CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS TIMÓTEO

Marcos Paulo Cardoso Soares Júnior

ESTUDO DE MODELOS DE DEEP LEARNING APLICADOS AO RECONHECIMENTO DAS PLACAS DE TRANSITO BRASILEIRAS UTILIZANDO HARDWARES DE BAIXO CUSTO

Timóteo

2021

Marcos Paulo Cardoso Soares Júnior

ESTUDO DE MODELOS DE DEEP LEARNING APLICADOS AO RECONHECIMENTO DAS PLACAS DE TRANSITO BRASILEIRAS UTILIZANDO HARDWARES DE BAIXO CUSTO

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

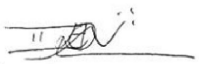
2021

Marcos Paulo Cardoso Soares Júnior

Estudo de modelos de Deep Learning aplicados ao reconhecimento das placas de trânsito brasileiras utilizando hardwares de baixo custo

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais, campus Timóteo, como requisito parcial para obtenção do título de Engenheiro de Computação.

Trabalho aprovado. Timóteo, 09 de abril de 2021:



Prof. Dr. Elder de Oliveira Rodrigues
Orientador

Prof. Me. Douglas Nunes de Oliveira
Professor Convidado

Prof. Me. Nelson Alexandre Estevão
Professor Convidado

Timóteo
2021

Agradecimentos

Agradeço à Deus pela minha saúde e por ter chegado até aqui. Agradeço aos meus pais por toda a ajuda que me deram quando eu muito precisei. Agradeço aos meus professores por me ajudarem a manter a disciplina e concluir este trabalho em plena época em que o mundo vivia uma pandemia por um vírus.

Resumo

Nos últimos anos, os termos Machine Learning e Deep Learning têm se tornado altamente populares no meio acadêmico e industrial dados os avanços no desenvolvimento de hardwares mais eficientes e em novos modelos de machine learning que fazem uso de redes neurais artificiais, culminando no desenvolvimento de tecnologias de alto valor agregado, como os chatbots em suas diversas aplicações, além dos sistemas que fazem uso da visão computacional como os veículos autônomos. Se tratando especificamente das aplicações de processamento e análise de imagens pelos algoritmos de Machine Learning e Deep Learning, grandes empresas como Google e Uber investiram na criação de carros autônomos, capazes de dirigir pelas ruas sem um motorista no comando. Entretanto, para que um modelo computacional detector de placas de trânsito de uma nacionalidade seja capaz de ser aplicado em larga escala industrial, ele necessita de ter uma alta assertividade combinadas com um custo baixo de produção a manutenção, de forma que o mercado consumidor do mesmo seja o maior possível. Mediante aos fatos expostos, foi desenvolvido neste trabalho um estudo comparativo entre os modelos de Deep Learning SSD Resnet-50 v1 FPN(RetinaNet) e o SSD Mobilenet(MobileDet), com uma base de dados de 500 imagens do trânsito brasileiro coletadas através do Google Maps, capazes de serem executadas em tempo hábil num Nvidia Jetson Nano e com uma precisão aceitável para o reconhecimento das placas de trânsito brasileiras a serem detectadas. De acordo com a métrica F1(Média harmônica), para a melhor placa de trânsito a ser detectada, o modelo RetinaNet teve uma precisão de 0.567 com uma média de 1.8 segundos de inferência numa imagem, enquanto que o MobileDet teve uma precisão de 0.694 com uma média de 1.104 segundos e inferência por imagem. Os resultados obtidos indicam que o mesmo ainda é insuficiente para ser usado num carro autônomo real devido à baixa precisão e tempo relativamente alto para cada inferência, entretanto, dado ao tamanho relativamente pequeno da base de dados quando se comparada com bases famosas como a COCO, o mesmo se mostra promissor para futuras melhorias a fim de ser usado em em carros autônomos, capazes de serem vendidos em larga escala.

Palavras-chave:Inteligência artificial,veículos autônomos, Machine Learning.

Abstract

In recent years, the terms Machine Learning and Deep Learning have become highly popular in academia and industry given the advances in the development of more efficient hardware and new models of machine learning that make use of artificial neural networks, culminating in the development of technologies of high added value, such as chatbots in their various applications, in addition to systems that make use of computer vision such as autonomous vehicles. When dealing specifically with image processing and analysis applications using Machine Learning and Deep Learning algorithms, large companies such as Google and Uber have invested in the creation of autonomous cars, capable of driving on the streets without a driver in charge. However, for a computational model to detect traffic signs from a given nationality and also being able to be used on a large industrial scale, it must have a high assertiveness combined with a low cost of production and maintenance, so that the consumer market of the same be as large as possible. Based on the exposed facts, a comparative study was developed in this work between the Deep Learning SSD Resnet-50 v1 FPN (RetinaNet) models and the Mobilenet SSD (MobileDet), with a database of 500 brazilian traffic images collected through Google Maps, capable of being executed in a timely manner on an Nvidia Jetson Nano and with acceptable accuracy for the recognition of Brazilian traffic signs to be detected. According to the metric F1 (Harmonic mean), for the best traffic sign to be detected, the RetinaNet model had an accuracy of 0.567 with an average of 1.8 seconds of inference on an image, while MobileDet had an accuracy of 0.694 with an average of 1,104 seconds and inference per image. The results obtained indicate that it is still insufficient to be used in a real autonomous car due to the low precision and relatively high time for each inference was made, however, given the relatively small size of the database when compared to famous bases such as COCO, it seems promising for future improvements and to be used in autonomous cars, capable of being sold on a large scale.

Keywords:Artificial Inteligence,autonomous vehicles, Machine Learning

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ANN: Artificial Neural Network

CNN: Convolutional Neural Network

ML: Machine Learning

DL: Deep Learning

AI: Artificial Inteligence – (IA) Inteligência Artificial

FPS: Frames per second – (QPS) Quadros por segundo

CV: Computer Vision –(VC) Visão computacional

Sumário

1 Introdução

Um dos campos da computação que vem ganhando destaque é o Machine Learning (ML). O Machine Learning é uma subárea de inteligência artificial (IA) que fornece aos sistemas a capacidade de aprender e melhorar automaticamente a partir da experiência sem ser explicitamente programado, aprendendo através de dados informados ao modelo. (TEAM, 2020). Muitos especialistas ao redor do mundo consideram que o ML e a IA vêm transformando o mundo(WEST; ALLEN, 2018) devido a sua enorme aplicabilidade em diferentes campos de conhecimento, melhorando processos e reduzindo erros comumente causados por humanos por problemas como cansaço e falta de atenção, além da falta de um viés emocional ao tomar decisões.

O Machine Learning juntamente com o Deep Leaning(DL) vêm trazendo melhorias significativas de resultados e, até mesmo o avanço do estado da arte com novas variações das redes neurais convolucionais(CNN) (TOUVRON et al., 2019). A visão computacional vem recebendo muitos usos de ponta na indústria devido a melhoria de resultados com o ML e DL, trazidos com modelos que usam CNN como aconteceu em março de 2020, quando os pesquisadores do Hospital Renmin da Universidade de Wuhan, da Wuhan EndoAngel Medical Technology Company e da Universidade de Geociências da China compartilharam um trabalho desenvolvido utilizando uma base de dados de aproximadamente 46 mil imagens, com uma precisão de 95.24% em detectar o novo Coronavírus nas imagens utilizadas.(CHEN et al., 2020)

O Deep Learning(DL) mencionado anteriormente é categorizado como um subconjunto do Machine Learning(ML) e também é amplamente utilizado em conjunto com a Visão Computacional (VC) em algoritmos como as redes neurais convolucionais e suas variações(SAHA, 2018). A visão computacional é um campo da ciência da computação que trabalha para permitir que os computadores vejam, identifiquem e processem imagens assim como faz a visão humana e, depois forneçam resultados apropriados (TECHOPEDIA, 2019). O trabalho de Krizhevsky, Sutskever e Hinton (2012), mostra que, o Deep Learning aplicado à visão computacional traz resultados expressivos, visto que a combinação de ambos gerou um algoritmo capaz de vencer uma competição internacional de visão computacional (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

Assim como no caso da universidade de Wuhan, da Wuhan EndoAngel Medical Technology Company e da Universidade de Geociências da China, para o treinamento de algoritmos que usam Inteligencia Artificial, mais especificamente Machine Learning e Deep Learning, necessitam de uma grande base de dados, variando de milhares até bilhões de amostras, como foi o caso dos carros autônomos da startup Waymo. Esse conjunto de imagens é utilizado para fazer com que a máquina consiga reconhecer padrões nas imagens de treinamento, abstrair tais padrões e utilizá-las posteriormente. Empresas como o Google utilizam os usuários da internet para fazerem a curadoria e validação das imagens de sua base de dados em mecanismos com o Recaptcha (GOEDEGEBUURE, 2016),que requer que o usuário clique em imagens que contenham o objeto desejado, utilizando-o como uma ferramenta padrão para autenticação em duas etapas em muitos sistemas por toda a internet.(GOOGLE, 2019). Dessa forma, o Google consegue treinar os seus algoritmos de Aprendizado de máquina e visão computacional utilizando os usuários da rede como "validadores"dos dados.

Um campo de atuação em que existe um crescimento de aplicações e estudos com ML e DL é na criação de carros autônomos, sendo considerado um tópico emergente para pesquisadores e cientistas(DAS; MISHRA, 2019). Empresas como Tesla, Uber, Volvo e outras ligadas ao setor automobilístico vêm trabalhando para a criação de veículos autônomos de preço acessível à grande massa de consumidores de veículos ao redor do mundo.Sobre o assunto, Subhranil Das e Sudhansu Kumar Mishra (2019, p.500) afirmam:

> Google começou um projeto de carro autônomo em 2009. Durante a última década, muitas empresas como a Uber, Google, Mercedes, Ford e muitos mais estão investindo muito dinheiro em AUGV(Veículos autônomos não tripulados) totalmente automatizado.Espera-se que até o ano 2050, tal fator gere cerca de 7 trilhões de dólares de receita anual,bem como como cerca de 10 milhões de veículos autônomos que pegariam a estrada. No entanto, existem muitos problemas que persistem durante a implementação de novos tecnologias em AUGV's. Portanto, uma substancial quantidade de pesquisa ainda precisa ser feita por completo para o desenvolvimento de veículos terrestres automatizados.

(DAS; MISHRA, 2019) (tradução nossa).

A grande maioria dos carros autônomos modernos sendo desenvolvido pelas grandes indústrias automotivas e da tecnologia, utilizam Lidars como sensores primários de informação aliados às câmeras de captura para reconhecimento do mundo ao redor do veículo além de vários outros sensores para detecção de informações diferentes, fazendo com que o veículo reconheça o ambiente ao seu redor (HECHT, 2018). Um dos problemas atuais que faz com

que os carros autônomos sejam tão caros é o fato de que os Lidars seja uma tecnologia muito cara e que inviabiliza a popularização dos mesmos (ACKERMAN, 2016).

De acordo com os dados mencionados anteriormente, é válido a indagação se é possível criar um algoritmo de Inteligência Artificial, capaz de detectar com precisão aceitável as placas de trânsito brasileiras e que, pode ser executado num computador a custo razoável de até aproximadamente 1000 reais ?

1.1 Justificativa e Problema

Este trabalho se justifica pela necessidade industrial devido aos altos preços de hardware para carros autônomos como os Lidars (ACKERMAN, 2016) e acadêmica de se criar um sistema que seja capaz de reconhecer as placas de trânsito brasileiras em tempo hábil e que possa ser executada num computador de custo relativamente baixo. Além disso, este trabalho é uma continuação proposta pelo ex-aluno de Engenharia de Computação Diego Haji, que começou o estudo de Inteligência Artificial para placas de trânsito brasileiras e que sugeriu outras ferramentas como o TensorFlow(que será usado neste trabalho) para a detecção e reconhecimento de placas de transito brasileiras. O trabalho proposto e desenvolvido aqui possui diversas aplicações, como por exemplo o que pode ser considerado atualmente o maior deles, que é o desenvolvimento de carros autônomos capazes de circularem pelo trânsito brasileiro.

1.2 Objetivos

O objetivo geral deste trabalho é desenvolver um classificador de placas de trânsito brasileiras, capaz de executar em um NVidia Jetson Nano, com um custo de aproximadamente R$ 1000 reais na cotação do dia 28/11/2020. Além disso, este trabalho é uma continuação do trabalho de (CAMPOS, 2019), com um foco maior nas placas de trânsito e a execução em um hardware inferior ao utilizado por ele.

Para resolver o problema anteriormente proposto, será necessário adquirir uma base de dados com informações do trânsito brasileiro, catalogar e marcar manualmente as placas de trânsito encontradas na imagem, para ser posteriormente utilizada nos algoritmos e técnicas de Machine Learning/Deep Learning.

Para o desenvolvimento do trabalho, será necessário:

1. Criar e catalogar uma base de dados com uma quantidade suficientemente grande de amostras do trânsito brasileiro com as placas escolhidas para detecão.
2. Criar e treinar um modelo de Machine Learning/Deep Learning capaz de reconhecer as placas de trânsito escolhidas num computador de relativo baixo custo, com precisão aceitável e num tempo hábil.
3. Testar e validar todo o projeto construído com técnicas estatísticas.

2 Revisão Bibliográfica

O objetivo central deste capítulo é contextualizar o leitor sobre os conceitos teóricos utilizados durante todo o trabalho na área de Inteligência Artificial, Machine Learning, Deep Learning Visão computacional e os conceitos relacionados aos carros autônomos.

2.1 Fundamentos teóricos

O objetivo deste trabalho é produzir uma base de dados e um modelo classificador das placas de trânsito brasileiras, que possa ser futuramente utilizada em outros trabalhos acadêmicos e industriais como por exemplo, no desenvolvimento de carros autônomos. Os próximos tópicos se tratam de uma explicação de conceitos fundamentais utilizados para o desenvolvimento e contextualização deste trabalho.

2.1.1 Inteligencia artificial

De acordo com Russel Norvig (2013,p.24):

> Denominamos nossa espécie Homo sapiens — homem sábio — porque nossa inteligência é tão importante para nós. Durante milhares de anos, procuramos entender como pensamos, isto é, como um mero punhado de matéria pode perceber, compreender, prever e manipular um mundo muito maior e mais complicado que ela própria. O campo da inteligência artificial, ou IA, vai ainda mais além: ele tenta não apenas compreender, mas também construir entidades inteligentes.

> Atualmente, a IA abrange uma enorme variedade de subcampos, do geral (aprendizagem e percepção) até tarefas específicas, como jogos de xadrez, demonstração de teoremas matemáticos, criação de poesia, direção de um carro em estrada movimentada e diagnóstico de doenças. A IA é relevante para qualquer tarefa intelectual; é verdadeiramente um campo universal

(RUSSEL, 2013).

2.1.2 Redes Neurais Artificiais

De acordo com Antônio de Pádua Braga(2000, p.8)

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 uma 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. Esta 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 ANDRé PONCE DE LEON E DE CARVALHO, 2007).

2.1.3 Machine Learning

A computação tradicional baseia-se na idéia de se programar uma máquina para se executar uma determinada tarefa. Entretanto, as dificuldades encontradas para se codificar sistemas com regras complexas faz com que exista a necessidade de que os sistemas extraiam as próprias regras atraves de inferências obtidas pelos dados informados ao algoritmo. Esta capacidade é conhecida como Machine Learning(GOODFELLOW; BENGIO; COURVILLE, 2016).

De acordo com Tom Mitchell (MITCHELL, 1997), uma perspectiva interessante a respeito de Machine Learning é quando se analisa um grande espaço de hipóteses para determinar qual das mesmas se encaixa melhor com os dados observados e com o conhecimento pré adquirido do leitor.

Quando estamos trabalhando em um problema que envolve uma solução de Machine Learning, alguns problemas precisam ser resolvidos a fim de se alcançar a solução para o problema em questão, alguns desses problemas são: (MITCHELL, 1997):

- Quais algoritmos de machine learning para aprendizado genérico(tais como Redes Neurais, SVM, KNN Árvores de decisão e outros) se encaixam para a resolução do problema em questão?
- Qual a quantidade necessária de dados para o treinamento do algoritmo ?
- Como e quando um conhecimento prévio adquirido pelo leitor pode guiar o processo de generalização ?
- O modelo é capaz de alterar sua representação interna do problema a fim de melhor representar e mapear a função geradora do problema em questão ?

2.1.4 Deep Learning

A idéia de aprender a representação correta para os dados fornece uma boa perspectiva especialmente em Deep Learning(aprendizagem profunda). Outra perspectiva sobre

a aprendizagem profunda é que a profundidade permite que o computador aprenda um programa de computador de várias etapas. Cada camada da representação pode ser pensada como o estado da memória do computador, quando o mesmo executa um outro conjunto de instruções em paralelo. Redes com maior profundidade podem executar mais instruções sequencialmente(GOODFELLOW; BENGIO; COURVILLE, 2016) e que, em computadores mais modernos, podem ter parte do processo paralelizado atráves do uso de placas de vídeo. Quando criamos uma rede neural que possui centenas e até milhares de camadas, a mesma é considerada um modelo de Deep Learning.

De acordo com (NVIDIA, 2016), a melhor maneira de demonstrar a diferença entre Inteligência Artificial, Machine Learning e Deep Learning é visualmente como demonstrado na figura abaixo:

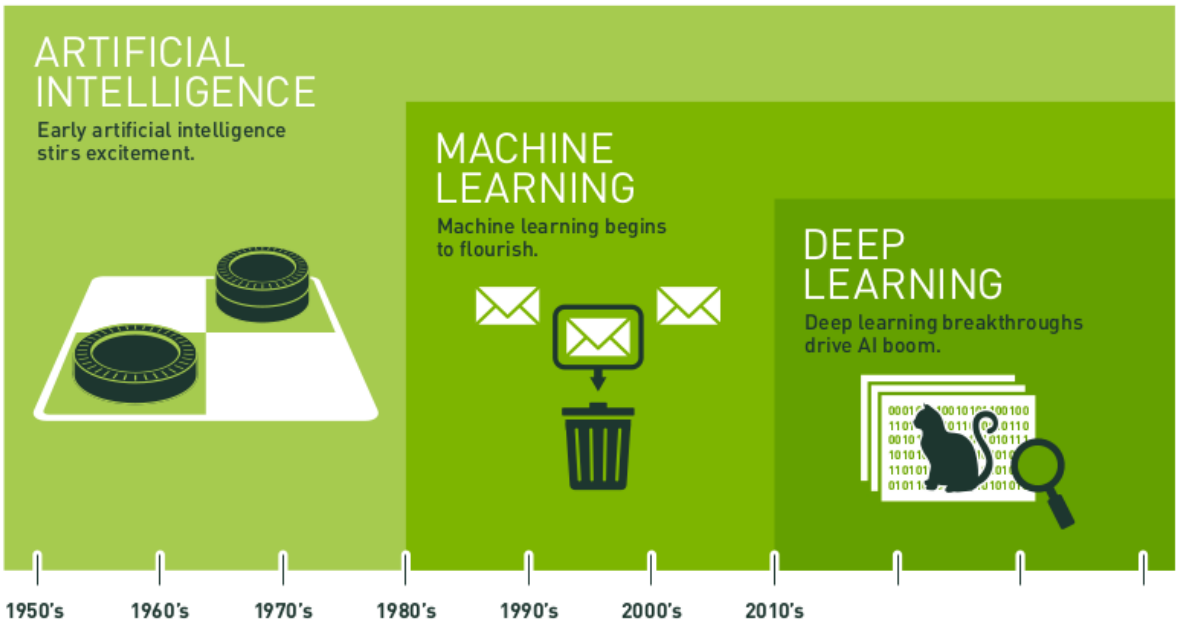


Figura 1 – Diferenças entre Inteligência Artifical,Machine Learning e Deep Learning Fonte:(NVIDIA, 2016)

A imagem 1 acima demonstra que Deep Learning é uma subárea de Machine Learning, que por sua vez é uma subárea da Inteligência Artificial.

2.1.5 Tensorflow

TensorFlow é uma plataforma Open-Source voltada para machine learning e deep learning. O TensorFlow possui um grande ecossistema ao seu redor, desde ferramentas e bibliotecas até uma grande comunidade de pesquisadores, doutores e indústrias utilizando a plataforma a fim de alcançar o estado da arte da Inteligência Artificial em seus produtos e serviços, além de simplificar a codificação dos desenvolvedores para os mesmos acelerarem o processo de prototipação e despache de um modelo de Inteligencia Artificial para uso. Tensorflow é utilizado por diversas empresas em todo o mundo como: AirBnb, Nvidia, Uber, SAP, Kakao, DeepMind, DropBox, Ebay, Google, Snapchat, Intel, Coca-Cola, Xiaomi, ZTE, Qualcomm, Twitter, Lenovo e outras para o desenvolvimento de sistemas inteligentes com alto poder de agregar valor em seus negócios.

2.1.6 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 módulo GPU também fornece versões com aceleração por CUDA em muitas de suas rotinas (para GPU's Nvidia) e as otimizações para OpenCL (para GPU's genéricas). Um dos objetivos do OpenCV é fornecer uma infraestrutura de visão computacional simples de usar que ajude as pessoas a construirem aplicações de visão computacional 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 o aprendizado de máquina e a visão computacional costumam estar juntos, o OpenCV também possui uma biblioteca de Aprendizado de Máquina (MLL) completa e de uso geral. Esta sub-biblioteca é especializada no reconhecimento de padrões estatísticos e agrupamento. A biblioteca MLL é de grande utilidade para as tarefas de visão computacional que fazem parte da ideia geral que o OpenCV proporciona, entretanto a mesma é generalista o suficiente para ser usada em qualquer problema considerado como aprendizado de máquina. (BRADSKI; KAEHLER, 2013).

2.1.7 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 as aplicações de computação, aproveitando o poder das GPU's. Em aplicações aceleradas por GPU, a parte sequencial(não paralelizável) das instruções do trabalho é executada na CPU, que é projetada para desempenho de encadeamento sequencial e único, enquanto a parte de computação intensiva da aplicação é executada em milhares de núcleos de GPU de forma paralela. Quando se usa CUDA, os desenvolvedores programam em linguagens populares como C/C++, Python e MATLAB e desenvolvem o paralelismo através de extensões na forma de algumas informações extras e palavras-chave básicas próprias da biblioteca. O CUDA Toolkit da NVIDIA fornece tudo o que é preciso para o desenvolvimento de aplicações aceleradas por GPU. O CUDA Toolkit inclui desde bibliotecas aceleradas por GPU, um compilador além ferramentas de desenvolvimento e o ambiente de execução CUDA (NVIDIA, 2018).

2.1.8 Redes Neurais Convolucionais

Devido a alta capacidade das redes neurais multicamada, utilizando o algoritmo de retroprogração(backpropagation) de aprender e mapear funções matemáticas complexas não lineares e de n-dimensões as torna um alternativa óbvia no que diz respeito às tarefas de reconhecimento de imagem vista a alta paridade da tarefa em questão. No modelo tradicional, exige-se que o programador extraia as características da imagem manualmente, selecionando os dados relevantes ao problema de forma manual. Dessa forma, um modelo treinado faz a categorização dos vetores de características obtidas resultantes em classes. Neste processo, as redes multicamada(MLP) totalmente conectadas podem ser utilizadas como classificadores[LECUN et al., 1998].

A arquitetura de uma Rede Convolucional típica, como demonstrado na figura 2, 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 conectada 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 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).

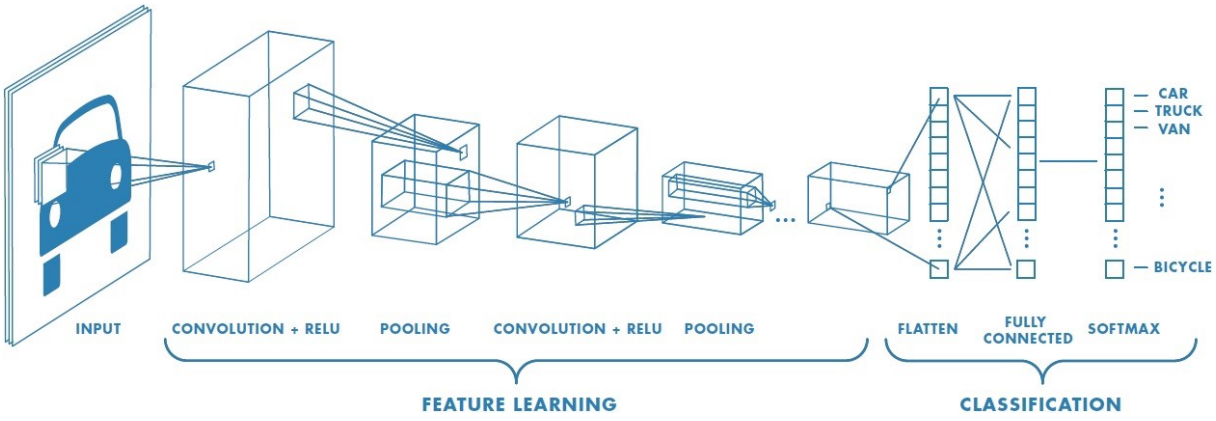


Figura 2 – Estrutura de uma rede neural convolucional comum. Fonte:(PRABHU, 2018)

2.1.9 SSD, Detector em uma única instância(Single Shot Detector)

Um modelo SSD necessita somente de uma imagem como entrada, e as suas marcações de classes durante o treinamento para a detecção de objetos em uma imagem. De acordo com o processo de convolução, cada marcação é analisado de acordo com os mapas de características em dimensões e escalas diferentes como demonstrado na figura 3.(LIU et al., 2016)

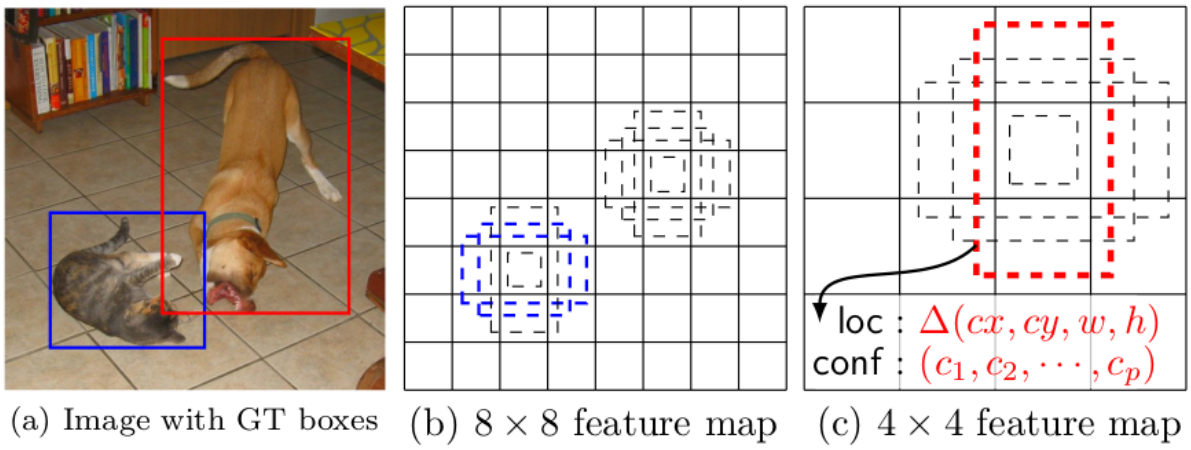


Figura 3 – Exemplo de uma análise feita por um modelo SSD analisando os possíveis candidatos de acordo com os mapas de atributos em diferentes escalas. Fonte:(LIU et al., 2016)

É importante mencionar que a abordagem SSD é baseada numa CNN que produz uma coleção de "bounding boxes"e taxas de precisão para os possíveis elementos detectados numa imagem numa quantidade fixa previamente definida (LIU et al., 2016).No trabalho em questão, foi escolhida uma quantidade de 100 possíveis bounding boxes para cada um dos modelos que utilizava a abordagem SSD.

2.1.10 Feature Pyramid Networks ou FPNs

As FPNs são considerados atributos básicos e essenciais para a detecção de objetos em escalas diferentes, que vêm sendo evitados devido ao alto custo computacional de sua execução. Entretanto, é importante mencionar que, a maior vantagem de se analisar as imagens em formato piramidal e em escalas diferentes é a produção de uma representação de atributos multi-escala,sendo semanticamente forte além de possuir uma maior robustez à variância de escalas diferentes para as classes a serem detectadas.A imagem 4 demonstra que um modelo FPN pode ser implementado na forma Bottom-UP(de baixo para cima) ou numa abordagem Top-Down(De cima para baixo), no qual cada abordagem possui suas vantagens e desvantagens.(LIN et al., 2016)

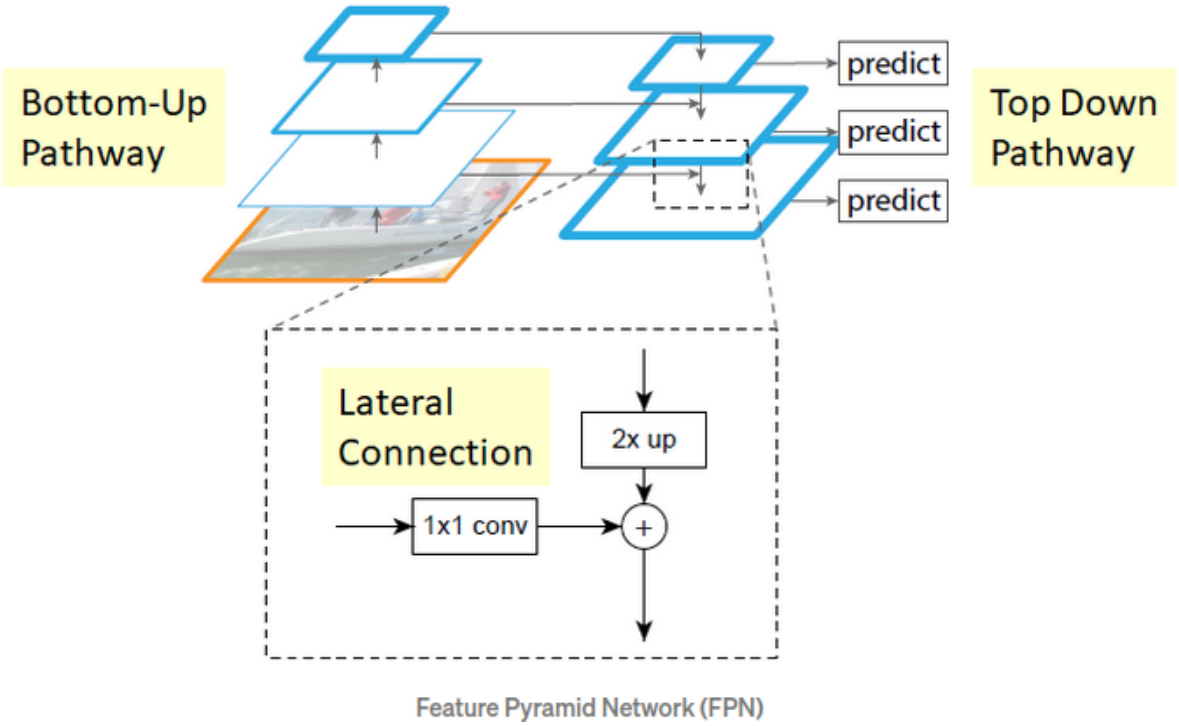


Figura 4 – Imagem demonstrando as duas formas de se implementar uma FPN: Top-Down e Bottom-UP além do processo de soma feito numa conexão lateral entre abordagens. Fonte:(LIN et al., 2016)

2.1.11 Google Street View

O Google Street View é uma representação virtual do mundo, composta de milhões de imagens panorâmicas catalogadas e distribuídas de forma que qualquer pessoa consiga conhecer com detalhes a paisagem de qualquer lugar do mundo que foi previamente catalogado. O conteúdo catalogado e disponível no Street View tem duas origens: o Google e seus colaboradores(GOOGLE, 2020)

2.1.12 Carros autônomos

Um carro autônomo é um veículo que consegue se locomover sem a necessidade de um ser humano no seu controle. Nas últimas décadas, muito foi pesquisado e desenvolvido no âmbito dos carros autônomos, também conhecidos por Driverless Cars ou carros sem motorista. O passo inicial foi dado pelo robô criado no laboratório de engenharia mecânica japonês de Tsukuba no final da década de 70. Este robô foi concebido em uma época onde os computadores ainda eram muito lentos e pesados, sendo criados hardwares específicos para ele. No

entanto, os primeiros carros robôs do mundo foram concebidos na Universidade Bundeswehr de Munique (UniBW) pelo professor Ernst Dickmanns (DICKMANNS; ZAPP, 1987) na década de 80. Seus veículos utilizavam desde visão com movimentos sacádicos a métodos probabilísticos, tais como o Filtro de Kalman, e computação paralela. Algumas fontes norte-americanas comparam Dickmanns ao irmãos Wright 1 , tamanha sua importância no desenvolvimento dos carros autônomos.

Com o avanço dos computadores portáteis e sensores, vários veículos mais complexos surgiram na década de 90. Nos Estados Unidos, a Universidade de Carnegie Mellon (CMU) apresentou o robô Navlab 5 com o intuito de realizar o projeto intitulado “No Hands Across America”, ou Sem as Mãos Percorrendo a America 2 . A viagem foi realizada em 1990, na qual o veículo percorreu 96% dos 490 km autonomamente, com uma velocidade média de 91 km/h. Outro exemplo, também deste período, é o projeto italiano ARGO (BROGGI A.AND BERTOZZI; GUARINO, 1999), que igualmente ao Navlab 5, foi capaz de realizar viagens em alta velocidade (média acima de 90 km/h), por longas distâncias (cerca de 2000 km) e a maior parte dela de forma autônoma (94% do tempo)(LIMA, 2010).

No entanto, os maiores avanços e contribuições ocorreram a partir do ano 2000,principalmente com os desafios propostos pela agência americana DARPA ((DARPA, 2020)) entre carros autônomos. Para superar estes desafios, diferentes técnicas foram desenvolvidas e aperfeiçoadas por pesquisadores de todo o mundo. As técnicas eram capazes de enfrentar as adversidades comuns aos veículos autônomos terrestres e realizar a sua navegação autônoma em ambientes desérticos e urbanos. Os desenvolvimentos foram tamanhos que veículos autônomos já podem ser comprados e aplicados em tarefas que apresentam risco aos seres humanos(JAMASMIE, 2009).

2.1.13 NVIDIA Jetson Nano

O Jetson Nano é um computador pequeno e poderoso para aplicativos incorporados e IoT de IA que fornece o poder da IA moderna em um módulo de US $ 129. O NVIDIA Jetson Nano permite o desenvolvimento de milhões de novos sistemas de AI, menores e com baixo consumo de energia. Ele traz novas possibilidades de aplicações integradas em IoT, incluindo gravadores de vídeo em rede (NVRs) de nível básico, robôs domésticos e gateways inteligentes com recursos de análise completos.(NVIDIA, 2020b)

2.2 Trabalhos relacionados

O conteúdo sobre Machine Learning, Deep Learning e processamento de imagens foi aprendido pelo autor via levantamento bibliográfico, cursos online pela empresa Udemy e competições do Google pela plataforma Kaggle. Os trabalhos obtidos via levantamento bibliográfico e embasamento foram obtidos através da busca de periódicos da CAPES e do Google Acadêmico.

2.2.1 Inteligência Artificial, Machine Learning e Deep Learning

2.2.1.1 McCulloch and Pitts

Em 1943 foi realizado o primeiro trabalho que foi categorizado como IA por Warren McCulloch e Walter Pitts. O conhecimento do trabalho foi baseado pelos autores via três fontes: o conhecimento da fisiologia básica e da função dos neurônios no cérebro humano; uma análise formal da lógica proposicional criada por Whitehead e Russell além da teoria da computação criada por Alan Turing. McCulloch e Pitts fizeram a proposta de um modelo de neurônios artificiais simulando, de forma simplificada os neurônios humanos, no qual cada neurônio possui dois estados possíveis caracterizados como “ligado” ou “desligado”, com a troca para “ligado” ocorrendo devido a uma resposta à estimulação por um número suficiente de neurônios vizinhos. O estado em que um neurônio se encontra era considerado, de acordo com Stuart Russel(2013, p.41) “equivalente em termos concretos a uma proposição que definia seu estímulo adequado”(RUSSEL, 2013). Dessa forma, eles mostraram, por exemplo, que qualquer função matematicamente computável poderia ser calculada por uma rede de neurônios interconectados e que todos os conectivos da lógica (e, ou, não etc.) podiam ser implementados através de estruturas projetadas por redes neuronais simples. Os dois pesquisadores também sugeriram que os modelos criados por eles, baseados em redes e definidas adequadamente, seriam capazes de aprender e se autocorrigir. Donald Hebb em 1949 demonstrou uma regra um tanto quanto simples de atualização para modificar as intensidades(definidas como pesos através de um número) de conexão entre cada neurônio. Sua regra, conhecida atualmente como aprendizado de Hebb ou aprendizado Hebbiano, continua a ser um modelo de grande influência até mesmo nos dias de hoje.(RUSSEL, 2013)

Devido ao caráter “tudo ou nada” da atividade nervosa, os eventos neurais e as relações entre eles podem ser tratados por meio da lógica proposicional. Verifica-se que o comportamento de toda rede pode ser descrito nesses termos, com a adição de meios lógicos mais complicados para redes contendo círculos; e que, para qualquer expressão lógica que satisfaça certas condições, é possível encontrar uma rede que se comporta da maneira que

descreve. Demonstra-se que muitas escolhas particulares entre possíveis suposições neurofisiológicas são equivalentes, no sentido de que para cada rede que se comporta sob uma suposição, existe outra rede que se comporta sob a outra e dá os mesmos resultados, embora talvez não ao mesmo tempo. Várias aplicações do cálculo são discutidas.(MCCULLOCH; PITTS, 1943)

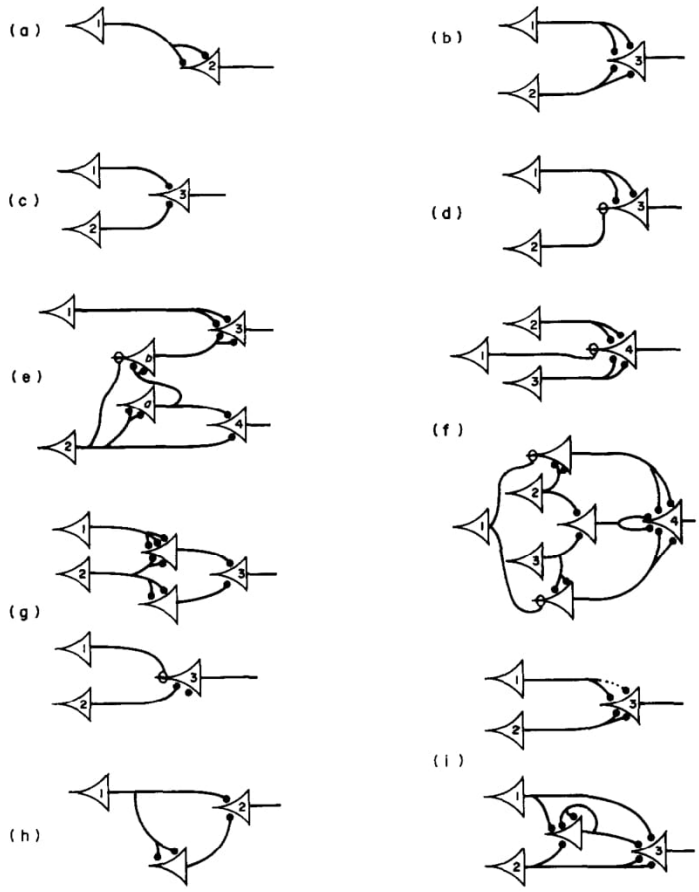


Figure 1. The neuron c_i is always marked with the numeral i upon the body of the cell, and the corresponding action is denoted by "N" with i s subscript, as in the text:

(a) $N_2(t) .\equiv . N_1(t-1)$;
(b) $N_3(t) .\equiv . N_1(t-1) \vee N_2(t-1)$;
(c) $N_3(t) .\equiv . N_1(t-1) . N_2(t-1)$;
(d) $N_3(t) .\equiv N_1(t-1) . \sim N_2(t-1)$;
(e) $N_3(t) :\equiv : N_1(t-1) . \vee . N_2(t-3) . \sim N_2(t-2)$;
$N_4(t) .\equiv . N_2(t-2) . N_2(t-1)$;
(f) $N_4(t) :\equiv : \sim N_1(t-1) . N_2(t-1) \vee N_3(t-1) . \vee . N_1(t-1) .$
$N_2(t-1) . N_3(t-1)$
$N_4(t) :\equiv : \sim N_1(t-2) . N_2(t-2) \vee N_3(t-2) . \vee . N_1(t-2) .$
$N_2(t-2) . N_3(t-2)$;
(g) $N_3(t) .\equiv . N_2(t-2) . \sim N_1(t-3)$;
(h) $N_2(t) .\equiv . N_1(t-1) . N_1(t-2)$;
(i) $N_3(t) :\equiv : N_2(t-1) . \vee . N_1(t-1) . (Ex) t-1 . N_1(x) . N_2(x)$.

Figura 5 – Imagem extraída da página 107 do artigo original de (MCCULLOCH; PITTS, 1943)

A imagem 5 foi extraída do artigo original de McCuloch and Pitts e demonstra as variações possíveis da estrutura criada por ambos para a representação do conhecimento num modelo matemático inspirado pela neurociência.

2.2.1.2 Marvin Minsky

Além do trabalho de McCuloch and Pitts,dois alunos de Harvard, Marvin Minsky e Dean Edmonds, construíram o primeiro computador de rede neural em 1950. O SNARC, como foi chamado, usava 3.000 válvulas eletrônicas e um mecanismo de piloto automático retirado de um bombardeiro B-24 para simular uma rede de 40 neurônios. Mais tarde, em Princeton, Minsky estudou computação universal em redes neurais. A banca examinadora de seu doutorado mostrou-se cética sobre esse tipo de trabalho, sem saber se deveria ser classificado como um trabalho de matemática. Porém, segundo contam, von Neumann teria dito: “Se não é agora, será algum dia.” Mais tarde, Minsky acabou provando teoremas importantes que mostravam as limitações da pesquisa em redes neurais.(RUSSEL, 2013)

2.2.1.3 Crescimento das expectativas em torno da Inteligência Artificial(1952-1969)

Os primeiros anos da IA foram repletos de sucessos, mas de uma forma limitada. Considerando-se os primitivos computadores, as ferramentas de programação da época e o fato de que apenas alguns anos antes os computadores eram vistos como objetos capazes de efetuar operações aritméticas e nada mais, causava surpresa o fato de um computador realizar qualquer atividade remotamente inteligente. Em geral, a classe intelectual preferia acreditar que “uma máquina nunca poderá realizar X”. Os pesquisadores da IA respondiam naturalmente demonstrando um X após outro. John McCarthy se referiu a esse período como a era do “Olhe, mamãe, sem as mãos!”.(RUSSEL, 2013)

2.2.1.4 Yan LeCun e a origem da CNN

Em 1989, um jovem cientista francês Yann LeCun aplicou um algoritmo de aprendizado no estilo backprop à arquitetura de rede neural convolucional de Fukushima. Depois de trabalhar no projeto por alguns anos, a LeCun lançou o LeNet-5(Lecun et al., 1998) - a primeira convnet moderna que introduziu alguns dos ingredientes essenciais que ainda hoje usamos nas CNNs.

Como Fukushima antes dele, LeCun decidiu aplicar sua invenção ao reconhecimento de caracteres e até lançou um produto comercial para leitura de códigos postais.

Além disso, seu trabalho resultou na criação do conjunto de dados MNIST de dígitos manuscritos - talvez o conjunto de dados de referência mais famoso no aprendizado de máquina.

2.2.1.5 Popularização do Backpropagation

Embora não tenha sido o criador do conceiro de BackPropagation, Geoffrey Hinton foi um dos responsáveis por popularizar o conceito.

A terceira abordagem é tentar desenvolver um procedimento de aprendizado capaz de aprender uma representação interna adequada para a execução da tarefa em questão. Um desses desenvolvimentos é apresentado na discussão das máquinas Boltzmann no capítulo 7. Como vimos, esse procedimento envolve o uso de unidades estocásticas, requer que a rede atinja o equilíbrio em duas fases diferentes e é limitada a trabalhos líquidos simétricos. Outra abordagem recente, também empregando unidades estocásticas, foi desenvolvida por Barlo (1985) e vários de seus colegas (cf. Barto Anandan, 1985).(RUMELHART et al., 1986)

Embora nossos resultados de aprendizado não garantam que possamos encontrar uma solução para todos os problemas solucionáveis, nossas análises e resultados mostraram que, na prática, o esquema de propagação de erros leva a soluções em praticamente todos os casos. Em suma, acreditamos que respondemos ao desafio de Minsky e Papert e descobrimos um resultado de aprendizado suficientemente poderoso para demonstrar que seu pessimismo em aprender em máquinas multicamadas foi extraviado.(RUMELHART et al., 1986)

2.2.1.6 O Algoritmo Viola-Jones

Em 2001, a primeira estrutura de detecção de rosto que funcionava em tempo real foi introduzida por Paul Viola e Michael Jones. Embora não seja baseado em aprendizado profundo(Deep Learning), o algoritmo ainda possui características semelhantes aos de um algoritmo de aprendizado profundo, pois ao processar imagens, ele é capaz de aprender quais recursos (embora sejam recursos simples do tipo Haar) podem ajudar a localizar um rosto.(DEMUSH, 2019)

A imagem 6 demonstra o processo como o algoritmo Viola-Jones(implementado em um FPGA) funcionava em analisar rostos humanos pelo padrão de formação de um rosto,

detectando o padrão do formato dos olhos, do nariz e da boca mediante seus padrões geométricos.

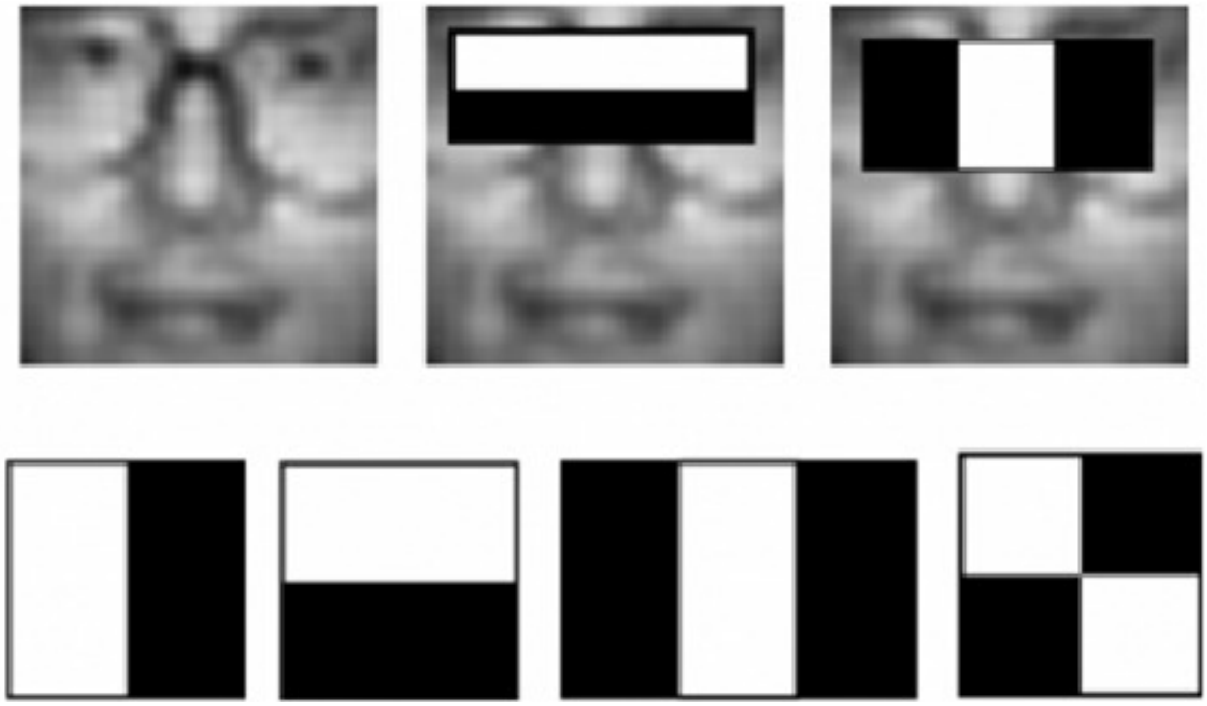


Figura 6 – Fonte:(NGO et al., 2009)

O detector de rosto Viola-Jones ainda é amplamente utilizado. É um classificador binário forte, construído com vários classificadores fracos.(DEMUSH, 2019)

2.2.1.7 Lei de Moore

A previsão de Gordon Moore de 1965 de que o número de componentes em um circuito integrado dobraria a cada ano até atingir 65.000 em 1975, é a maior previsão tecnológica do último meio século. Quando se provou correto em 1975, ele revisou o que ficou conhecido como Lei de Moore para uma duplicação de transistores em um chip a cada dois anos.

Desde então, sua previsão definiu a trajetória da tecnologia e, de várias maneiras, do próprio progresso.

O argumento de Moore foi econômico. Os circuitos integrados, com vários transistores e outros dispositivos eletrônicos interconectados com linhas de metal de alumínio em um pe-

queno quadrado de pastilha de silício, haviam sido inventados alguns anos antes por Robert Noyce, da Fairchild Semiconductor. Moore, o diretor de P&D da empresa, percebeu, como escreveu em 1965, que, com esses novos circuitos integrados, "o custo por componente é quase inversamente proporcional ao número de componentes". Foi uma bela pechincha - em teoria, quanto mais transistores você adicionou, mais barato cada um ficou. Moore também viu que havia muito espaço para avanços na engenharia para aumentar o número de transistores que você poderia colocar de forma econômica e confiável em um chip.(WE'RE. . . ,)

2.2.1.8 Alexnet

Em 2012, Alex Krizhevsky treinou 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 do teste, foi alcançado taxas de erro entre 1 e 5 entre 37,5% e 17,0%, o que é consideravelmente melhor do que o estado da arte anterior. A rede neuronal, que possui 60 milhões de parâmetros e 650.000 neurônios, consiste em cinco camadas convolucionais, algumas das quais são seguidas por camadas de pool máximo e três camadas totalmente conectadas com um softmax final de 1000 vias. Para acelerar o treinamento, Alex usou neurônios não saturantes e uma implementação muito eficiente da GPU da operação de convolução. Para reduzir o excesso de ajustes nas camadas totalmente conectadas, empregamos um método de regularização recentemente desenvolvido chamado "desistência"que provou ser muito eficaz. Também entrou em uma variante deste modelo na competição doILSVRC-2012,alcançando uma taxa de erro de teste entre os 5 melhores em 15,3%, em comparação com 26,2% alcançados pela segunda melhor entrada.(KRIZHEVSKY; SUTSKEVER; HINTON, 2012)

2.2.2 Carros autônomos

2.2.2.1 Darpa Grand Challenge

Em 2004,a Agência de Projetos de Pesquisa Avançada de Defesa(Defense Advanced Research Projects Agency) do Departamento de Defesa dos Estados Unidos iniciou a competição DARPA Grand Challenge, no objetivo de promover o desenvolvimento de veículos autônomos pelos pesquisadores americanos.

A primeira competição do DARPA Grand Challenge foi realizada em 13 de março de 2004 na região do deserto de Mojave, nos Estados Unidos, num percurso de 240 quilômetros. Nenhum veículo robótico que participou terminou a rota. A equipe vermelha da Universidade Carnegie Mellon e o carro Sandstorm (um Humvee convertido) percorreram a maior distância, completando 11,78 km (7,32 milhas) do percurso antes de serem pendurados em uma rocha

depois de fazer um retorno. Nenhum vencedor foi declarado e o prêmio em dinheiro não foi entregue, fato este que levou a ter um segundo evento em 2005.(DARPA,).

2.2.2.2 Google Street View

Em 2007, Google lança um projeto visando mapear as ruas de todo o mundo. Durante o projeto, testes com carros autônomos foram feitos em perímetros controlados, com o objetivo de obter os dados das ruas, além de calibrar os modelos de carros autônomos desenvolvidos pela empresa.

2.2.2.3 Waymo

O projeto Waymo começou em 2009, sendo inicialmente liderado pelo empresário e cientista alemão Sebastian Thrun , ex-diretor do Laboratório de Inteligência Artificial de Stanford e coinventor do Google Street View . A equipe de Thrun em Stanford criou o veículo robótico Stanley que ganhou o DARPA Grand Challenge 2005 além do prêmio de US$ 2 milhões do Departamento de Defesa dos Estados Unidos . A equipe de desenvolvimento do sistema consistia de 15 engenheiros que trabalham para o Google, incluindo Chris Urmson , Mike Montemerlo, e Anthony Levandowski que havia trabalhado na DARPA desafios urbanos .

Em 28 de março de 2012, o Google postou um vídeo no YouTube mostrando Steve Mahan, morador da cidade de Morgan Hill, Califórnia, em um passeio no veículo autônomo do Google Toyota Prius num percurso de aproximadamente 2.4 quilômetros. Steve Mahan perdeu 95% de sua visão e portanto, foi considerado legalmente cedo.(VOELCKER, 2012)

No final de maio de 2014, o Google revelou um novo protótipo de seu carro sem motorista, que não tinha volante, pedal de acelerador ou pedal de freio, sendo 100% autônomo. e apresentou um protótipo totalmente funcional em dezembro do mesmo ano. Em 2015, Steve Mahan(o motorista cedo do parágrafo anterior), fez o primeiro passeio de autocondução em vias públicas, em Austin, Texas. Além disso, em abril de 2014, a equipe anunciou que seus veículos já registraram quase 700 mil milhas autônomas (1,1 milhão de km).

Em Dezembro de 2016, Waymo chegou a testar sua frota no total de 3,2 milhões de quilômetros percorridos.(ESTADAO, 2016)

Em novembro de 2017 a empresa publicou dados sobre os testes sendo realizados para aprimorar sua tecnologia para carros autônomos. Entre os dados divulgados, a empresa afirmou que seus carros autônomos já rodaram mais de 4 milhões de milhas (aproximadamente 6,5 milhões de quilômetros) em estradas públicas localizadas em 23 cidades nos Estados Unidos.(TEAM, 2017)

2.2.2.4 Acidente do Uber

A imagem 7 mostra o caso que ocorreu em Março de 2018, quando um veículo autônomo da Uber atropelou e matou uma ciclista enquanto transitava em um perímetro urbano no período da noite, levantando uma preocupação no que diz respeito aos veículos autônomos e sua capacidade de serem completamente livres da direção de um ser humano em seu controle.

Figura 7 – Imagem mostrando o veículo que atropelou e matou a pedreste Elaine Herzberg enquanto empurrava uma bicicleta no meio da noite (SELF-DRIVING. . . ,)

2.2.3 Ferramentas computacionais

O número de algoritmos ML, bem como suas diferentes implementações de software, é bastante grande. Muitas ferramentas de software para técnicas DL de ML estão em desenvolvimento nos últimos 25 anos (JOVIC; BRKIć; BOGUNOVIć, 2014). Seu objetivo comum é facilitar o complicado processo de análise de dados e propor ambientes integrados sobre as linguagens de programação padrão. Essas ferramentas são projetadas para diversos fins: como plataformas de análise, sistemas preditivos, sistemas de recomendação, processadores (para imagens, som ou linguagem). Vários deles são orientados ao processamento e streaming rápidos de dados em grande escala, enquanto outros são especializados na implementação de algoritmos de ML, incluindo NNs e DL. Novamente, é importante enfatizar que não existe uma ferramenta única adequada para todos os problemas e, muitas vezes, é necessária uma combinação deles para ter sucesso. A Figura 8 fornece uma visão geral agrupada e abrangente de estruturas e bibliotecas de ML.(NGUYEN et al., 2019)

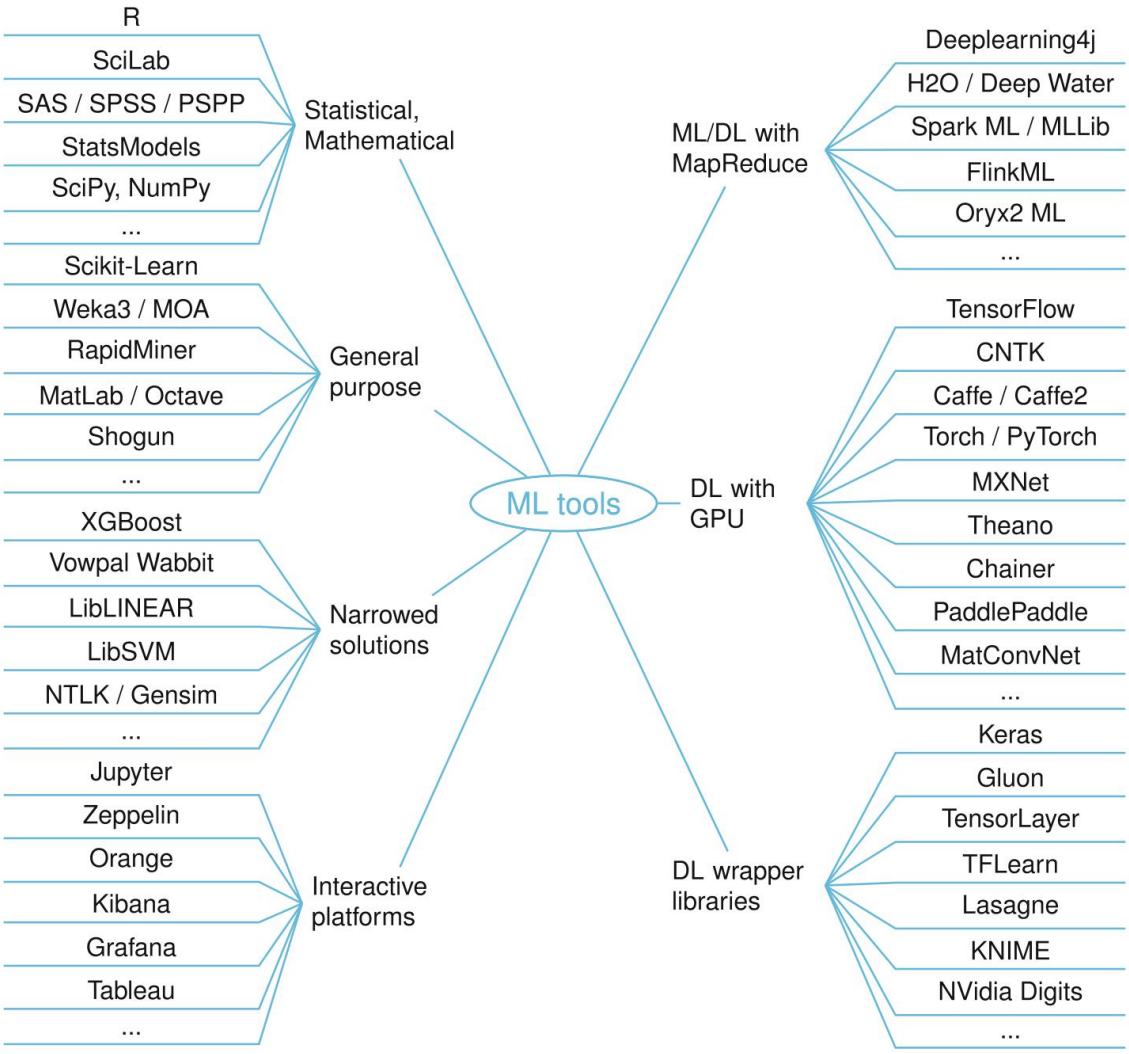


Figura 8 – Imagem mostrando o conjunto de ferramentas computacionais de Machine Learning e Deep Learning utilizadas hoje pela comunidade internacional acadêmica e industrial. Fonte:(NGUYEN et al., 2019)

3 Métodos e materiais

Como mencionado anteriormente, este trabalho objetiva construir um classificador de placas de trânsito que seja capaz de executar em um NVIDIA Jetson Nano, a fim de ajudar na redução de custo do maquinário necessário para a construção de um carro autônomo. No que diz respeito à construção e desenvolvimento desta pesquisa, de acordo com (GERHARDT; SILVEIRA, 2009), sua abordagem pode ser considerada como quantitativa pois seus dados podem ser quantificados através de uma linguagem matemática. No que diz respeito à natureza da pesquisa, a mesma pode ser qualificada como pequisa aplicada devido ao objetivo de se gerar conhecimento aplicado tanto para a indústria quanto para a academia.

Os métodos foram organizados da seguinte maneira:

1. Coletar, reunir e organizar conhecimentos à respeito de Machine Learning, Deep Learning e Visão computacional
2. Criar a base de dados utilizando o Google Street view
3. Configurar o ambiente no hardware de treinamento(Google Colab)
4. Desenvolver o algoritmo de treinamento
5. Treinar o modelo no hardware de treinamento utilizando a base de dados obtida previamente
6. Exportar o modelo treinado no hardware de treinamento
7. Configurar o ambiente no NVIDIA Jetson Nano para executar o algoritmo principal
8. Coletar resultados do modelo obtido a fim validar e verificar sua precisão e desempenho
9. Comparar os resultados obtidos para diferentes modelos

O processo acima é o mesmo a ser utilizado para cada modelo a ser treinado/validado. Dessa forma, o diagrama da figura 9 abaixo complementa o processo previamente descrito.

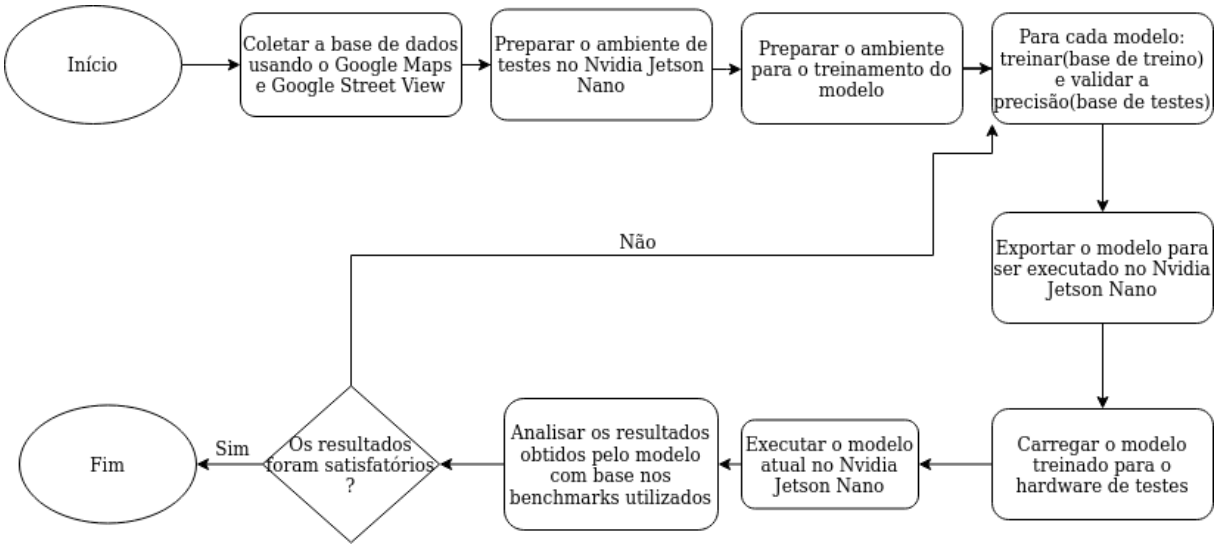


Figura 9 – Representação visual da metodologia empregada para o desenvolvimento do trabalho. Fonte: o autor

Os materiais utilizados no trabalho foram:

- Laptop Lenovo G50-80 para estudo pessoal do autor, usado para coletar e "etiquetar"a base de dados, não sendo utilizado para a execução do modelo de IA proposto;
- Instância no Google Colab para a execução do algoritmo, modelos com o Tensorflow e treinamento do algoritmo;
- NVIDIA Jetson Nano.

Especificações no notebook pessoal do autor:

- Ubuntu 18.04 LTS
- Processador Intel Core I5
- Memória RAM de 12 GB
- SSD Samsung EVO 512GB

Especificações do NVIDIA Jetson Nano utilizado no trabalho:

- GPU: 128-core Maxwell^{TM} GPU.
- CPU: quad-core ARM® Cortex®-A57 CPU.
- Memoria: 4GB 64-bit LPDDR4.
- Armazenamnto: Micro SD 16GB
- Video: Encode: 4K @ 30 (H.264/H.265)
- Interfaces: Ethernet: 10/100/1000BASE-T auto-negotiation.
- Power: Micro USB (5V 2A)

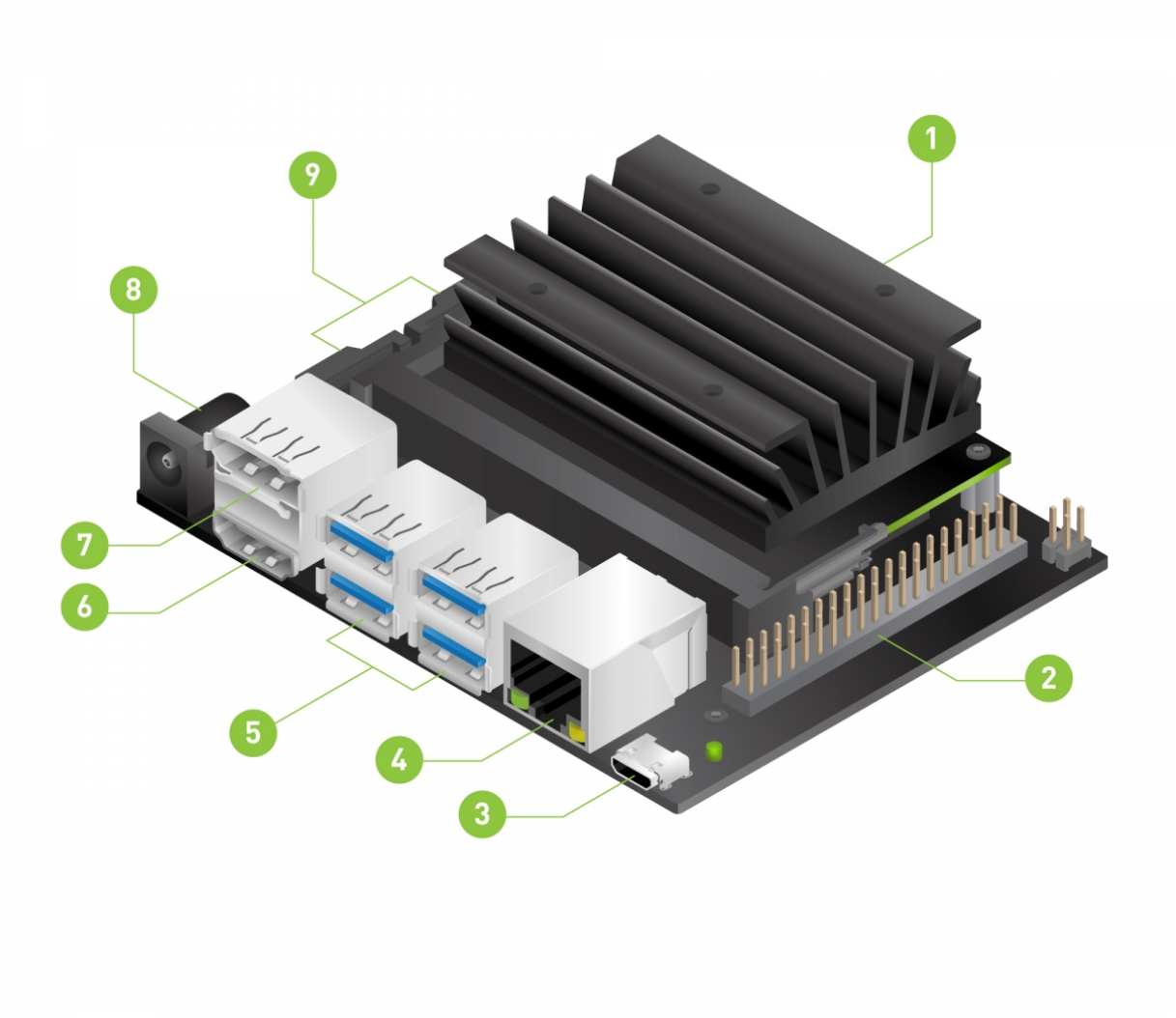


Figura 10 – Fonte (NVIDIA, 2020a)

1. Entrada de cartão MicroSD para armazenamento principal
2. cabeçote de expansão com 40 pinos
3. Porta Micro-USB de 5V para alimentação típico de celular
4. Porta Gigabit Ethernet
5. 4 Portas USB 3.0
6. Saída HDMI
7. Conector DisplayPort
8. Entrada DC 5V para alimentação
9. Conectores de câmera MIPI CSI-2

3.1 Criação e catalogação da base de dados

Das etapas do fluxo da figura 9 mostrado anteriormente, vale a pena destacar a etapa de construção da base de dados, pois sem a base de dados, não é possível construir o modelo proposto pelo trabalho. Para construir a base de dados mencionada na segunda etapa (contando com o "balão"de Início), foi necessária horas de análise de fotos do Google Maps e Google Street View, percorrendo pelas ruas brasileiras através da ferramente e tirando "print"da tela de forma com que os detalhes da rua ficassem visíveis. Após a coleta de um número suficientemente grande de imagens, é necessário um mapeamento manual, demarcando em cada imagem, a posição de cada placa de trânsito a ser identificada pelo modelo a ser construído. As placas escolhidas para serem reconhecidas e treinadas pelo modelo foram:

- Proibido estacionar
- Permitido estacionar
- Proibido parar e estacionar
- Pare
- Proibido virar à esquerda
- Proibido virar à direita

As placas acima foram escolhidas pela maior quantidade encontrada durante a coleta em contrapartida com placas como "Curva sinuosa", que não são frequentes em ambientes urbanos como foi o caso do ambiente de coleta.

3.2 Escolha dos modelos de Deep Learning

Para o trabalho em questão, foi escolhido dos modelos de Deep Learning disponíveis no Model Zoo(NADA, 2021), que é uma coleção de modelos de CNN pré-treinados utilizando o Tensorflow pela comunidade de cientistas da computação. O Model Zoo permite a técnica de Transfer learning, que consiste em repassar o conhecimento adquirido em um treinamento, para outro, fazendo com que o engenheiro ganhe tempo no processo de treinamento. Os modelos selecionados foram uma SSD MobileNet(YOUNIS et al., 2020) e uma SSD Resnet (Lu et al., 2019)

Os modelos acima foram escolhidos devido ao fato de trabalharem com GPUs e terem recebido modificações arquiteturais em prol de um maior desempenho do modelo no que diz respeito à sua execução.

4 Implementando a solucao de Deep Learning para a detecção das placas de trânsito

4.1 Coleta e tratamento da base de dados

Para desenvolver a base de dados, em suma foram necessárias as seguintes etapas:

- Ir ao Google Maps e tirar prints de tela com as imagens do trânsito brasileiro que exibiam ao menos uma placa de trânsito
- Para cada imagem coletada, foi utilizado um programa chamada labelImg para gerar um arquivo XML manualmente contendo:
    1. Nome do arquivo de imagem no computador (ex: img1.png)
    2. Caminho do arquivo no computador
    3. Tamanho da imagem em pixels(largura e altura)
    4. Uma lista contendo os pontos x1,y1(pontos iniciais) e x2,y2(pontos finais) contendo as coordenadas do objeto na imagem como um plano cartesiano e, o nome da classe dado
- Separar os arquivos entre treinamento e teste do modelo

O processo de mapeamento de uma imagem utilizando o labelImg funciona como na imagem 11

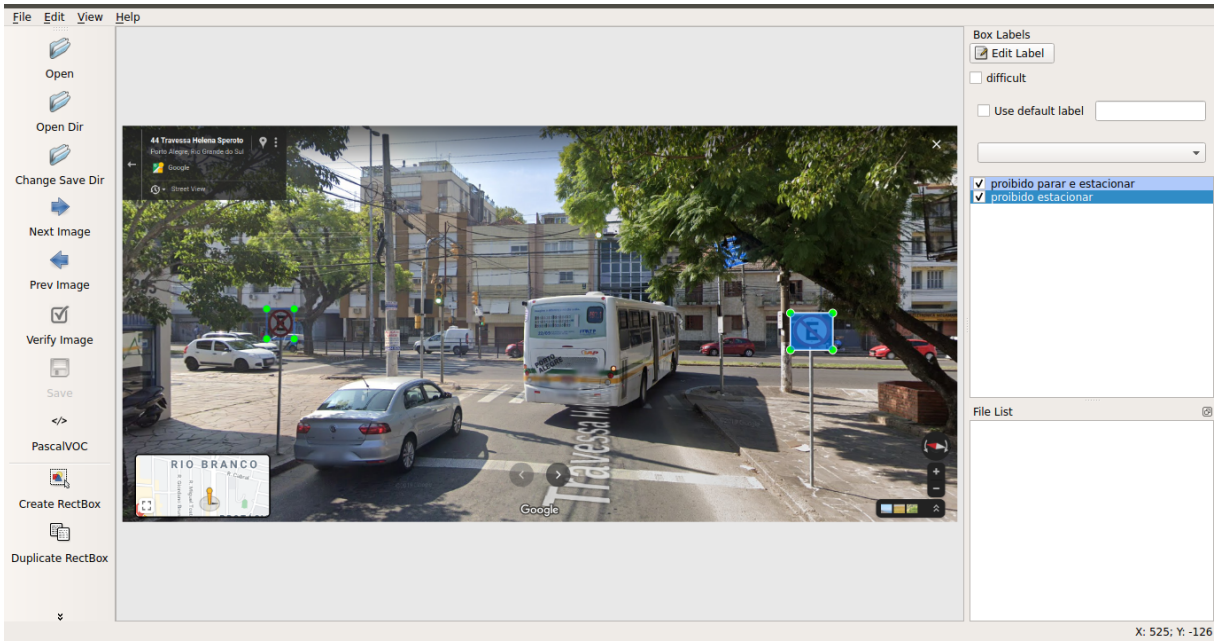


Figura 11 – Imagem mostrando o funcionamento do software de mapeamento labelImg. Fonte: O autor

4.1.1 Escolha das classes a serem detectadas

Para se obter um classificador de alta precisão, é preciso ter uma base de dados razoavelmente grande contendo cada classe a ser detectada. Devido ao tempo necessário para a coleta de dados ser grande, o mesmo inviabiliza obter uma base de dados com amostras suficientemente grandes de cada placa de trânsito(classe) a ser detectada. Dessa forma, foi reduzido para 5 placas de trânsito a serem reconhecidas pelo modelo.

A escolha das classes de trânsito se deu da seguinte forma:

- Mapeamento com o labelImg de todas as classes encontradas nas 125 imagens obtidas na primeira coleta.
- Desenvolvimento de um script feito em Python, que foi capaz de ler todos os arquivos .xml gerados pelo software labelImg, contando a quantidade em que cada placa foi encontrada

Após a execução do script que contabilizava a quantidade de placas de trânsito, o algoritmo detectou:

Tabela 1 – Quantidade de classes encontradas na primeira amostragem de 125 imagens. Fonte: O autor

Classe	Quantidade
permitido estacionar	73
proibido estacionar	32
proibido parar e estacionar	31
pare	21
proibido virar à direita	9
proibido virar a esquerda	8
vel max de 60km/h	5
siga em frente ou a esquerda	4
vel max de 40km/h	3
proibido retornar a esquerda	3
proibido seguir em frente	3
sentido circular na rotatória	3
vire à direita	2
siga em frente ou a direita	2
vel max de 30km/h	2
siga em frente	2
duplo sentido de circulação	1

Portanto, de acordo com os dados obtidos pertinentes à quantidade de cada classe encontrada nas imagens, foi escolhido as cinco classes mais populares, que foram:

- permitido estacionar
- proibido estacionar
- proibido parar e estacionar
- pare
- proibido virar à direita

Após ter definido as imagens, foi-se necessário remover do conjunto inicial, as amostras que não continham nenhuma das 5 classes selecionadas. Depois disso, foi feita uma nova coleta da base de dados, totalizando 500 imagens no total, contendo amostras das placas de trânsito escolhidas

Após a nova coleta de dados, foi feita o mapeamento dos arquivos .xml dessas novas imagens, além da divisão total da base entre treino e teste, totalizando 125 imagens para teste e, 375 para treino(considerando apenas os arquivos .png), numa proporção de 25% das imagens para testes(validação) e 75% das imagens para treinamento.

É importante mencionar que o mapeamento de todas as classes(placas de trânsito) nas 500 imagens finais foi feito de forma manual, sendo necessário analisar, imagem por imagem, a fim de se encontrar as placas de trânsito e fazer as marcações a serem utilizadas pelos algoritmos.

4.2 Seleção dos modelos

O primeiro modelo escolhido foi a SSD Resnet-50 v1 FPN, redimensionando as imagens para um tamanho de 640x640 pixels. O modelo SSD Resnet foi escolhido por utilizar o conceito de SSD(single-shot detector ou detector em um único disparo), que teve,em seu artigo original, um tempo de inferência superior ao YOLO, como utilizado pelo (CAMPOS, 2019) e mais preciso que arquiteturas como a Faster R-CNN, que é uma outra arquitetura que evoluiu da CNN tradicional. Além disso, um modelo que trabalha como SSD é de grande relevância para o problema dos carros autônomos visto que, o modelo de Deep Learning deve ser rápido o suficiente para fazer as inferências no trânsito

4.2.1 SSD Resnet-50 FPN(RetinaNet)

O primeiro modelo escolhido é chamado de SSD Resnet-50 FPN ou RetinaNet e foi desenvolvido pela equipe de pesquisa em Deep Learning do Facebook, sendo considerado um modelo de apenas 1 estágio, diferente de vários outros modelos que trabalham em dois estágios. Modelo de 1 estágio é quando a detecção das possíveis "bounding boxes"e a classificação da possível "bounding box"é feito em apenas uma etapa, ao invés de duas. Modelos de um estágio são conhecidos por terem problemas quando existem um desbalanceamento de classes na base de dados e gerarem muitos falsos positivos com uma quantia muito alta de possíveis "bounding boxes"a serem classificados pelo CNN como na imagem 13, que neste caso, é derivado da arquitetura Resnet. Além disso, o arquivo indicando a estrutura do modelo em questão utilizado (pipeline.config) está no apêndice A.1.

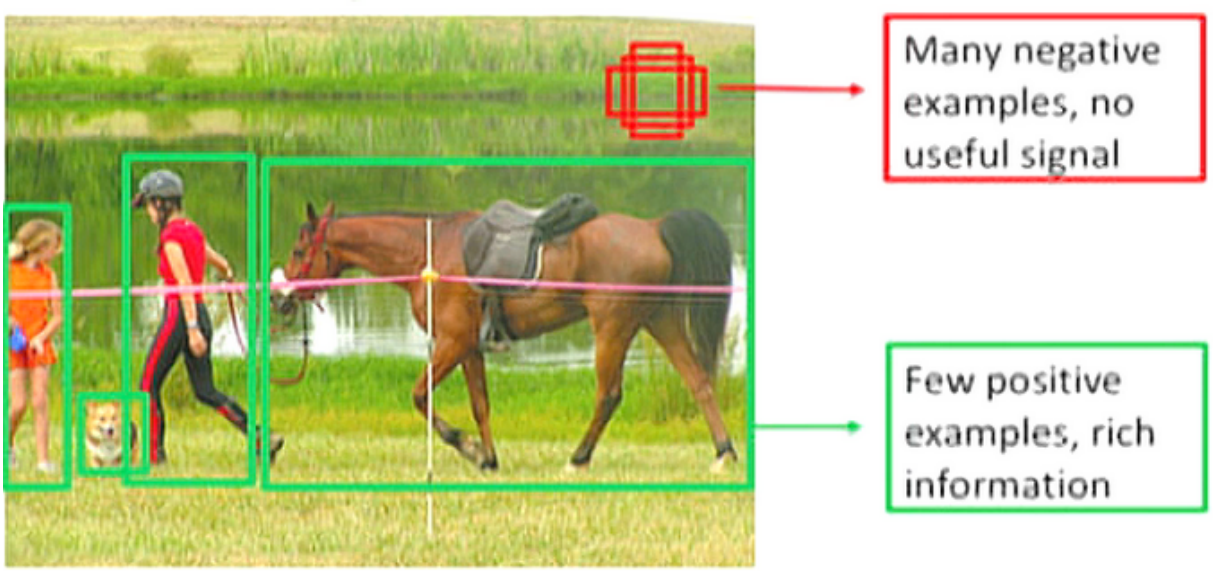


Figura 12 – Imagem demonstrando como a RetinaNet atua em um único estágio, gerando os candidatos bounding boxes a serem detectados e os possíveis falsos positivos.. Fonte: (NADA, 2019)

Sobre a arquitetura da RetinaNet (Lin et al., 2020) afirmam:

> RetinaNet é uma rede única, unificada e composta por uma rede principal backbone e duas sub-redes de tarefas específicas. A rede backbone é responsável por computar um mapa de características convolucionais sobre uma imagem de entrada inteira. A primeira sub-rede realiza uma classificação convolucional de objetos com a saída da rede backbone; a segunda sub-rede realiza a regressão da caixa delimitadora convolucional(bounding boxes). As duas sub-redes apresentam um design simples que propomos especificamente para detecção densa de um estágio. Embora existam muitas opções possíveis para os detalhes desses componentes, a maioria dos parâmetros de design não são particulares amplamente sensível aos valores exatos, conforme mostrado nos experimentos. Descrevemos cada componente do RetinaNet a seguir.

Recurso Backbone de rede em pirâmide(Feature Pyramid Network). Adotamos o recurso Rede Pirâmide como a rede de backbone para RetinaNet. Em resumo, FPN aumenta um padrão convolucional de rede com um caminho de cima para baixo e conexões laterais para que a rede construa de forma eficiente um recurso

> rico e em várias escalas pirâmide de estrutura a partir de uma imagem de entrada de resolução única. Cada nível da pirâmide pode ser usado para detectar objetos em uma escala diferente. FPN melhora dimensionar previsões de redes totalmente convolucionais (FCN), como mostrado por seus ganhos para RPN e DeepMask-style propostas, bem como em detectores de duas fases, como Fast R- CNN ou Máscara R-CNN.
>
> Construímos o FPN sob o topo da arquitetura ResNet.

(Lin et al., 2020) (tradução nossa).

O termo SSD (LIU et al., 2016)é um método que significa que a detecção é feita em um único disparo(Single Shot Detection).Além disso,a base deste modelo foi desenvolvida com uma Resnet além do FPN(explicada com mais detalhes na seção de Revisão Bibliográfica deste trabalho juntamente com o conceito de SSD) para extração profunda de características como demonstrada na figura 13 abaixo.

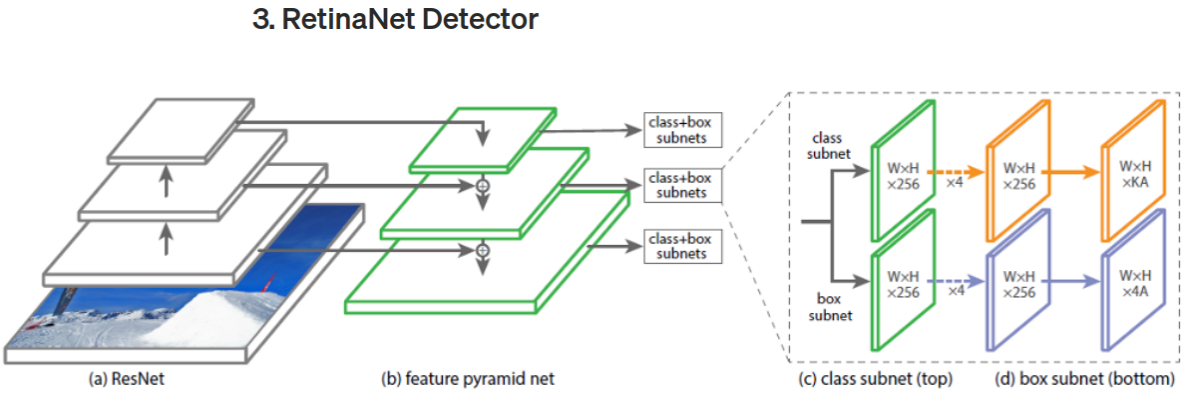


Figura 13 – Imagem exemplificando como funciona a arquitetura da RetinaNet. Fonte: (Lin et al., 2020)

A Retinanet teve o objetivo de bater, quando foi desenvolvido, o estado da arte atual e outros modelos de um único estágio como o Yolo utilizado por (CAMPOS, 2019) no que diz respeito ao grau de precisão do modelo, além de ter as vantagens dos modelos de um único estágio.

4.2.2 SSD Mobilenet(Mobile-Det)

Assim como mencionado na arquitetura do modelo anterior, este modelo também trabalha utilizando SSD, o que significa que a detecção dos candidatos(possíveis classes encontradas na imagem) e as "bounding boxes"e a classificação desses "bounding boxes"ocorre em um único estágio. A combinação da arquitetura Mobilenet juntamente com o processo SSD ficou conhecido como Mobile-Det e, sua arquitetura é demonstrada na figura 14 abaixo:

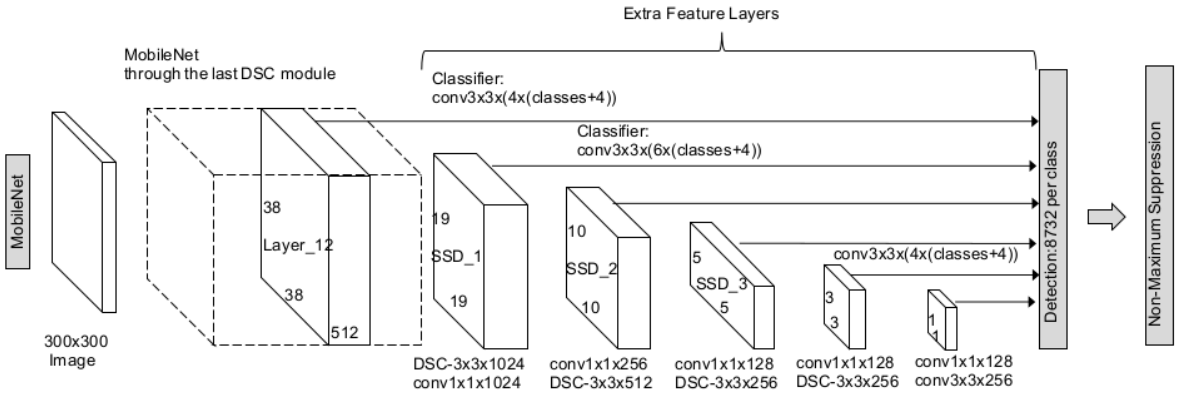


Figura 14 – Imagem demonstrando como funciona a arquitetura da Mobile-Det. Fonte: (ZHANG; PENG; HU, 2017)

Deve-se mencionar que, assim como o modelo anterior, o arquivo indicando a estrutura do modelo em questão utilizado (pipeline.config) está no apêndice A.2.

O principal motivo que levou à escolha deste modelo foi o seu sucesso ao funcionar com hardwares de baixo custo, tendo uma boa relação custo-desempenho, apresentando bons resultados mesmo com baixo recurso de hardware disponível (HOWARD et al., 2017). Tal fato pode ser verificado na imagem 15 abaixo, onde, num conjunto de modelos e o Mobile-Det por último, é analisado o tamanho em disco,velocidade de detecção e a mAP(mean average precision ou média da precisão média das classes obtidas).

Table 3. Comparison of different models (Note: the result of Mobile-Det is still preliminary)

Type of model	Size on disk	Detection speed	mAP
Yolo2-full	269.9MB	Out of memory	76.8
Yolo2-tiny	60.5MB	0.487fps	57.1
Yolo2-full eight-bit	64.4MB	0.153fps	61.3
Yolo2-tiny eight-bit	15.2MB	0.343fps	49.8
Temporal Detection	4.4MB	2.566fps	*
Mobile-Det	27.5MB	0.712fps	41.9

*Under-defined, please see section 5.4

Figura 15 – Comparativo da SSD Mobilenet(Mobile-Det) e outros modelos de redes neurais convolucionais. Fonte: (ZHANG; PENG; HU, 2017)

4.3 Desenvolvimento do algoritmo

O algoritmo, bem como os modelos de Deep Learning foram desenvolvidos utilizando o Google Colab como na figura 16, que é uma plataforma colaborativa do Google, que permite o acesso à hardwares com GPU. Dessa forma, foi criado uma instância no Google Colab utilizando uma GPU como acelerador gráfico.

Além disso, foi utilizada a API de detecção de objetos do Tensorflow, que permite a utilização dinâmicamente de diferentes modelos de detecção de objetos, como as selecionadas para este trabalho.

Foi configurado no Google Drive, uma pasta com a API do Tensorflow para ser utilizado no algoritmo, bem como os modelos pré-treinados obtidos pelo Model Zoo.

Foi configurado o arquivo pipeline para ambos os modelos escolhidos. Um arquivo pipeline é utilizado para definir o processo de treinamento do modelo, bem como o tamanho da imagem a ser treinada (em pixels), além da quantidade épocas de treinamento e a taxa de aprendizagem da primeira iteração.

Para a execução do modelo criado no Jetson Nano, foi necessário configurar o ambiente de execução no mesmo, assim como o Tensorflow e todas as dependências Python.

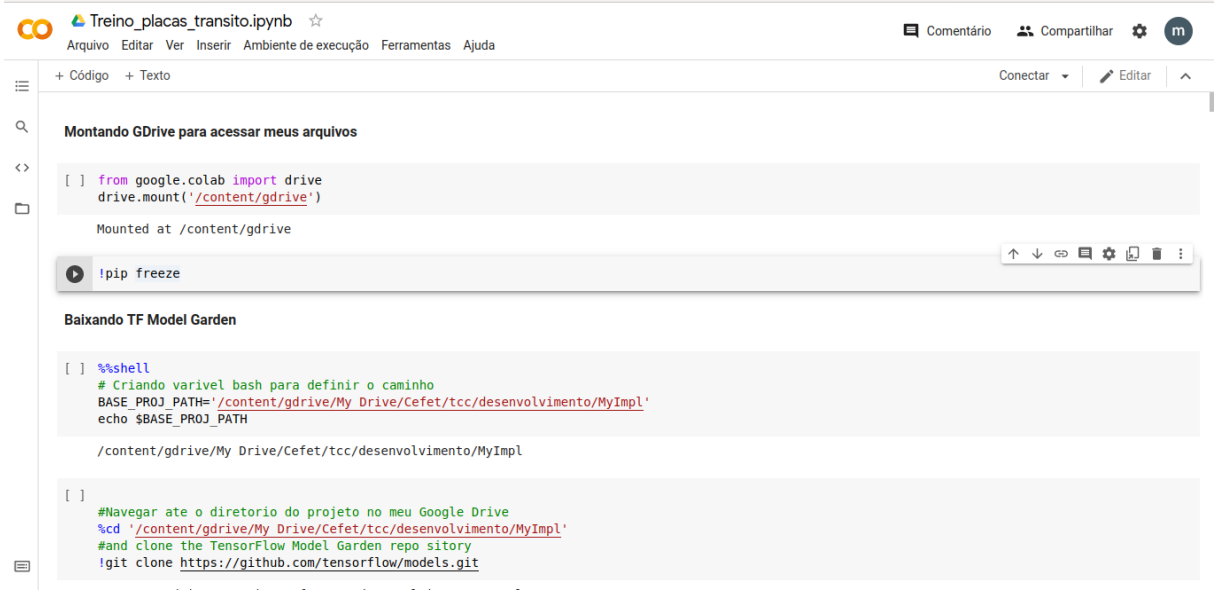


Figura 16 – Imagem mostrando o ambiente utilizado para desenvolver o algoritmo de treinamento com o Google Colab. Fonte: O autor

O processo desenvolvido para treinar o algoritmo foi:

- Integrar o Google Colab com o Google Drive, para ter acesso à base de dados, modelos pré-treinados e a API do TensorFlow
- Clonar a API de detecção de objetos do Tensorflow hospedado no Github
- Instalar as dependências Python para realizar o desenvolvimento do algoritmo
- Compilar os modelos baixo nível que são integrados ao ambiente Python, mas desenvolvidos com C++ e CUDA que usam o Protobuf compiler.
- Definir o caminho no sistema para facilitar a integração de todos os recursos existentes na API do Tensorflow e desenvolvidos pela comunidade
- Compilar e construir as dependêcias remanescentes após ajustar o caminho de recursos da API do Tensorflow
- Gerar os arquivos .record de treinamento e de teste a ser utilizados pelo Tensorflow
- Carregar o Tensorboard, para visualizar a curva de aprendizagem e erros dos modelos a serem treinados

- Configurar a quantidade de GPUs a serem utilizadas do hardware atual
- Executar o treinamento(uma execução para cada modelo escolhido)
- Exportar o modelo treinado para ser utilizado em outra máquina
- Baixar o modelo exportado devidamente gerado no treinamento
- Transferir o modelo treinado para o Jetson Nano previamente configurado utilizando o protocolo SCP.
- Executar o modelo gerado no Google Colab na base de teste e gerar os bounding-boxes nas placas de trânsito encontradas
- Analizar a taxa de acerto de cada modelo executado no Jetson Nano, bem como o tempo médio de inferência de cada modelo.

O processo acima pode ser analisado de forma mais simplificada no diagrama da figura 17:

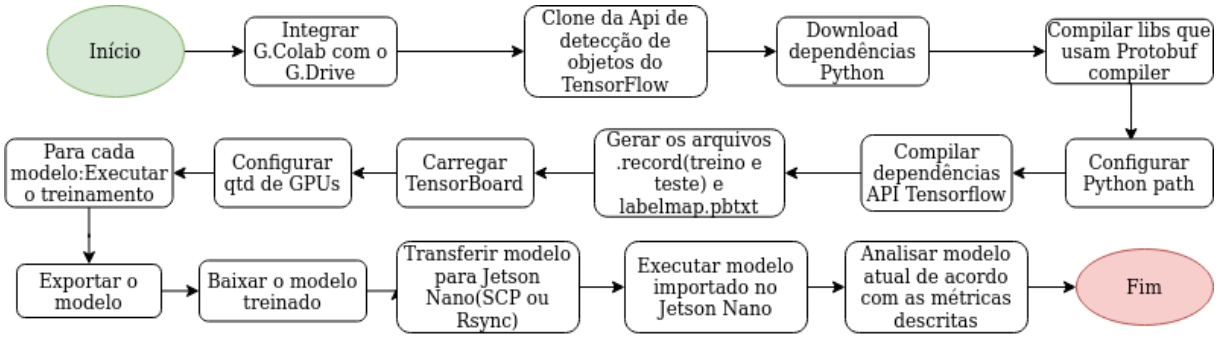


Figura 17 – Descrição em forma de fluxograma do passo a passo para se desenvolver o algoritmo de treinamento dos modelos. Fonte: O Autor

As imagens 18 e 19 abaixo demonstram o aprendizado do segundo modelo ao longo do tempo, no qual a imagem 18 mostra, através da ferramenta TensorBoard, a taxa de erro do modelo ao longo do tempo e o seu decaimento ao longo das gerações de execução do modelo. Enquanto isso, a imagem 19 mostra o mesmo que a imagem 18, porém num formato de saídas diretas geradas pelo modelo num formato de logs, o que é comum em sistemas computacionais a fim de se visualizar o que o sistema está produzindo e o seu estado atual de máquina.

É importante mencionar que, ambos os modelos escolhidos foram treinados(cada um) num período de aproximadamente de 4 a 5 horas usando a mesma base de dados para treino e validação.

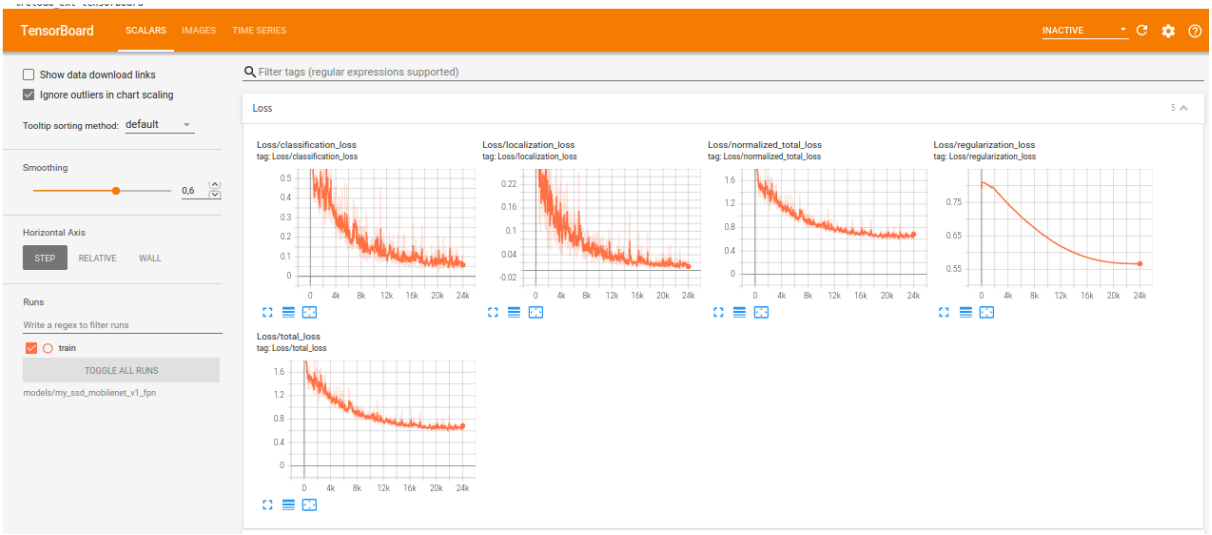


Figura 18 – Imagem mostrando a taxa de erros do modelo Mobilenet ao longo do treinamento do mesmo. Fonte: O autor

```
# treinando o modelo
# mobilenet
!python model_main_tf2.py --model_dir=models/my_ssd_mobilenet_v1_fpn --pipeline_config_path=models/my_ssd_mobilenet_v1_fpn/pipeline.config
# resnet
# !python model_main_tf2.py --model_dir=models/my_ssd_resnet50_v1_fpn --pipeline_config_path=models/my_ssd_resnet50_v1_fpn/pipeline.config
# !python model_main_tf2.py --model_dir=models/[name_of_pre-trained-model_you_downloaded] --pipeline_config_path=models/[name_of_pre-trained-model_you_downloaded]/pipeline.config
I0320 01:55:21.259084 139759569127296 model_lib_v2.py:651] Step 6100 per-step time 0.788s loss=0.987
2021-03-20 01:55:34.513654: W tensorflow/core/lib/png/png_io.cc:88] PNG warning: iCCP: known incorrect sRGB profile
2021-03-20 01:56:22.592002: W tensorflow/core/lib/png/png_io.cc:88] PNG warning: iCCP: known incorrect sRGB profile
2021-03-20 01:56:30.572612: W tensorflow/core/lib/png/png_io.cc:88] PNG warning: iCCP: known incorrect sRGB profile
INFO:tensorflow:Step 6200 per-step time 0.784s loss=0.992
I0320 01:56:40.175546 139759569127296 model_lib_v2.py:651] Step 6200 per-step time 0.784s loss=0.992
2021-03-20 01:57:04.087902: W tensorflow/core/lib/png/png_io.cc:88] PNG warning: iCCP: known incorrect sRGB profile
2021-03-20 01:57:10.277596: W tensorflow/core/lib/png/png_io.cc:88] PNG warning: iCCP: known incorrect sRGB profile
INFO:tensorflow:Step 6300 per-step time 0.779s loss=0.876
I0320 01:57:59.170687 139759569127296 model_lib_v2.py:651] Step 6300 per-step time 0.779s loss=0.876
INFO:tensorflow:Step 6400 per-step time 0.792s loss=0.987
I0320 01:59:17.670767 139759569127296 model_lib_v2.py:651] Step 6400 per-step time 0.792s loss=0.987
2021-03-20 01:59:43.133086: W tensorflow/core/lib/png/png_io.cc:88] PNG warning: iCCP: known incorrect sRGB profile
INFO:tensorflow:Step 6500 per-step time 0.827s loss=0.966
I0320 02:00:37.273338 139759569127296 model_lib_v2.py:651] Step 6500 per-step time 0.827s loss=0.966
2021-03-20 02:00:43.922031: W tensorflow/core/lib/png/png_io.cc:88] PNG warning: iCCP: known incorrect sRGB profile
INFO:tensorflow:Step 6600 per-step time 0.773s loss=0.913
I0320 02:01:56.610817 139759569127296 model_lib_v2.py:651] Step 6600 per-step time 0.773s loss=0.913
2021-03-20 02:02:43.403498: W tensorflow/core/lib/png/png_io.cc:88] PNG warning: iCCP: known incorrect sRGB profile
INFO:tensorflow:Step 6700 per-step time 0.780s loss=0.944
I0320 02:03:15.864028 139759569127296 model_lib_v2.py:651] Step 6700 per-step time 0.780s loss=0.944
2021-03-20 02:03:45.063248: W tensorflow/core/lib/png/png_io.cc:88] PNG warning: iCCP: known incorrect sRGB profile
2021-03-20 02:04:09.872053: W tensorflow/core/lib/png/png_io.cc:88] PNG warning: iCCP: known incorrect sRGB profile
INFO:tensorflow:Step 6800 per-step time 0.811s loss=1.088
I0320 02:04:33.955677 139759569127296 model_lib_v2.py:651] Step 6800 per-step time 0.811s loss=1.088
2021-03-20 02:04:35.748608: W tensorflow/core/lib/png/png_io.cc:88] PNG warning: iCCP: known incorrect sRGB profile
INFO:tensorflow:Step 6900 per-step time 0.875s loss=0.922
I0320 02:05:53.169466 139759569127296 model_lib_v2.py:651] Step 6900 per-step time 0.875s loss=0.922
2021-03-20 02:06:58.189259: W tensorflow/core/lib/png/png_io.cc:88] PNG warning: iCCP: known incorrect sRGB profile
INFO:tensorflow:Step 7000 per-step time 0.769s loss=1.267
I0320 02:07:12.342145 139759569127296 model_lib_v2.py:651] Step 7000 per-step time 0.769s loss=1.267
INFO:tensorflow:Step 7100 per-step time 0.778s loss=0.941
I0320 02:08:31.708401 139759569127296 model_lib_v2.py:651] Step 7100 per-step time 0.778s loss=0.941
2021-03-20 02:08:35.026948: W tensorflow/core/lib/png/png_io.cc:88] PNG warning: iCCP: known incorrect sRGB profile
INFO:tensorflow:Step 7200 per-step time 0.777s loss=0.965
I0320 02:09:51.310583 139759569127296 model_lib_v2.py:651] Step 7200 per-step time 0.777s loss=0.965
2021-03-20 02:09:53.831667: W tensorflow/core/lib/png/png_io.cc:88] PNG warning: iCCP: known incorrect sRGB profile
2021-03-20 02:10:06.738163: W tensorflow/core/lib/png/png_io.cc:88] PNG warning: iCCP: known incorrect sRGB profile
2021-03-20 02:10:10.603601: W tensorflow/core/lib/png/png_io.cc:88] PNG warning: iCCP: known incorrect sRGB profile
INFO:tensorflow:Step 7300 per-step time 0.811s loss=1.146
I0320 02:11:10.437998 139759569127296 model_lib_v2.py:651] Step 7300 per-step time 0.811s loss=1.146
2021-03-20 02:11:25.078766: W tensorflow/core/lib/png/png_io.cc:88] PNG warning: iCCP: known incorrect sRGB profile
```

Figura 19 – Imagem mostrando os logs do modelo Mobilenet ao longo do treinamento do modelo de forma textual. Fonte: O autor

5 Resultados

Após treinado ambos os modelos e feito a inferência na base de testes, foi coletado dados a respeito da precisão de cada modelo, bem como o tempo médio de execução das inferências em cada modelo no hardware Jetson Nano. A base de dados usada para inferência em ambos os modelos foi a mesma, com 125 imagens do trânsito brasileira e contendo as placas a serem identificadas.

Foi executado todo o processo descrito no capítulo anterior para cada modelo SSD Resnet-50 v1 FPN e SSD Mobilenet , desde a obtenção do modelo pré-treinado, passando pela transferência de aprendizado para o problema das placas de trânsito até a execução do mesmo do Jetson Nano. Depois de todo o processo feito, foi feito a catalogação a respeito do grau de precisão de cada modelo, até o tempo médio de inferência utilizando o hardware NVidia Jetson Nano.Foram desenvolvidos três análises do modelo, sendo duas com base no grau de assertividade do modelo e a terceira, com base no tempo de resposta. As métricas foram:

1. Uma análise com base nos itens identificados e não identificados em cada imagem
2. Uma análise sobre quantas placas foram corretamente identificadas numa imagem
3. O tempo médio de inferência de cada modelo

Como dados de benchmark a respeito do grau de acerto do modelo para a primeira métrica, foi considerado:

- Quantidade de placas corretamente identificadas numa imagem
- Quantidade placas não identificadas numa imagem
- Quantidade de placas confundidas numa imagem, no qual uma placa foi confundida com outra
- Quantidade de objetos não catalogados confundidos como se fossem placas de trânsito

A segunda métrica consistia em contabilizar quantas classes(placas de trânsito) foram corretamente identificadas(verdadeiros positivos) em cada imagem e quantas não foram(falsos negativos). Além disso, foram calculados as métricas: Precision,Recall(POWERS; AILAB, 2011) e F1 para cada classe sendo calculado de acordo com a equação abaixo.

$$Recall = \frac{TP}{TP + FP} \tag{5.1}$$

$$Precision = \frac{TP}{TP + FN} \tag{5.2}$$

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{5.3}$$

Sendo TP os verdadeiros positivos, FP os falsos positivos e FN os falsos negativos.

- Precision(precisão) é uma boa medida para determinar quando os custos de um falso positivo são altos. Por exemplo, detecção de spam de e-mail. Na detecção de spam de e-mail, um falso positivo significa que um e-mail que não é spam (verdadeiro negativo) foi identificado como spam (spam previsto). O usuário de email pode perder emails importantes se a precisão não for alta para o modelo de detecção de spam.(SHUNG, 2018)
- Recall(recuperação) calcula quantos dos positivos reais o nosso modelo captura, rotulando-o como positivo (positivo verdadeiro). Aplicando o mesmo entendimento, sabemos que o Recall será a métrica do modelo que usaremos para selecionar nosso melhor modelo quando houver um alto custo associado ao Falso Negativo.(SHUNG, 2018)
- A pontuação F1 é necessária quando você deseja buscar um equilíbrio entre Precision(precisão) e Recall (recuperação). a Além disso, a pontuação F1 pode ser um melhor medida a ser usada se precisarmos buscar um equilíbrio entre precisão e recall E houver uma distribuição de classe desigual (grande número de verdadeiros negativos).(SHUNG, 2018)

Para se obter as métricas acima de cada classe, foi necessário obter as seguintes informações de cada classe para cada modelo a ser validado:

- Verdadeiros positivos: É quando o modelo identifica corretamente uma classe(placa de trânsito) na imagem;
- Falsos negativos: É quando a classe(placa de trânsito) existe na imagem mas ela não foi identificada pelo modelo;
- Falsos positivos: É quando o modelo diz que uma determinada classe foi encontrada na imagem, entretanto, não existe na imagem a classe informada.

Para se medir o tempo médio de cada inferência, foi contabilizado o tempo de inferência de cada modelo, somado e dividido pela quantidade de inferências que foram feitas como na equação abaixo.

$$tempomedio = \frac{(\sum_{i=1}^{n} x^i)}{n} \tag{5.4}$$

É importante mencionar que as métricas definidas foram validadas juntamente com a base de validação e não com a base de treinamento.

5.1 Testes de benchmark(precisão e tempo de inferência) SSD Resnet

De acordo com as métricas de benchmarking definidas previamente, foi obtido para o modelo SSD Resnet v1 FPN:

Tabela 2 – Métricas de precisão obtidas para o primeiro modelo treinado. Fonte: O autor

Métrica	Quantidade
1)Identificados corretamente	54
2)Não identificados	93
3)Placas confundidas	8
4)Objetos confundidos com placas	3

Somando todos os erros e acertos, foram 54 acertos contra 104 erros nas imagens como a 20, mostrando que o algoritmo teve uma precisão de 34.18%, para os dados em questão.

Além disso, é importante mencionar que o modelo em questão teve um tempo médio de 1.8 segundos para processar cada imagem utilizada para os testes em questão.

Tabela 3 – Métricas de precisão obtidas para cada placa isoladamente nas imagens de teste. Fonte: O autor

Classe(Placa de trânsito)	Verdadeiros positivos	Falsos negativos	Falsos positivos
Permitido estacionar	17	23	3
Proibido estacionar	16	37	3
Proibido parar e estacionar	8	19	2
Pare	13	10	0
Proibido virar à direita	0	7	1

Com base nos dados acima, a tabela abaixo mostra as métricas: Precision, Recall e F1(média harmônica) para cada classe:

Tabela 4 – Métricas Precision, Recall e F1 para cada classe, executada na base de testes para o modelo atual. Fonte: O autor

Classe(Placa de trânsito)	Precision	Recall	Média harmônica(F1)
Permitido estacionar	0.85	0.425	0.567
Proibido estacionar	0.842	0.302	0.444
Proibido parar e estacionar	0.8	0.296	0.432
Pare	1	0.565	0.722
Proibido virar à direita	0	0	0

Com base na tabela acima, percebe-se que, com base nas placas de trânsito encontradas e analisadas, a placa melhor identificada foi a placa "pare", com uma média harmônica de 0.722 e, a pior placa a ser identificada foi a placa "proibido virar à direita", com uma média harmônica de 0.

Figura 20 – Imagem mostrando uma inferência feita pelo modelo 1 numa das imagens de teste. Fonte: O autor

5.2 Testes de benchmark(precisão e tempo de inferência) SSD Mobilenet

De acordo com as métricas de benchmarking definidas previamente, foi obtido para o modelo SSD Mobilenet v1 FPN:

Tabela 5 – Métricas de precisão obtidas para o segundo modelo treinado. Fonte: O autor

Métrica	Quantidade
1)Identificados corretamente	87
2)Não identificados	56
3)Placas confundidas	18
4)Objetos confundidos com placas	7

Somando todos os erros e acertos, foram 87 acertos contra 81 erros em imagens como a imagem 21, mostrando que o algoritmo teve uma precisão de 51.78%, podendo ser considerados, dada o tamanho relativamente pequeno da base de dados quando se comparada a bases como a COCO, a uma precisão aceitável.

Foi desenvolvido uma análise de cada acertos e erros individuais de cada placa nas imagens, onde, caso a placa fosse corretamente identificada na imagem, era considerada um acerto, caso contrário, um erro. Dessa forma, obteve-se a seguinte relação:

Tabela 6 – Métricas de precisão obtidas para cada placa isoladamente nas imagens de teste. Fonte: O autor

Classe(Placa de trânsito)	Verdadeiros positivos	Falsos negativos	Falsos positivos
Permitido estacionar	25	15	7
Proibido estacionar	31	22	12
Proibido parar e estacionar	12	15	2
Pare	19	4	2
Proibido virar à direita	0	7	3

Assim como no modelo anterior, a tabela abaixo exibe os resultados obtidos com base nas métricas: Precision, Recall e F1(média harmônica):

Tabela 7 – Métricas Precision, Recall e F1 para cada classe, executada na base de testes para o modelo atual. Fonte: O autor

Classe(Placa de trânsito)	Precision	Recall	Média harmônica(F1)
Permitido estacionar	0.781	0.625	0.694
Proibido estacionar	0.721	0.585	0.646
Proibido parar e estacionar	0.857	0.444	0.585
Pare	0.905	0.826	0.864
Proibido virar à direita	0	0	0

Com base na tabela acima, percebe-se que, com base nas placas de trânsito encontradas e analisadas, a placa melhor identificada foi a placa "pare", com uma média harmônica

de 0.864 e, a pior placa a ser identificada foi a placa "proibido virar à direita", com uma média harmônica de 0.

Além disso, é importante mencionar que o modelo em questão teve um tempo médio de 1.104 segundos para processar cada imagem utilizada para os testes em questão.

Figura 21 – Imagem mostrando uma inferência feita pelo modelo 2 numa das imagens de teste. Fonte: O autor

5.3 Comparativo entre os modelos selecionados

Analisando-se os resultados obtidos para ambos os modelos na base de validação para as métricas definidas, obteve-se:

- De acordo com a métrica 1, o segundo modelo foi aproximadamente 17.6% mais eficiente do que o primeiro;
- De acordo com a métrica 2, tivemos os seguintes resultados para F1(média harmônica) sendo calculados como uma diferença do valor obtido do segundo modelo menos o valor obtido pelo primeiro modelo:
    1. Permitido estacionar: 0.127
    2. Proibido estacionar: 0.202
    3. Proibido parar e estacionar: 0.153
    4. Pare: 0.142
    5. Proibido virar à direita: 0
- De acordo com a métrica 3, o segundo modelo foi, em média, 700ms mais rápido do que o primeiro modelo para fazer as inferências em cada imagem.

Com base nos dados mencionados anteriormente, percebe-se que o segundo modelo teve um desempenho superior ao primeiro, visto que, nos dois benchmarks utilizados(precisão e tempo de resposta), o mesmo foi superior, demonstrando uma maior capacidade para resolver o problema principal descrito neste trabalho.

É importante observar que,embora a taxa de acerto foi diferente, a ordem da placa mais corretamente identificada para a menos corretamente identificada(de acordo com a média harmônica de cada classe) foi igual em ambos os modelos, sendo:

1. Pare (melhor identificada pelos modelos);
2. Permitido estacionar;
3. Proibido estacionar;
4. Proibido parar e estacionar;
5. Proibido virar à direita (pior identificada pelos modelos).

O fato da ordem das classificações ter sido a mesma para ambos os modelos indica que o problema de precisão pode estar na base de dados incompleta e não nos modelos, sendo um bom indicador para uma futura melhoria deste trabalho, começar aumentando a base de dados, dando prioridade para as classes com menor taxa de acerto, para melhorar a taxa de acerto dos modelos. Vale ressaltar que isso não descarta melhorias a serem feitas na arquitetura dos modelos propriamente dita, apenas que melhorar a base de dados pode trazer resultados mais significativos num primeiro momento.

Um detalhe que é importante observar foi a distorção no padrão de cores das imagens como nas imagens 20 e 21. Tal distorção ocorreu devido a um erro de conversão dos pixels num formato de 0 a 1 utilizado pelas ferramentas de visualização e, no formato de 0 a 255 utilizado pela imagem original. Além disso, essa distorção das imagens não compromete o resultado do trabalho, sendo portanto, apenas um efeito estético indesejado.

É importante mencionar também que, ao carregar o ambiente com o Tensorflow no hardware Jetson Nano, um tempo maior foi gasto a fim de carregar a ferramenta e o modelo para a memória RAM, entretanto, esse tempo não foi considerado devido ao fato de ser feito apenas uma vez, o que pode ser feito, caso seja um carro autônomo, antes do mesmo ser utilizado em rodovias e/ou em algum lugar seguro antes de seu uso.

6 Conclusão

A fim de resolver o problema proposto neste trabalho, foi necessário criar uma base de dados contendo dados relevantes do trânsito brasileiro, similar ao que seria encontrado por um veículo autônomo, além de um modelo que detectasse as placas de trânsito em cada imagem com base nas métricas de benchmarking previamente definidas. Para tal, foi necessário, além da base de dados, estudar os princípios de Deep Learning e Inteligência Artificial, áreas sendo amplamente utilizadas para o desenvolvimento nas pesquisas de carros e veículos autônomos.

Dessa forma, foi desenvolvido dois modelos capazes de detectar placas de trânsito brasileiras com tempo de resposta e eficiências diferentes. Embora o segundo modelo tenha sido 17.60% mais preciso(de acordo com a primeira métrica) em suas inferências e aproximamente 700 ms mais rápido, ainda assim o trabalho não foi o suficiente para ser utilizado em sistemas reais de carros autônomos visto que, à medida que a velocidade média do carro aumenta, o tempo de resposta do veículo precisará ser menor a fim de não provocar um acidente e até mesmo, uma fatalidade.

Entretanto, com o que foi desenvolvido até aqui, funciona como um trabalho a ser melhorado em diversas formas possíveis.

6.1 Trabalhos futuros

Como mencionado anteriormente, para que este trabalho seja capaz de ser realmente utilizado em um carro autônomo, destacam-se os seguintes trabalhos futuros e/ou melhorias:

- Reduzir o tamanho das imagens de entrada, sendo o tamanho atual como 640x640 pixels;
- Testar com hardwares diferentes, desde arquiteturas diferentes como variações da ARM e modelos que usam TPU (Tensor Processing Unit);
- Testar com outros hardwares de baixo custo como o Intel NCS,Google Coral, Rasbperry 3 e Raspberry 4;

- Aumentar a base de dados, começando pelas classes que obtiveram as piores taxas de acerto, como a classe "proibido virar à direita";
- Aumentar a base de dados para incluir novas placas de trânsito brasileiras;
- Testar outros modelos de CNN, visto que, a arquitetura utilizada é considerada como "caixa preta"e respeita fortemente uma arquitetura definida, facilitando assim,a troca de um modelo de CNN por outro.

Referências

KOO Ping Shung. 2019. Citado nas páginas 6 e 46.

2021. Citado na página 41.

ACKERMAN, E. Lidar that will make self-driving cars affordable [News]. *IEEE Spectrum*, v. 53, n. 10, p. 14–14, 2016. ISSN 0018-9235. Citado na página 14.

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. Citado na página 20.

BRAGA ANDRé PONCE DE LEON E DE CARVALHO, T. B. L. Antônio de P. *Redes Neurais Artificiais: Teoria e Aplicações*. [S.l.]: LTC, 2007. ISBN 9788521615644. Citado na página 18.

BROGGI A.AND BERTOZZI, A. F. A.; GUARINO, C. The argo autonomous vehicles vision and control systems. international journal on inteligent control systems. p. 409–441, 1999. Citado na página 25.

CAMPOS, D. H. C. CriaÇÃo e validaÇÃo de uma base de dados com alguns elementos do trÂnsito brasileiro para veÍculos autÔnomos. v. 1, n. 1, 2019. Citado nas páginas 14, 45 e 47.

CHEN, J. et al. Deep learning-based model for detecting 2019 novel coronavirus pneumonia on 1 high-resolution computed tomography: a prospective study. 2020. Citado na página 12.

DARPA. Disponível em: <https://www.darpa.mil/about-us/timeline/-grand-challenge-for-autonomous-vehicles>. Citado na página 33.

DARPA. 2020. Citado na página 25.

DAS, S.; MISHRA, S. K. A review on vision based control of autonomous vehicles using artificial intelligence techniques. *Proceedings - 2019 International Conference on Information Technology, ICIT 2019*, IEEE, p. 500–504, 2019. Citado na página 13.

DEMUSH, R. 2019. Disponível em: <https://hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3>. Citado nas páginas 30 e 31.

DICKMANNS, E. D.; ZAPP, A. Autonomous high speed road vehicle guidance by computer vision. p. 232–237, 1987. Citado na página 25.

ESTADAO. *Carro sem motorista do Google já dirigiu por 3,2 milhões de quilômetros*. 2016. Citado na página 33.

GERHARDT, T. E.; SILVEIRA, D. T. *Métodos de Pesquisa*. [S.l.: s.n.], 2009. Citado na página 37.

GOEDEGEBUURE, D. *You Are Helping Google AI Image Recognition*. 2016. Disponível em: <https://medium.com/@thenextcorner/you-are-helping-google-ai-image-recognition-b24d89372b7e>. Citado na página 13.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. Citado nas páginas 18 e 19.

GOOGLE. 2019. Citado na página 13.

GOOGLE. *O que é o Street View?* 2020. Disponível em: <https://www.google.com.br/intl/pt-BR/streetview/>. Citado na página 24.

HECHT, J. Lidar for Self-Driving Cars. n. January, p. 26–33, 2018. Disponível em: <https://www.osapublishing.org/DirectPDFAccess/EABE935F-C02D-41AC-B362A0B1229A6AE2_380434/opn-29-1-26.pdf?da=1&id=380434&seq=0&mobile=no>. Citado na página 13.

HOWARD, A. G. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. Disponível em: <http://arxiv.org/abs/1704.04861>. Citado na página 48.

JAMASMIE, C. Lonely trucks in a lonely place: Autonomous trucks debut in chile's desert. p. 23–24, 2009. Citado na página 25.

JOVIC, A.; BRKIć, K.; BOGUNOVIć, N. An overview of free software tools for general data mining. in: 2014 37th international convention on information and communication technology, electronics and microelectronics. p. 1112–1117, 2014. Citado na página 35.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 60, n. 6, p. 84–90, maio 2012. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/3065386>. Citado nas páginas 12 e 32.

LECUN, Y.; BENGIO, Y.; HINTON, G. *Deep Learning*. [S.l.: s.n.], 2015. <https://www.nature.com/articles/nature14539>. Citado na página 22.

Lecun, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2324, 1998. Citado na página 29.

LIMA, D. alves de. Navegação segura de um carro autônomo utilizando campos vetoriais e o método da janela dinâmica. p. 22–23, 2010. Citado na página 25.

LIN, T. et al. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016. Disponível em: <http://arxiv.org/abs/1612.03144>. Citado nas páginas 6, 23 e 24.

Lin, T. et al. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 42, n. 2, p. 318–327, 2020. Citado nas páginas 6, 46 e 47.

LIU, W. et al. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, Springer International Publishing, p. 21–37, 2016. ISSN 1611-3349. Disponível em: <http://dx.doi.org/10.1007/978-3-319-46448-0_2>. Citado nas páginas 6, 22, 23 e 47.

Lu, X. et al. Object detection based on ssd-resnet. In: *2019 IEEE 6th International Conference on Cloud Computing and Intelligence Systems (CCIS)*. [S.l.: s.n.], 2019. p. 89–92. Citado na página 41.

MCCULLOCH, W. S.; PITTS, W. A logical calculus nervous activity. p. 1–2, 1943. Citado nas páginas 6, 27 e 28.

MITCHELL, T. M. *Machine Learning*. [S.l.]: McGraw-Hill Science/Engineering/Math, 1997. Citado na página 18.

NGO, H. T. et al. Learning representations by back-propagating errors. p. 3, 2009. Citado nas páginas 6 e 31.

NGUYEN, G. et al. Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey. p. 87–88, 2019. Citado nas páginas 6, 35 e 36.

NVIDIA. 2016. Citado nas páginas 6 e 19.

NVIDIA. 2018. Citado na página 21.

NVIDIA. 2020. Disponível em: <https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit>. Citado nas páginas 6 e 39.

NVIDIA. 2020. Disponível em: <https://www.nvidia.com/pt-br/autonomous-machines/embedded-systems/jetson-nano/>. Citado na página 25.

POWERS, D.; AILAB. Evaluation: From precision, recall and f-measure to roc, informedness, markedness correlation. *J. Mach. Learn. Technol*, v. 2, p. 2229–3981, 01 2011. Citado na página 54.

PRABHU. *Understanding of Convolutional Neural Network (CNN) — Deep Learning*. 2018. Disponível em: <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148>. Citado nas páginas 6 e 22.

RUMELHART, D. E. et al. Learning representations by back-propagating errors. p. 321–361, 1986. Citado na página 30.

RUSSEL, P. N. S. *Inteligência Artificial*. [S.l.]: Campus,Elsevier, 2013. ISBN 9788535237016. Citado nas páginas 16, 26 e 29.

SAHA, S. 2018. Citado na página 12.

SELF-DRIVING Uber car that hit and killed woman did not recognize that pedestrians jaywalk. Citado nas páginas 6 e 34.

SHUNG, K. P. 2018. Citado na página 54.

TEAM, E. S. 2020. Citado na página 12.

TEAM, W. *Waymo's fleet reaches 4 million self-driven miles*. 2017. Citado na página 34.

TECHOPEDIA. 2019. Citado na página 12.

TOUVRON, H. et al. Fixing the train-test resolution discrepancy. In: *Advances in Neural Information Processing Systems*. [s.n.], 2019. v. 32. ISSN 10495258. Disponível em: <http://arxiv.org/abs/2003.08237>. Citado na página 12.

VOELCKER, J. 2012. Citado na página 33.

WEST, D. M.; ALLEN, J. R. 2018. Citado na página 12.

WE'RE not prepared for the end of Moore's Law. Citado na página 32.

YOUNIS, A. et al. Real-time object detection using pre-trained deep learning models mobilenet-ssd. In: *Proceedings of 2020 the 6th International Conference on Computing and Data Engineering*. New York, NY, USA: Association for Computing Machinery, 2020. (ICCDE 2020), p. 44–48. ISBN 9781450376730. Disponível em: <https://doi.org/10.1145/3379247.3379264>. Citado na página 41.

ZHANG, Y.; PENG, H.; HU, P. CS341 Final Report: Towards Real-time Detection and Camera Triggering. *CS341 Final Report*, p. 9, 2017. Disponível em: <http://files/96/Zhangetal.-CS341FinalReportTowardsReal-timeDetectionan.pdf>. Citado nas páginas 6, 48 e 49.

Apêndices

APÊNDICE A – Arquivos de pipeline dos modelos utilizados

A.1 pipeline.config do RetinaNet(Modelo 1)

```
model {
  ssd {
    num_classes: 5
    image_resizer {
      fixed_shape_resizer {
        height: 640
        width: 640
      }
    }
    feature_extractor {
      type: "ssd_resnet50_v1_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 0.00039999998989515007
          }
        }
        initializer {
          truncated_normal_initializer {
            mean: 0.0
            stddev: 0.029999999329447746
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.996999979019165
          scale: true
          epsilon: 0.0010000000474974513
        }
      }
      override_base_feature_extractor_hyperparams: true
      fpn {
```

```
    min_level: 3
    max_level: 7
  }
}
box_coder {
  faster_rcnn_box_coder {
    y_scale: 10.0
    x_scale: 10.0
    height_scale: 5.0
    width_scale: 5.0
  }
}
matcher {
  argmax_matcher {
    matched_threshold: 0.5
    unmatched_threshold: 0.5
    ignore_thresholds: false
    negatives_lower_than_unmatched: true
    force_match_for_each_row: true
    use_matmul_gather: true
  }
}
similarity_calculator {
  iou_similarity {
  }
}
box_predictor {
  weight_shared_convolutional_box_predictor {
    conv_hyperparams {
      regularizer {
        l2_regularizer {
          weight: 0.00039999998989515007
        }
      }
      initializer {
        random_normal_initializer {
          mean: 0.0
          stddev: 0.009999999776482582
        }
      }
      activation: RELU_6
      batch_norm {
        decay: 0.996999979019165
```

```
        scale: true
        epsilon: 0.0010000000474974513
      }
    }
    depth: 256
    num_layers_before_predictor: 4
    kernel_size: 3
    class_prediction_bias_init: -4.599999904632568
  }
}
anchor_generator {
  multiscale_anchor_generator {
    min_level: 3
    max_level: 7
    anchor_scale: 4.0
    aspect_ratios: 1.0
    aspect_ratios: 2.0
    aspect_ratios: 0.5
    scales_per_octave: 2
  }
}
post_processing {
  batch_non_max_suppression {
    score_threshold: 9.99999993922529e-09
    iou_threshold: 0.6000000238418579
    max_detections_per_class: 100
    max_total_detections: 100
    use_static_shapes: false
  }
  score_converter: SIGMOID
}
normalize_loss_by_num_matches: true
loss {
  localization_loss {
    weighted_smooth_l1 {
    }
  }
  classification_loss {
    weighted_sigmoid_focal {
      gamma: 2.0
      alpha: 0.25
    }
  }
```

```
      classification_weight: 1.0
      localization_weight: 1.0
    }
    encode_background_as_zeros: true
    normalize_loc_loss_by_codesize: true
    inplace_batchnorm_update: true
    freeze_batchnorm: false
  }
}
train_config {
  batch_size: 5
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
      overlap_thresh: 0.0
    }
  }
  sync_replicas: true
  optimizer {
    momentum_optimizer {
      learning_rate {
        cosine_decay_learning_rate {
          learning_rate_base: 0.03999999910593033
          total_steps: 25000
          warmup_learning_rate: 0.013333000242710114
          warmup_steps: 2000
        }
      }
      momentum_optimizer_value: 0.8999999761581421
    }
    use_moving_average: false
  }
  fine_tune_checkpoint: "pre-trained-models/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/chec
  num_steps: 250000
  startup_delay_steps: 0.0
```

```
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  use_bfloat16: false
  fine_tune_checkpoint_version: V2
}
train_input_reader {
  label_map_path: "annotations/label_map.pbtxt"
  tf_record_input_reader {
    input_path: "annotations/train.record"
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}
eval_input_reader {
  label_map_path: "annotations/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "annotations/test.record"
  }
}
```

A.2 pipeline.config do Mobile-Det(Modelo 2)

```
model {
  ssd {
    num_classes: 5
    image_resizer {
      fixed_shape_resizer {
        height: 640
        width: 640
      }
    }
    feature_extractor {
      type: "ssd_mobilenet_v1_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
```

```
      l2_regularizer {
        weight: 3.9999998989515007e-05
      }
    }
    initializer {
      random_normal_initializer {
        mean: 0.0
        stddev: 0.009999999776482582
      }
    }
    activation: RELU_6
    batch_norm {
      decay: 0.996999979019165
      scale: true
      epsilon: 0.0010000000474974513
    }
  }
  override_base_feature_extractor_hyperparams: true
  fpn {
    min_level: 3
    max_level: 7
  }
}
box_coder {
  faster_rcnn_box_coder {
    y_scale: 10.0
    x_scale: 10.0
    height_scale: 5.0
    width_scale: 5.0
  }
}
matcher {
  argmax_matcher {
    matched_threshold: 0.5
    unmatched_threshold: 0.5
    ignore_thresholds: false
    negatives_lower_than_unmatched: true
    force_match_for_each_row: true
    use_matmul_gather: true
  }
}
similarity_calculator {
  iou_similarity {
```

```
    }
  }
  box_predictor {
    weight_shared_convolutional_box_predictor {
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 3.9999998989515007e-05
          }
        }
        initializer {
          random_normal_initializer {
            mean: 0.0
            stddev: 0.009999999776482582
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.996999979019165
          scale: true
          epsilon: 0.0010000000474974513
        }
      }
      depth: 256
      num_layers_before_predictor: 4
      kernel_size: 3
      class_prediction_bias_init: -4.599999904632568
    }
  }
  anchor_generator {
    multiscale_anchor_generator {
      min_level: 3
      max_level: 7
      anchor_scale: 4.0
      aspect_ratios: 1.0
      aspect_ratios: 2.0
      aspect_ratios: 0.5
      scales_per_octave: 2
    }
  }
  post_processing {
    batch_non_max_suppression {
      score_threshold: 9.99999993922529e-09
```

```
        iou_threshold: 0.6000000238418579
        max_detections_per_class: 100
        max_total_detections: 100
        use_static_shapes: false
      }
      score_converter: SIGMOID
    }
    normalize_loss_by_num_matches: true
    loss {
      localization_loss {
        weighted_smooth_l1 {
        }
      }
      classification_loss {
        weighted_sigmoid_focal {
          gamma: 2.0
          alpha: 0.25
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    encode_background_as_zeros: true
    normalize_loc_loss_by_codesize: true
    inplace_batchnorm_update: true
    freeze_batchnorm: false
  }
}
train_config {
  batch_size: 5
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
      overlap_thresh: 0.0
    }
```

```
  }
  sync_replicas: true
  optimizer {
    momentum_optimizer {
      learning_rate {
        cosine_decay_learning_rate {
          learning_rate_base: 0.03999999910593033
          total_steps: 25000
          warmup_learning_rate: 0.013333000242710114
          warmup_steps: 2000
        }
      }
      momentum_optimizer_value: 0.8999999761581421
    }
    use_moving_average: false
  }
  fine_tune_checkpoint: "pre-trained-models/ssd_mobilenet_v1_fpn_640x640_coco17_tpu-8/che
  num_steps: 250000
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  fine_tune_checkpoint_version: V2
}
train_input_reader {
  tf_record_input_reader {
    input_path: "annotations/train.record"
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
  batch_size: 1
}
eval_input_reader {
  label_map_path: "annotations/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "annotations/test.record"
  }
}
```