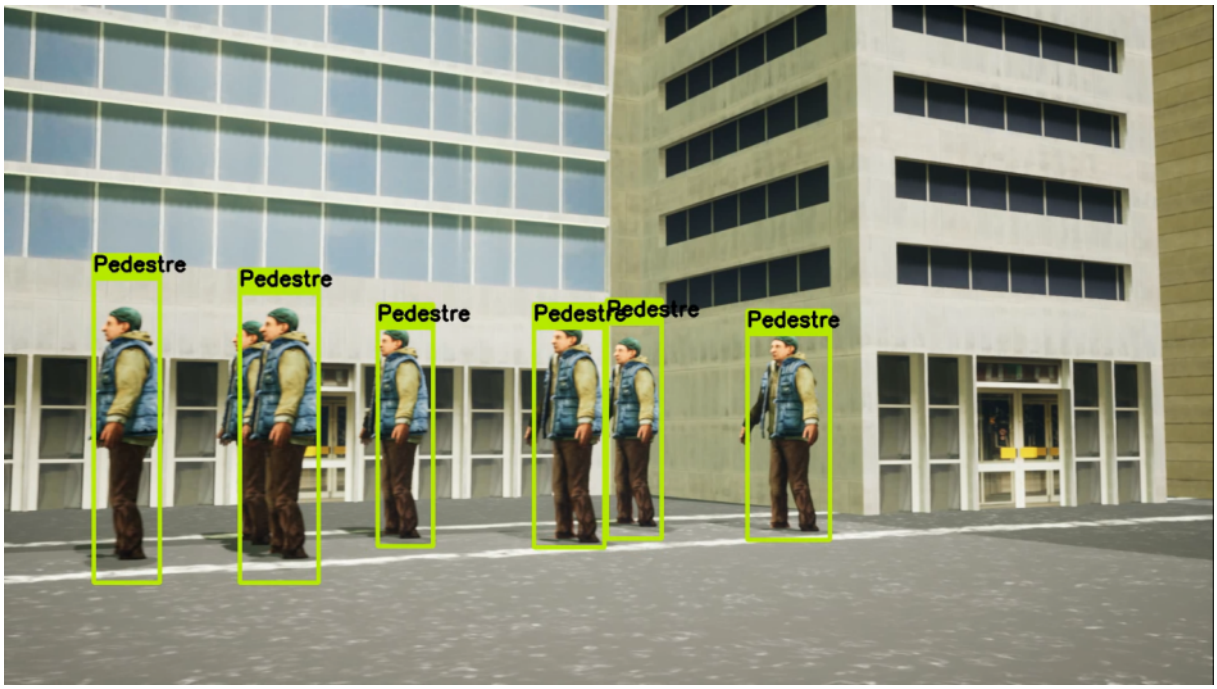
Figura 19 – Detecção de objetos no ambiente virtual do cenário 2.

Fonte: O Autor.

Figura 20 – Detecção de objetos no ambiente virtual do cenário 3.

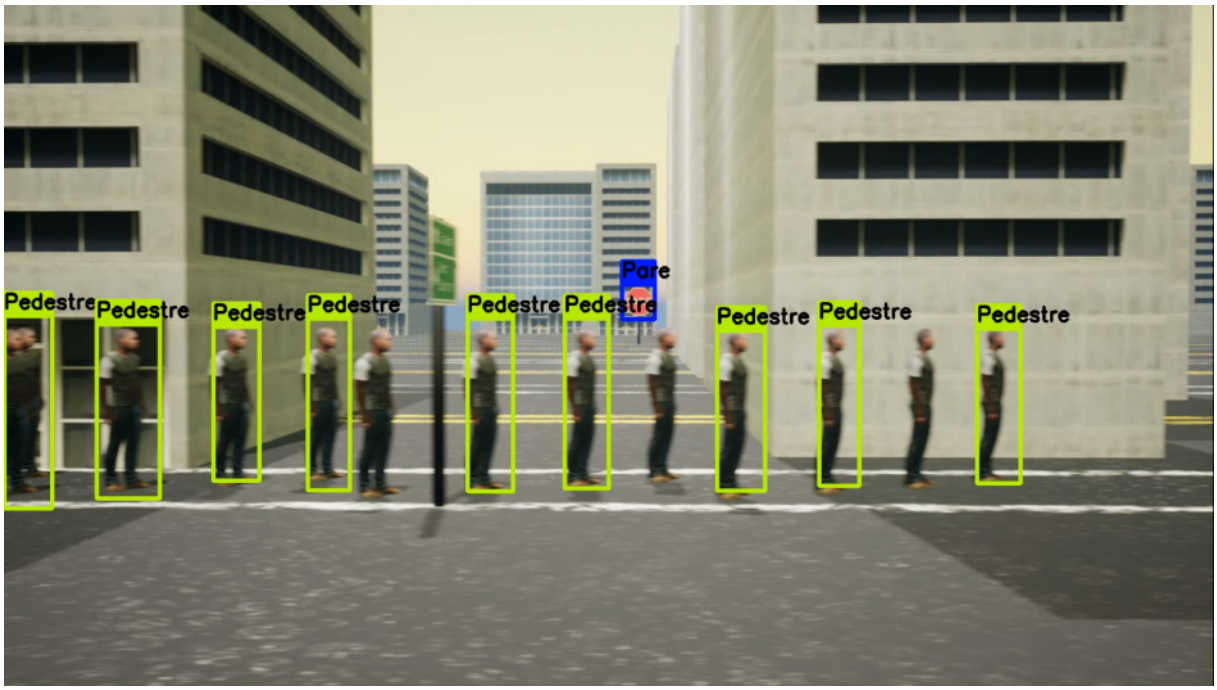
Fonte: O Autor.

Figura 21 – Detecção de objetos no ambiente virtual do cenário 4.



Fonte: O Autor.

Figura 22 – Detecção de objetos no ambiente virtual do cenário 5.



Fonte: O Autor.

Figura 23 – Detecção de objetos no ambiente virtual do cenário 6.

Fonte: O Autor.

Na tabelas 11, 12, 14 e 13 encontram-se os testes feitos no ambiente virtual.

Na construção da tabela 11, foi levado em conta que a *API Darknet* faz novas detecções a cada quadro. Por essa razão foi considerada a quantidade de objetos possíveis à serem

detectados em cada quadro na contagem total de objetos do cenário, logo, se havia 5 pessoas em quadro e haviam 6 no próximo o total dos dois quadros seria 11, por isso há uma grande quantidade de objetos em cada cenário.

Tabela 11 – Tabela demonstrando a quantidade de objetos presentes em cada cenário.

Cenários	Quantidade total de objetos
01	3247
02	302
03	900
04	969
05	2427
06	290

Fonte: O Autor

Na tabela 12 é possível identificar a quantidade de objetos detectados corretamente, ou seja, o objeto classificado com a classe correta. Desse montante estão excluídos os falsos negativos (quando um objeto pertencente a uma classe presente na base de dados não é identificado como tal) e falsos positivos (quando objetos pertencentes a uma classe presente na base de dados é identificado equivocadamente como outro objeto). Na mesma tabela estão apresentados os valores de detecções máximas em cada cenário, permitindo identificar a taxa de sucesso.

Tabela 12 – Tabela demonstrando a quantidade de objetos encontrados pela rede treinada, e quantos desses estão corretos, no ambiente virtual.

Cenários	Manhã	Tarde	Noite	Muita Iluminação	Pouca Iluminação
01	737/737	520/520	328/332	1458/1459	186/188
02	37/70	64/65	25/38	72/81	11/11
03	72/73	69/71	44/45	119/122	56/56
04	564/570	620/631	106/106	615/617	471/472
05	1186/1186	1160/1160	679/679	1010/1010	791/791
06	0/0	0/0	0/1	1/1	0/0

Fonte: O Autor

Lê-se detecções corretas e total de detecções.

Durante todos os vídeos foi possível atingir uma média de aproximadamente 30 *FPS*, com uma resolução interna da rede de 608x608.

NA tabela 13 é apresentada a taxa de objetos localizados em cada cenário em relação aos dados da tabela 11, em que há o total possível de objetos detectáveis.

Tabela 13 – Tabela demonstrando a precisão relativa à quantidade de objetos detectados pela rede durante os testes no ambiente virtual.

Cenários	Manhã	Tarde	Noite	Muita Iluminação	Pouca Iluminação
01	22,6%	16%	10%	44%	5%
02	12%	21%	8%	23%	3,6%
03	8%	7,6%	4,8%	13%	6%
04	58%	63,9%	10,9%	63%	48,6%
05	48,8%	47,7%	27,9%	41,6%	32,5%
06	0%	0%	0%	0%	0%

Fonte: O Autor

A tabela 14 mostra o percentual de acerto na classificação correta dos objetos detectados pela rede, em relação somente aos que ela foi capaz de detectar.

Tabela 14 – Tabela demonstrando a precisão relativa classificação de objetos detectados pela rede durante os testes no ambiente virtual.

Cenários	Manhã	Tarde	Noite	Muita Iluminação	Pouca Iluminação
01	100%	100%	100%	99,9%	98,9%
02	52,8%	98%	65,7%	88,8%	100%
03	98,6%	97%	97,7%	97,5%	100%
04	98,9%	98%	100%	99,6%	99,7%
05	100%	100%	100%	100%	100%
06	0%	0%	0%	100%	0%

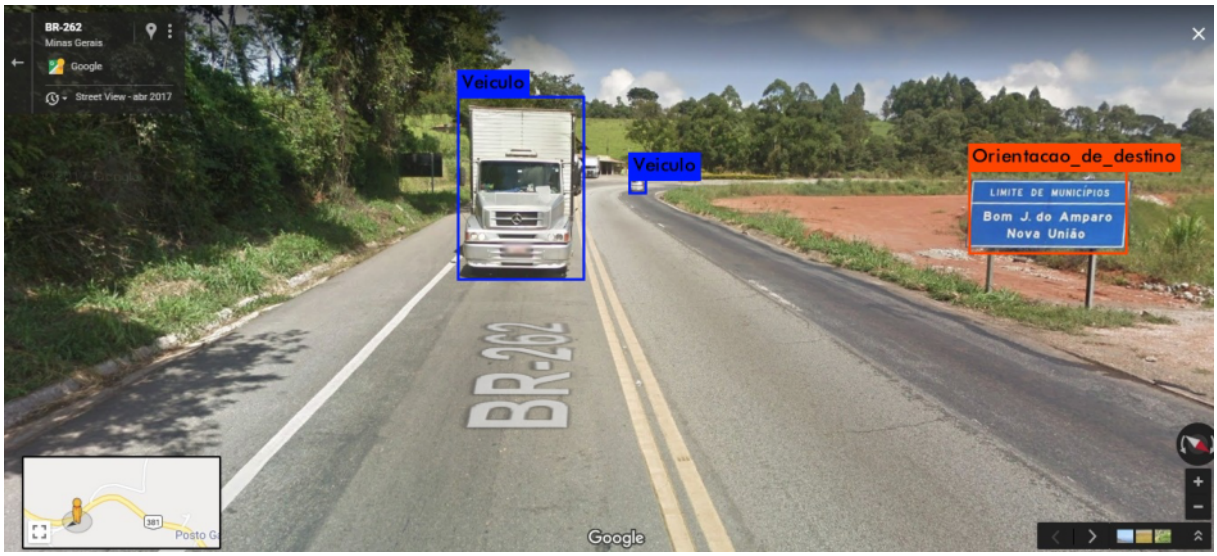
Fonte: O Autor

Os resultados e os vídeos estão disponíveis no link: Testes

5.1.2 Testes de cenários reais

A partir das figuras 24 a 29 é possível perceber como se comportam as detecções de cenários reais.

Figura 24 – Detecção de objetos de cenários reais com imagens do *Google Images*.



Fonte: O Autor.

Na figura 24 foram localizados 3 objetos de 4 possíveis, mostrando uma precisão de 75% em relação a quantidade de objetos e 100% em relação aos objetos detectados.

Figura 25 – Detecção de objetos de cenários reais com imagens do *Google Images*.

Fonte: O Autor.

Na figura 25 foram localizados 3 objetos de 3 possíveis, mostrando uma precisão de 100% em relação a quantidade de objetos e 100% em relação aos objetos detectados.

Figura 26 – Detecção de objetos de cenários reais com imagens do *Google Images*.

Fonte: O Autor.

Na figura 26 foram localizados 3 objetos de 4 possíveis, mostrando uma precisão de 75% em relação a quantidade de objetos e 100% em relação aos objetos detectados.

Figura 27 – Detecção de objetos de cenários reais com imagens do *Google Images*.

Fonte: O Autor.

Na figura 27 foram localizados 2 objetos de 4 possíveis, mostrando uma precisão de 50% em relação a quantidade de objetos e 66.6% em relação aos objetos detectados, uma vez que a detecção da placa foi ambígua, sobressaindo-se o resultado de maior probabilidade.

Figura 28 – Detecção de objetos de cenários reais com imagens do *Google Images*.

Fonte: O Autor.

Na figura 28 foram localizados 9 objetos de 30 possíveis, mostrando uma precisão de 30% em relação a quantidade de objetos e 100% em relação aos objetos detectados.

Figura 29 – Detecção de objetos de cenários reais com imagens do *Google Images*.

Fonte: O Autor.

Na figura 29 foram localizados 7 objetos de 17 possíveis, mostrando uma precisão de 41% em relação a quantidade de objetos e 100% em relação aos objetos detectados.

Um resumo dos testes de cenários reais pode ser encontrado na tabela 15

Tabela 15 – Tabela demonstrando os resultados dos cenários reais.

Cenários	Total de Objetos	Qtd Detecções	% de Detecções	% de Classificações	Tempo(ms)
01	4	3/4	75%	100%	31
02	3	3/3	100%	100%	29
03	4	3/4	75%	100%	30
04	4	2/4	50%	66,6%	29
05	30	9/30	30%	100%	31
06	17	7/17	41%	100%	32

Fonte: O Autor

5.2 Resultados

5.2.1 Ambiente virtual

Em relação ao ambiente virtual houve uma localização de 17,9[1]% de objetos, o qual 83[2]% destes foram classificados corretamente. É possível perceber também na tabela 12 que a iluminação possui uma grande relevância nos ambientes, como demonstrado, cenários com maior ou melhor iluminação possuem mais objetos detectados, enquanto cenários com iluminação mais precária possuem piores resultados.

- No cenário 1, que possuía veículos, pedestres, placas e motocicletas, todos foram detectados, mas não necessariamente ao mesmo tempo, fazendo assim que o número total de detecções caia em relação ao valor total possível;
- No cenário 2, que possuía apenas placas de regulamentação e uma placa de serviço auxiliar, foram detectadas a placas de regulamentação, mas não a de serviços auxiliares;
- No cenário 3, que possuía pedestres, motocicletas e placas de advertência, em que propositalmente o vídeo foi acelerado, para ver se eram possíveis detecções em alta velocidade, foram detectados menos objetos, e talvez pelo fato da câmera não estar orientada diretamente para as placas, nenhuma placa foi detectada;
- No cenário 4, que possuía apenas pedestres, obteve uma taxa maior de sucesso, demonstrando que a rede consegue discernir bem o que é um pedestre;
- No cenário 5, que possuía apenas pedestres, uma placa de pare e uma placa de identificação ,obteve também uma taxa maior de sucesso, demonstrando mais uma vez que a rede consegue discernir bem o que é um pedestre e a placa de pare;
- No cenário 6, que possuía apenas placas de serviços auxiliares não conseguiu detectar nada, com exceção de quando confundiu uma de suas detecções, indicando que talvez a base precise de mais imagens de placas de serviços auxiliares e/ou um treinamento maior para conseguir discernir as mesmas.

[1] Esse número foi atingido fazendo uma média com os percentuais das células da tabela 13

[2] Esse número foi atingido fazendo uma média com os percentuais das células da tabela 14

5.2.2 Cenários reais

Em relação ao ambiente virtual houve uma localização de 94[3]% de objetos, o qual 61,8[4]% destes foram classificados corretamente. Em relação a rodovias, como geralmente há um número menor de elementos a ser detectados, a base consegue se sair bem. Já em relação ao trânsito dentro de cidades é possível perceber uma queda no número de detecções, mas também deve-se levar em conta que o número de elementos é muito superior ao de uma rodovia.

- No cenário 1, que possuía veículos e placas, foram detectados a placa e dois dos três veículos, ficando de fora um caminhão que estava bem ao fundo;
- No cenário 2, que possuía veículos e placas, foram detectados todos corretamente;
- No cenário 3, que possuía um motociclista, veículos e placas, foram detectados o motociclista, a placa e um dos dois veículos, ficando de fora um carro que estava bem ao fundo;
- No cenário 4, que possuía um motociclista, veículos e placas, foram detectados a placa e um veículo, ficando de fora um carro que estava bem ao fundo e o motociclista, além de ter gerado uma ambiguidade na placa;
- No cenário 5, que possuía motociclistas, veículos, pedestres e semáforos, foram detectados apenas alguns veículos e o pedestre, ficando de fora diversos veículos, os semáforos e os motociclistas;
- No cenário 6, que possuía placas, motociclistas, veículos e semáforos, foram detectados alguns veículos, um motociclista, uma moto, os semáforos e uma moto, ficando de fora diversos veículos, um motociclista e uma placa.

5.3 Comparação com trabalhos relacionados

O trabalho de Martins (2017) faz uso de métodos do OpenCV, para detecção de 5 placas de regulamentação, com uma rede neural perceptron de múltiplas camadas, utilizando 250 imagens para treino, com um tempo total de 7,5h de treinamento. Ele conseguiu atingir uma classificação de 96.6% na classificação e um tempo de predição de 93 ms com imagens.

O trabalho de Sobrinho et al. (2016) faz uso de Deep Learning para a detecção de 19 placas de trânsito, sendo 17 de regulamentação e 2 de advertência. Foram utilizadas 1300 imagens sintéticas, sendo 533 para treinamento, 267 para validação e 500 para testes. Seu modelo obteve uma classificação de 99.98%, mas não foram testadas imagens de cenários reais.

[3] Esse número foi atingido fazendo uma média com os percentuais das detecções da tabela 15
[4] Esse número foi atingido fazendo uma média com os percentuais das classificações da tabela 15

Em comparação com trabalhos relacionados, este trabalho consegue uma precisão geral de 61,8% na classificação, com um conjunto maior de placas de trânsito, em cenários reais com uma média de 30 fps.

6 Conclusão

Para resolver o problema proposto, foram adquiridas e criadas imagens, além de catalogadas, para a criação de uma base de dados e validadas com técnicas de *Deep Learning*, juntamente com uma *API*. Dada a complexidade da tarefa em questão, teve-se a necessidade de compreender o fenômeno *Deep Learning*, que faz parte de um subconjunto do *Machine Learning*, que é amplamente utilizado nas áreas de pesquisas de veículos autônomos.

Assim, este trabalho desenvolveu uma base de dados e a treinou para validá-la. Durante os testes foi percebido que a base consegue prover um alto índice de acertos na classificação dos objetos e um índice ainda ruim em relação à quantidade de objetos detectados por quadro. A API consegue uma taxa média de 30 FPS com uma resolução interna da rede de 608x608, com o hardware utilizado, o que significa que hardwares melhores conseguem taxas de quadros maiores, possibilitando até mesmo aumentar a resolução interna da rede, que aumenta a capacidade de detecções, sem a perda de desempenho.

Com o que foi desenvolvido até aqui, acredita-se que o problema proposto na seção 1.1 foi resolvido considerando as limitações encontradas.

6.1 Considerações e limitações

Durante o treinamento e os testes foram constatadas algumas limitações:

- A rede *Yolov3* não consegue rastrear a movimentação dos objetos de um quadro a outro como faz o processamento de visão em seres humanos, fazendo com que novas detecções tenham que ser feitas a cada mudança de quadro;
- Também em decorrência da detecção quadro a quadro sem rastreamento do objeto é que de um quadro para o outro o mesmo objeto pode deixar de ser detectado, principalmente quando a imagem apresenta elevada densidade de objetos;
- etapa de verificação individual e indicação manual de objetos presentes em cada imagem mostrou-se determinante para o sucesso das detecções. A partir dessas orientações manuais que a rede DL identifica quais são os padrões a ser analisados e incorporados e quais são os elementos que devem ser desconsiderados;
- A geração e análise de relatórios não está implementada de forma suficiente nas ferramentas utilizadas, demandando ajustes próprios nas ferramentas capazes de analisar a massa de dados produzida.

A base de dados resultante deste trabalho representa um dos primeiros esforços de mapeamento da sinalização de trânsito brasileira para utilização em VC, algo essencial para

viabilizar que veículos autônomos e similares possam trafegar nas vias. Neste contexto, trabalhos podem surgir como: expansões da base de dados e aprimoramentos para produzir resultados ainda melhores tanto em abrangência quanto na precisão da detecção de objetos.

6.2 Trabalhos futuros

Como trabalhos futuros ou melhorias, destacam-se:

- Treinar novamente a rede a partir do zero, para poder comparar a precisão na classificação e quantidade de objetos encontrados, com o modelo atual, para verificar qual seria o mais recomendado;
- A implementação de algum algoritmo *tracking*, desde que o mesmo consiga diminuir o tempo de processamento, uma vez que diminui o número de detecções realizadas;
- O treinamento utilizando outra *API* como, por exemplo, *tensorflow* para poder determinar a consistência da base de dados, ou se existe alguma *API* melhor que deveria ser utilizada;
- A implementação do treinamento em algum protótipo para a avaliação do comportamento em tempo real;

Referências

ALEXEYAB. *Darknet*. 2017. Disponível em: <https://github.com/AlexeyAB/darknet>. Citado nas páginas 32, 41, 42, 43, 45 e 47.

ALEXEYAB. *Yolo mark*. 2017. Disponível em: <https://github.com/AlexeyAB/Yolo_mark>. Citado na página 40.

BLENDER.ORG. *Blender 3D*. 2019. Disponível em: <https://www.blender.org/>. Citado na página 16.

BRADSKI, G.; KAEHLER, A. *Learning OpenCV: Computer Vision in C++ with the OpenCV Library*. 2nd. ed. [S.l.]: O'Reilly Media, Inc., 2013. ISBN 1449314651, 9781449314651. Citado na página 15.

BRAGA, A. P.; CARVALHO, A. P. L. F.; LUDERMIR, T. B. *Redes Neurais Artificiais: Teoria e Aplicações*. 2000. Citado na página 15.

BULLOCK, J.; CUESTA-LAZARO, C.; QUERA-BOFARULL, A. Xnet: A convolutional neural network (CNN) implementation for medical x-ray image segmentation suitable for small datasets. *CoRR*, abs/1812.00548, 2018. Disponível em: <http://arxiv.org/abs/1812.00548>. Citado na página 12.

COCO. *COCO - Common Objects in Context*. 2018. Disponível em: <http://cocodataset.org/#download>. Citado nas páginas 13, 42 e 46.

EPIC. *Unreal Engine 4 - IA*. 2018. Disponível em: <https://www.unrealengine.com/en-US/techblog/ai-in-unreal-engine-learning-through-virtualsimulations>. Citado na página 50.

EPIC. *Unreal Engine 4*. 2019. Disponível em: <https://www.unrealengine.com/en-US/features>. Citado na página 16.

GERHARDT, T. E.; SILVEIRA, D. T. *Métodos de Pesquisa*. 2009. Disponível em: <http://www.ufrgs.br/cursopgdr/downloadsSerie/derad005.pdf>. Citado na página 28.

GOOGLE. *reCAPTCHA: Easy on Humans, Hard on Bots*. 2019. Disponível em: <https://www.google.com/recaptcha/intro/v3.html#introducing-new-recaptcha>. Citado na página 12.

HE, K. et al. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. Disponível em: <http://arxiv.org/abs/1512.03385>. Citado na página 25.

HUVAL, B.; COATES, A.; NG, A. Y. Deep learning for class-generic object detection. *CoRR*, abs/1312.6885, 2013. Disponível em: <http://arxiv.org/abs/1312.6885>. Citado na página 24.

HUVAL, B. et al. An empirical evaluation of deep learning on highway driving. *CoRR*, abs/1504.01716, 2015. Disponível em: <http://arxiv.org/abs/1504.01716>. Citado na página 22.

KARRAS, T.; LAINE, S.; AILA, T. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018. Disponível em: <http://arxiv.org/abs/1812.04948>. Citado na página 24.

KRAUSE, J. et al. 3d object representations for fine-grained categorization. In: *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*. Sydney, Australia: [s.n.], 2013. Citado na página 39.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. USA: Curran Associates Inc., 2012. (NIPS'12), p. 1097–1105. Disponível em: <http://dl.acm.org/citation.cfm?id=2999134.2999257>. Citado nas páginas 12, 17, 23 e 24.

LECUN, Y.; BENGIO, Y.; HINTON, G. E. Deep learning. *Nature*, v. 521, n. 7553, p. 436–444, 2015. Disponível em: <https://doi.org/10.1038/nature14539>. Citado nas páginas 13, 17, 18, 19 e 20.

LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2324, Nov 1998. ISSN 0018-9219. Citado nas páginas 13, 17, 18 e 23.

MARTINS, M. A. *Identificação de Placas de Trânsito Através da Classificação de Imagens Usando Redes Neurais Artificiais.* 2017. Disponível em: <https://repositorio.ufu.br/bitstream/123456789/19551/1/IdentificacaoPlacasTransito.pdf>. Citado na página 60.

MUELLER, M. et al. Ue4sim: A photo-realistic simulator for computer vision applications. *CoRR*, abs/1708.05869, 2017. Disponível em: <http://arxiv.org/abs/1708.05869>. Citado na página 27.

NVIDIA. *CUDA Zone*. 2018. Disponível em: <https://developer.nvidia.com/cuda-zone>. Citado nas páginas 16 e 29.

NVIDIA. *NVIDIA cuDNN*. 2018. Disponível em: <https://developer.nvidia.com/cudnn>. Citado nas páginas 16 e 29.

RAJPURKAR, P. et al. Driverseat: Crowdstrapping learning tasks for autonomous driving. *CoRR*, abs/1512.01872, 2015. Disponível em: <http://arxiv.org/abs/1512.01872>. Citado na página 22.

REDMON, J. *darknet*. 2018. Disponível em: <https://pjreddie.com/darknet/yolo/>. Nenhuma citação no texto.

REDMON, J. et al. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. Disponível em: <http://arxiv.org/abs/1506.02640>. Citado nas páginas 20, 21 e 25.

REDMON, J.; FARHADI, A. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016. Disponível em: <http://arxiv.org/abs/1612.08242>. Citado na página 26.

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. Disponível em: <http://arxiv.org/abs/1804.02767>. Citado nas páginas 26, 31 e 32.

SERMANET, P. et al. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013. Disponível em: <http://arxiv.org/abs/1312.6229>. Citado nas páginas 20 e 24.

SHAH, S. et al. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In: *Field and Service Robotics*. [s.n.], 2017. Disponível em: <https://arxiv.org/abs/1705.05065>. Citado nas páginas 27 e 50.

SOBRINHO, M. V. O. et al. *Reconhecimento de Sinais de Transito Utilizando Deep Learning.* 2016. Disponível em: <http://sistemas.deinf.ufma.br/anaisjim/artigos/2016/201622.pdf>. Citado na página 60.

STALLKAMP, J. et al. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In: *IEEE International Joint Conference on Neural Networks*. [S.l.: s.n.], 2011. p. 1453–1460. Citado na página 13.

SYSTEM, E. *What is Machine Learning? A definition.* 2017. Disponível em: <https://www.expertsystem.com/machine-learning-definition/>. Citado na página 12.

TECHOPEDIA. *What is Computer Vision? - Definition.* 2019. Disponível em: <https://www.techopedia.com/definition/32309/computer-vision>. Citado na página 12.

TECHOPEDIA. *What is Deep Learning? - Definition.* 2019. Disponível em: <https://www.techopedia.com/definition/30325/deep-learning>. Citado na página 12.

VISICS. *Traffic Sign Recognition*. 2009. Disponível em: <http://www.vision.ee.ethz.ch/~timofter/traffic_signs/>. Citado na página 13.

WORCESTER POLYTECHNIC, I. *WPI Datasets*. 2018. Disponível em: <http://computing.wpi.edu/dataset.html>. Citado na página 39.

ZEILER, M. D.; FERGUS, R. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013. Disponível em: <http://arxiv.org/abs/1311.2901>. Citado na página 24.

APÊNDICE A – Apêndices

A.1 Objetos

Figura 30 – Imagens de cada classe respectivamente.

Fonte: O Autor.