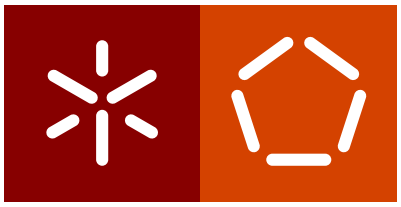


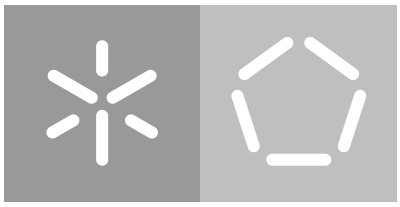
Universidade do Minho
Escola de Engenharia
Departamento de Eletrónica Industrial

João Pedro de Oliveira Faria

Sistema de visão detetor de automóveis em 3D e em tempo real para veículos autónomos

A real-time 3D car-detector vision system for autonomous vehicles

Guimarães, Fevereiro de 2023



Universidade do Minho
Escola de Engenharia
Departamento de Eletrónica Industrial

João Pedro de Oliveira Faria

Sistema de visão detetor de automóveis em 3D e em tempo real para veículos autónomos

A real-time 3D car-detector vision system for autonomous vehicles

Dissertação de Mestrado
Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do
Professor Doutor Jaime Francisco Cruz Fonseca
Doutor João Borges

Guimarães, Fevereiro de 2023

AGRADECIMENTOS

Aos meus pais que sempre me suportaram e apoiaram-me durante estes 5 anos incondicionalmente.

Ao meu irmão que me ajudou nos momentos mais difíceis a definir um curso de ação a tomar bem como a ganhar ainda mais sensibilidade no que diz respeito à carga de trabalho e tempo necessário na realização dos projetos durante todo o meu percurso académico.

Aos meus orientadores que me deixaram escolher um tema livre e que prontamente me puderam ajudar ao dar indicações sobre as metodologias de trabalho e a definir as ferramentas a utilizar. À equipa do *motionlab* sempre pronta para me ajudar em qualquer momento na tese.

Aos meus amigos pelo convívio, camaradagem e partilha dos bons e mais difíceis momentos aquando a produção desta tese.

A todos o meu muito obrigado!

RESUMO

Atualmente, a indústria dos veículos autónomos encontra-se em rápido crescimento perspetivando-se uma revolução na forma como os meios de transporte são utilizados bem como a experiência de condução. Consequentemente, a transição para uma maior autonomia na capacidade de condução necessita, em primeira instância, de que o veículo percecione o ambiente e tome decisões tão rapidamente como as pessoas. Idealmente, que desempenhe o exercício de condução ainda melhor que estas. Para isso, é essencial o desenvolvimento de um sistema de visão para uma interpretação em tempo real do meio exterior, onde as entidades que o constituem sejam corretamente detetadas em frações de segundos, no contexto deste problema. Para isso, foi efetuado um estudo do desempenho de redes neuronais de modalidade singular e multi-modal quando as mesmas são treinadas com conjunto de dados 100% reais, 100% sintéticos ou híbridos. Pretende-se assim averiguar até que ponto os dados sintéticos permitem para melhorar a precisão dos algoritmos, em qual das modalidades são mais eficazes, que são mais fáceis de produzir em massa e menos dispendiosos, e se o pré-treino com conjunto de dados híbridos antes do treino em conjunto de dados reais melhora o desempenho na deteção de veículos do que se os mesmos detetores fossem apenas treinados com conjunto de dados reais. Concluiu-se que é possível obter melhores resultados de classificação que os de literatura para os modelos Fcos3D e MVX-Net ao pré-treinar estes em conjunto de dados com pelo menos 25% de dados reais, desde que seja efetuado o processo de fine-tuning de seguida em dados reais. O modelo MVX-Net, multi-modal, obteve significativamente melhores resultados que o Fcos3D, modalidade única, para os diferentes tipos de conjunto de dados. Verificou-se também que, em geral, o fine-tuning em dados 100% reais melhorou os resultados de classificação para ambos os modelos, independentemente do conjunto de dados inicial de treino. Finalmente, de todas as experiências realizadas, para ambos os modelos obteve-se os melhores resultados de classificações após o fine-tuning em conjunto de dados puramente reais, em 8 épocas, nos conjunto de dados iniciais de treino compostos por 75% e 100% de dados reais, apesar de se terem atingido resultados superiores aos de literatura também para conjunto de dados com proporções de dados reais de 25% e 50%, o que viabiliza deste modo o uso de dados sintéticos para o treino.

Palavras chave: Veículos Autónomos, câmara, Deteção de Objetos, Fine Tuning, Modalidade Singular, Modalidade Múltipla, conjunto de dados Real, conjunto de dados Sintético, conjunto de dados Híbrido.

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações, ou resultados em nenhuma das etapas conducente à sua elaboração. Mais declaro que conheço e respeitei o Código de Conduta Ética da Universidade do Minho.

ABSTRACT

Currently, the autonomous vehicle industry is growing rapidly, with the prospect of an evolution in the way means of transportation are used as well as the driving experience. Consequently, the transition to greater autonomy in driving requires the vehicle to perceive the environment and make decisions as quickly as people do. Ideally, it should perform the driving exercise even better than they do. For this, it is essential to develop a robust and fail-safe vision system for real-time interpretation of the external environment, where entities are correctly detected in fractions of seconds. To this end, the purpose of this work was to study the performance of singular and multi-modal neural networks when they are trained on datasets with 100% real data, third-party datasets with 100% synthetic or simulated data that preferably are a mimic of the real datasets, and hybrid datasets. Specifically, the goal is to find out to what extent it is feasible to use synthetic data to improve the accuracy of the algorithms, in which way simulation data is most effective, which is easier to mass produce and less expensive, and whether pre-training on hybrid datasets before training on real datasets improves vehicle detection performance, in relation to detectors that were only trained on real datasets. It was concluded that it is possible to obtain better classification results than those in the literature for detection models which were proven for the Fcos3D and MVX-Net models by training them on datasets composed of at least 25% of real data, as long as it is followed by a fine-tuning process of said models in a real-world datasets. The MVX-Net model, multi-modal, obtained significantly better results than Fcos3D, single modality, for all types of datasets. It was also found that, in general, fine-tuning on real 100% data improved the classification results for both models, regardless of the initial training datasets. Finally, from all the experiments performed, for both models, the best classification results were obtained after fine-tuning on purely real datasets, in 8 epochs, in the initial training datasets composed of 75% and 100% of real data, which demonstrates the feasibility of using synthetic data for training, although higher results than in the literature were also achieved for datasets with real data proportions of 25% and 50%, making it feasible to use synthetic data for training.

Keywords: Autonomous Vehicles, Camera, Object Detection, Fine-Tuning, Single Modality, Multiple Modality, Real conjunto de dados, Synthetic conjunto de dados, Hybrid conjunto de dados.

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

ACRÓNIMOS

A

ADAS Advanced Driving Assistance Systems.

B

BEV Birds' Eye View.

C

CNN Convolutional Neural Networks.

D

DL Deep Learning.

DLA Deep Layer Aggregation.

F

FCL Fully Connected Layer.

FOV Field of View.

M

ML Machine Learning.

P

PCD Point Cloud Data.

R

RGB Reg Green Blue.

ROS Robotic Operative System.

S

SAE Sociedade de Engenheiros Automóveis.

SLAM Simultaneous Localization and Mapping.

SSL Solid State Lidar.

U

UE União Europeia.

V

V2I Vehicle to infrastructure.

V2V Vehicle to vehicle.

V2X Vehicle to everything.

VA Veículo autónomo.

1

INTRODUÇÃO

Neste capítulo inicial dá-se a conhecer o âmbito do projeto, o seu enquadramento na sociedade e o estado de trabalhos desenvolvidos em áreas semelhantes. Refere-se do mesmo modo a importância dos progressos realizados na área, as partes interessadas e qual o objetivo final da dissertação.

1.1 Enquadramento

A experiência de condução como atualmente a conhecemos irá ser completamente transformada com o rápido desenvolvimento da tecnologia de condução autónoma. Segundo (Fagnant and Kockelman, 2018), esta tecnologia deve estar comercialmente desenvolvida até ao fim da década. As vantagens que esta tecnologia pode trazer nos meios de transporte parecem ser muitas, tais como, uma maior segurança e um menor impacto ambiental, segundo (Littman, 2015). Em contrapartida, não se pode desvalorizar o caminho que ainda falta até os carros completamente autónomos estarem disponíveis. Em 2019 houve cerca de 22.800 acidentes de condução que resultaram em mortes nos 27 Estados-Membros da *União Europeia (UE)* (EuropeanCommission, 2020). Embora a sinistralidade rodoviária tenha diminuído bastante nos últimos dez anos em relação a relatórios da UE, a mesma também lançou uma série de novos esforços regulamentares para minimizar possíveis ocorrências de colisão automóvel. Em 2019, uma das regras exige que equipamento de segurança avançado deve ser obrigatório em todos os veículos rodoviários novos vendidos no mercado da UE (EuropeanParliamentNews, 2020), nomeadamente, assistentes de condução inteligentes que avisam o condutor quando ultrapassa o limite de velocidade, sofre de sonolência ou ainda notificam este de objetos por trás do veículo recorrendo a câmaras e monitores. Além disso, 95% de todos os acidentes rodoviários na UE são causados por erro humano. Os carros sem condutor poderiam, portanto, atenuar grandemente as hipóteses de acidentes de trânsito, assegurando segurança na condução. Aliás, a questão do congestionamento e do agravamento do efeito de estufa pode ser gradualmente solucionada com a adoção desta nova tecnologia. Níveis 3 e 4 de autonomia em veículos de condução foram recentemente testados no mundo real. Está previsto que veículos totalmente autónomos chegarão em 2030. Nos próximos anos, o mercado de veículos autónomos pode crescer dramaticamente, o que criará mais empregos e levará ao desenvolvimento da indústria automóvel na UE até aos 620 mil milhões até 2025 (EUParliamentNews, 2020).

A investigação de carros autónomos é assim um tema popular na última década. Existem três componentes principais na condução autónoma que são a perceção, o planeamento, e o controlo. A perceção inclui a visão por computador, fusão de sensores, e posicionamento. A visão por computador deteta objetos tais como pistas, motociclos, peões, sinais de trânsito, etc., e pode identificar faixas de trânsito, veículos, semáforos,

e outros detalhes de forma eficaz. A visão digital é semelhante aos olhos humanos, capazes de perceber o mundo. A fusão de sensores é utilizada para aprofundar a compreensão da visão, obter informações tais como a distância do veículo, a velocidade de deslocação de outros objetos, e compreender a relação entre si próprio e o mundo circundante. A localização do veículo por GPS requer uma precisão ao nível do centímetro, já que erros na ordem de metros não podem satisfazer os requisitos necessários para uma transição para um carro completamente autónomo. O planeamento no caminho do mapa de navegação é um plano sumário global que orienta o veículo autónomo para onde ir. O controlo é o último passo da condução autónoma. Controla a direção, rodas, travões, acelerador, luzes, e outros equipamentos de modo a dirigir de forma autónoma ou semi-autónoma o veículo.

1.2 Motivação e importância

São efetuados esforços para uma transição na produção em massa de veículos totalmente manuais para veículos totalmente autónomos. No entanto, como esta vertente da indústria automóvel situa-se ainda longe de atingir o seu objetivo final, existem atualmente 6 níveis de autonomia (figura 1) definidos pela sociedade de engenheiros automóveis (SAE) como metas a atingir pelas mais diversas empresas automóveis na produção de veículos. Esses níveis são as capacidades de autonomia (Staff, 2021). Tendo em conta o investimento das grandes empresas automobilísticas, existe uma procura crescente por engenheiros de inteligência artificial dotados não só de conhecimentos de software, nomeadamente escolha de conjunto de dados, conjuntos de dados do mesmo tipo, arquitetura de modelos para treino e capacidade de avaliação de algoritmos de *Machine Learning (ML)* como também de hardware, por exemplo, na escolha, calibração e fusão sensorial dos mais diversos sensores visuais atualmente usados na condução autónoma. Associado a isso, existe também falta de dados reais dado estes serem custosos de se obterem a nível de tempo, dinheiro ou obtenção de autorizações, o que serve de motivação para estudar a viabilidade de conjuntos de dados sintéticos para treino de algoritmos de condução autónoma.

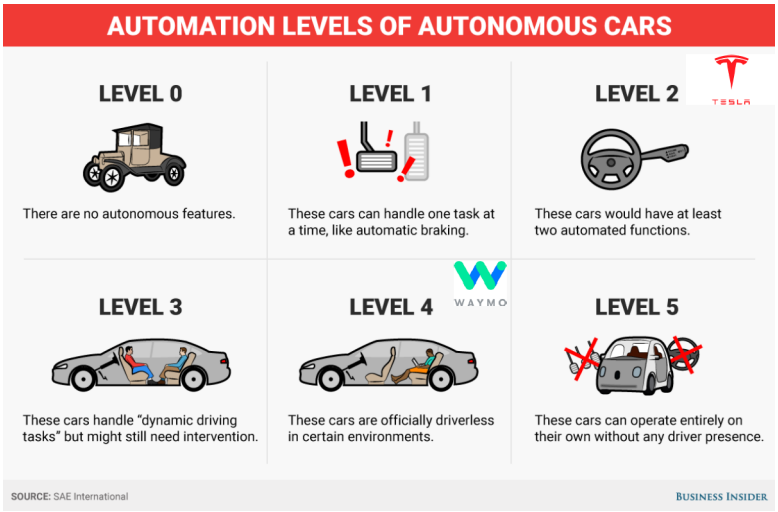


Figura 1: 6 Níveis de autonomia propostos pela SAE, adaptado de (Radovanovic and Muoio, 2016).

1.3 Objetivo

Neste trabalho pretende-se descobrir a viabilidade em treinar redes neuronais para deteção de veículos com conjunto de dados compostos por dados puramente sintéticos e/ou reais mais sintéticos. Particularmente, será analisado em que tipos de redes (modalidade singular ou múltipla) estes dados sintéticos são mais eficazes para melhorar a precisão dos algoritmos. Redes de modalidade singular são redes que apenas consomem um tipo de dados (como imagens de câmara) ao passo que redes multi-modais são redes que podem ser treinadas com mais do que um tipo de dados (dados de câmara e lidar). Além disso, pretende-se verificar se, por exemplo, redes pré-treinadas com conjunto de dados híbridos e/ou puramente sintéticos que posteriormente são treinadas com dados obtidos no mundo real são mais eficazes que redes apenas treinadas com dados reais.

1.4 Estrutura da dissertação

Esta dissertação encontra-se divida em seis capítulos que se encontram organizados e sequenciados para que o leitor consiga compreender de forma simples o trabalho desenvolvido, acompanhando assim detalhadamente o processo realizado.

O primeiro capítulo é constituído pela Introdução, que serve para fazer o enquadramento do trabalho desenvolvido, expondo os objetivos, o propósito da dissertação e a sua estrutura.

O segundo capítulo, Estado da arte, irá explorar e relatar os conceitos envolvidos na elaboração deste projeto. Neste capítulo serão apresentados documentos bibliográficos de interesse direto para a resolução deste projeto. Especificamente, será abordado de modo geral as várias funcionalidades caraterísticas de um sistema de condução autónomo, com ênfase na perceção sensorial. Além disso, serão discutidas as diversas abordagens para obtenção da representação tridimensional dos objetos detetados, critério que levará à escolha do número e categoria de sensores a usar no capítulo três. Também serão apresentados algoritmos de deteção eficientes.

O terceiro capítulo contém o estudo e seleção dos recursos para execução das experiências a realizar. Serão abordados o ambiente de execução, conjunto de dados, estruturas de software e algoritmos utilizados, parâmetros de treino/teste e o planeamento e estruturação de código, bem como quais as experiências a realizar.

O quarto capítulo é dedicado à implementação e preparação dos cenários de execução, nomeadamente, à geração de conjunto de dados híbridos com diferentes proporções de dados reais e sintéticos.

O quinto capítulo é reservado à descrição dos resultados das experiências efetuadas nos cinco conjunto de dados tanto para modelos multi-modais como para modelos de modalidade singular. Os resultados do teste e processo de fine-tune serão amostrados segundo o critério de avaliação do conjunto de dados Kitti.

Por fim, o sexto capítulo é dedicado à conclusão do trabalho desenvolvido com as respostas às perguntas descritas na introdução assim como é efetuado uma alusão ao trabalho futuro sobre o tema.

O documento termina com uma apresentação dos anexos que inclui os ficheiros de código desenvolvidos que foram utilizados para obtenção dos resultados.

2

ESTADO DE ARTE

Neste capítulo serão apresentadas as várias abordagens existentes na perceção de um sistema autónomo, sensores tipicamente utilizados na condução autónoma, fusão sensorial, conjunto de dados, algoritmos de deteção e ambientes de simulação mais usados.

2.1 Abordagens na perceção de um sistema de condução autónomo

Um sistema de condução autónomo possui um hardware associado para sensorização, atuação e opcionalmente comunicação para com outros veículos no meio ambiente. Entre tipos de comunicações com outros veículos destacam-se *Vehicle to vehicle (V2V)*, infraestruturas *Vehicle to infrastructure (V2I)* ou ainda ambos *Vehicle to everything (V2X)*, figura 2. Quanto às funcionalidades inerentes, inicialmente o mesmo começa por interpretar o meio ambiente em que se enquadra assim como achar a sua localização neste para modelar o ambiente e estimar onde realmente se encontra o veículo, isto é, a sua localização, posição, velocidade, aceleração e a orientação do mesmo. Após completar esta etapa, o veículo autónomo planeia o trajeto que fará desde a posição em que se encontra até ao destino definido pelo utilizador (**Motion planning**) traçando a sequência de ruas e caminhos a seguir. Por sua vez, este percurso terá de ter em conta o tráfego e outros obstáculos no caminho para retornar informação como a velocidade de cruzeiro, a distância à berma e quão próximo o veículo pode ficar do veículo da frente, etapas alcançadas no processo de (**Behavioral planning**). Por fim, após definido o percurso bem como interpretado os obstáculos neste passa-se para o estágio de controlo onde os percursos e os obstáculos são interpretados de forma a originar uma sequência de ações em cada instante de tempo que permite uma condução segura levada a cabo pelo veículo de condução autónomo. As ações são nada mais nada menos do que sinais enviados que fazem com que o veículo acelere, trave, vire à esquerda ou direita (**Motion planning**) bem como consiga traçar o caminho a efetuar entre dois pontos de passagem (**Path tracking**) e ainda o tempo necessário a completar esse mesmo caminho (**Trajectory tracking**).

Especificamente, esta tese foca-se apenas na componente de perceção (figura 3) que corresponde à visão por computador, responsável pela captação visual do mundo e fusão sensorial, que permite um melhor descrição da realidade através da conjugação de dados sensoriais de diversas fontes.

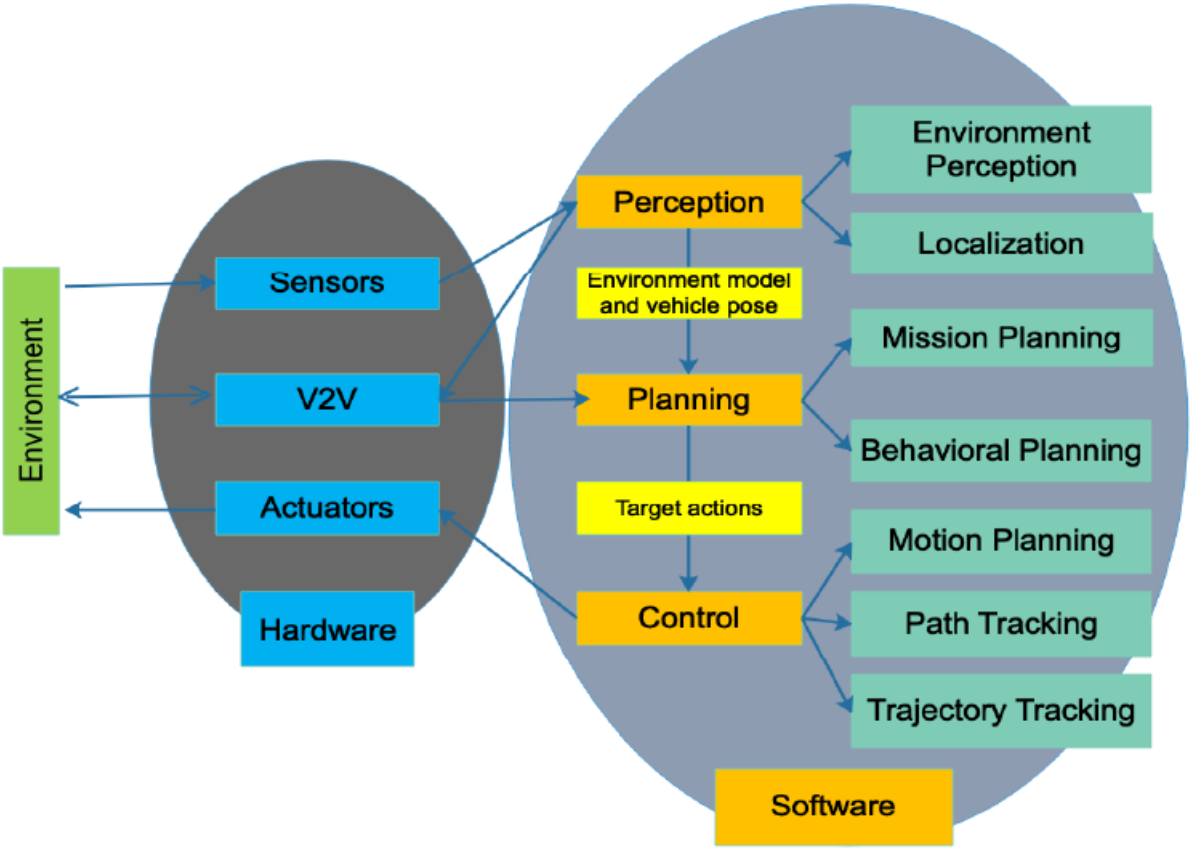


Figura 2: Visão global de um SAE. Imagem obtida de (Ranjan and Senthamilarasu).

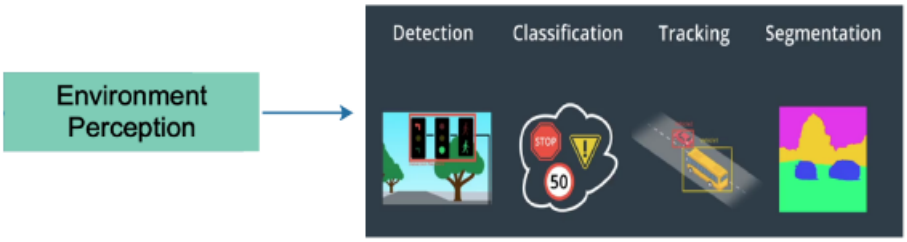


Figura 3: Funcionalidade de perceção e subtarefas: deteção, classificação, traceamento e segmentação. Imagem adaptada de (Ranjan and Senthamilarasu)

2.2 Sensores usados na condução autónoma

Quanto à função de aquisição de dados do meio externo, um sistema de navegação autónomo pode ser munido dos mais diversos sensores exterocetivos, nomeadamente:

- **Câmara**
- **Radar**
- **Lidar**

- **Sensor de ultrassons**
- **Sensores odométricos**

Existem ainda sensores acoplados ao veículo que são responsáveis por adquirir dados propriocetivos, como:

- **GPS**
- **GNSS**
- **IMU**

Tanto GPS (Global Positioning System) como o GNSS (Global National Satellite System) são sistemas de localização do veículo no planeta Terra que usam um sistema de satélites ou vários sistemas destes, respetivamente. Um IMU é um dispositivo usado para medir acelerações, rotações e ainda obter a orientação do corpo a ela associado.

Nas figuras 4 e 5 encontra-se uma representação visual de um possível posicionamento de sensores bem como dos seus respetivos ângulos de visão.

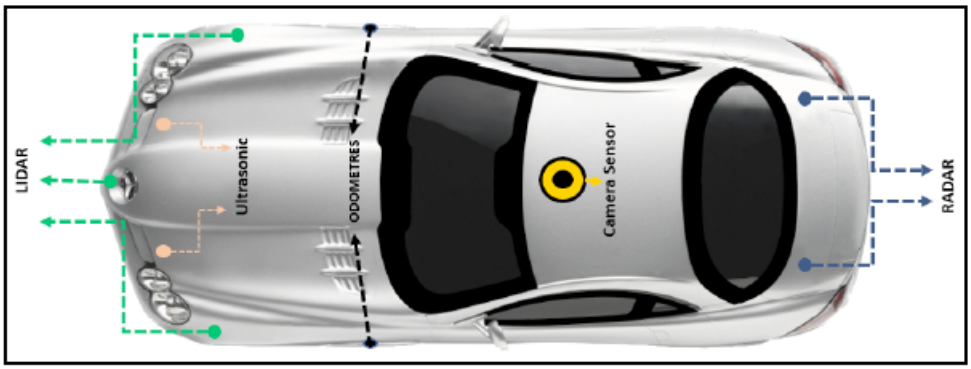


Figura 4: Posicionamento possível dos sensores habitualmente usado num VA. Imagem adaptada de (Ranjan and Senthamilarasu)

Na imagem , retirada de (Starepravo, 2021), é possível verificar o papel dos diferentes sensores quanto às mais diversas operações a realizar aquando na condução. Abaixo segue-se uma breve descrição de cada um destes, informações retiradas de (Yeong et al., 2021a).

2.2.1 *Câmara*

As câmaras são uma das tecnologias mais adotadas para a perceção do meio envolvente. Uma câmara funciona com base no princípio de deteção de luzes emitidas do meio envolvente numa superfície fotossensível (plano de imagem) por uma lente fotográfica (montada em frente do sensor) para produzir imagens claras do exterior (Campbell et al., 2018). As câmaras são relativamente baratas e com o software apropriado, podem detetar tanto os obstáculos móveis como estáticos do campo de visão e ainda fornecer imagens de alta resolução dos arredores. Estas capacidades permitem que o sistema de perceção do veículo identifique a

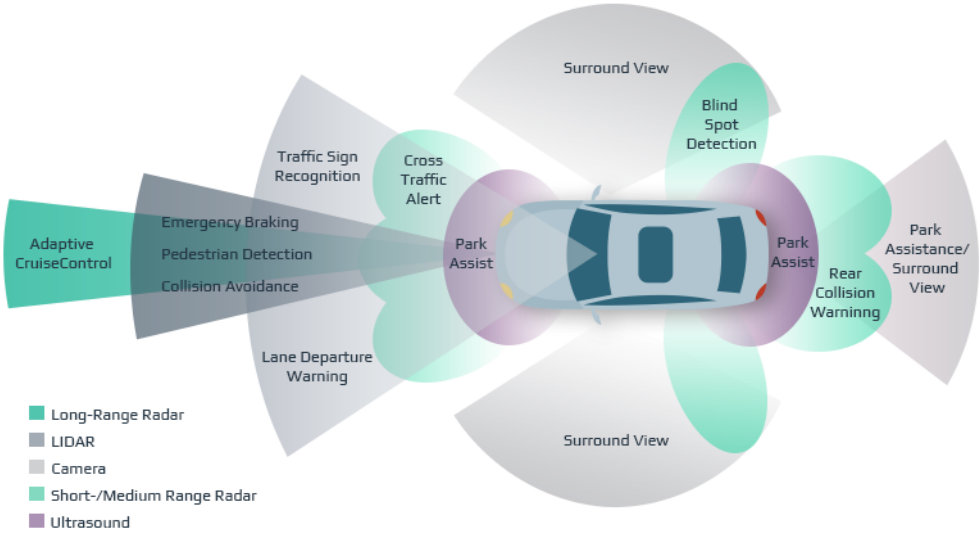


Figura 5: Alguns dos sensores usados num VA.

sinalização rodoviária, os semáforos, a faixa de rodagem, marcas e barreiras no caso de veículos de tráfego rodoviário e uma série de outros objetos no caso dos veículos todo-o-terreno. O sistema de câmaras num VA pode empregar câmaras monoculares ou câmaras binoculares, ou uma combinação de ambas. Como o nome indica, as câmaras monoculares usam uma única câmara para criar uma série de imagens. As câmaras monoculares são fundamentalmente mais limitadas do que as câmaras estereoscópicas, dado lhes faltar informação de profundidade nativa, embora em algumas aplicações ou em determinadas câmaras monoculares mais avançadas essa desvantagem é colmatada com a utilização de hardware de duplo píxel *autofocus*, onde a informação de profundidade pode ser calculada usando algoritmos complexos (Joglekar et al., 2011). Como resultado, duas câmaras são frequentemente instaladas lado a lado para formar um sistema binocular em veículos autónomos. A câmara estereoscópica, também conhecida como câmara binocular, imita a perceção de profundidade encontrada nos animais, em que a "disparidade"entre as imagens ligeiramente diferentes formado em cada olho é subconscientemente utilizado para proporcionar uma sensação de profundidade. As câmaras contém dois sensores de imagem, separados por uma linha de base. O termo linha de base refere-se a a distância entre os dois sensores de imagem (e é geralmente citada nas especificações de câmaras estereoscópicas). Baseado na visão animal, os mapas de disparidade calculados a partir das imagens da câmara estereoscópica permitem a geração de mapas de profundidade usando geometria epipolar e métodos de triangulação. Existem outras câmaras como as *Fisheye* que são normalmente utilizadas em certas aplicações, tais como no estacionamento e na assistência no engarrafamento, e requerem apenas quatro câmaras para proporcionar uma vista de 360 ° graus do meio envolvente. Contudo, muitas vezes torna-se necessário "calibrar intrinsecamente"a câmara para estimar os parâmetros desta e retificar as distorções geométricas (O' Mahony et al., 2018). Além disso, a qualidade (resolução) das imagens capturadas pelas câmaras pode ser significativamente afetada pela iluminação e por condições meteorológicas adversas, por exemplo, neve, sol intenso, tempestade, tempo nublado, entre outras. Por fim, existem ainda outras desvantagens relacionadas com a utilização de câmaras, nomeadamente, a exigência de grande poder de cálculo necessário para efetuar a análise dos dados das imagens criadas (Campbell et al., 2018).

2.2.2 *Radar*

O radar funciona com base no princípio da radiação de ondas eletromagnéticas (EM) na área de interesse e recebe as ondas dispersas (ou reflexões) de alvos para processamento e estabelecimento de informações sobre estes. Utiliza a propriedade das ondas EM para determinar a velocidade relativa e a posição relativa das ondas EM detetadas através do efeito Doppler, também conhecido como Doppler shift, que se refere às mudanças na frequência das ondas resultantes do movimento relativo entre uma fonte de ondas e os seus alvos. Por exemplo, a frequência do sinal recebido aumenta (ondas mais curtas) quando o alvo se desloca na direção do sistema de radar. A equação de Doppler do deslocamento de frequência de um radar pode ser representada como:

$$f_D = \frac{2 * V_r * f}{C} = \frac{2 * V_r}{\lambda} \tag{1}$$

onde f_D é a frequência de Doppler em Hertz (Hz); V_r é a velocidade relativa do alvo; f é a frequência do sinal transmitido; c é a velocidade da luz, descrita na equação 2:

$$c = 3 * 10^8 m/s \tag{2}$$

e Λ é o comprimento de onda da energia emitida. Na prática, a frequência de Doppler num radar ocorre duas vezes; em primeiro lugar, quando as ondas EM são emitidas para o alvo e, em segundo lugar, durante o reflexo para o radar (fonte). Os radares comerciais disponíveis no mercado funcionam atualmente às frequências de 24 GHz (Gigahertz), 60 GHz, 77 GHz e 79 GHz. A propagação das ondas EM (radar) não é afetada por condições adversas meteorológicas nem pela iluminação do ambiente. Por conseguinte, podem operar de dia ou de noite em condições de nevoeiro, neve ou nebulosidade. Entre os inconvenientes dos sensores de radar estão a falsa deteção de objetos metálicos em redor do ambientes percebidos como sinalização rodoviária ou guarda-corpos e os desafios de distinguir objetos estáticos e estacionários (Wang et al., 2020b). Por exemplo, a diferença entre uma carcaça de animal (objeto estático) e a estrada pode representar um desafio para os radares resolverem devido à semelhança em Doppler shift. É essencial assegurar a precisão das posições e orientações de montagem de radares em produção, pois qualquer desalinhamento angular pode ter consequências fatais para o funcionamento do veículo, tais erros incluindo deteções falsas ou tardias de obstáculos em redor (Walling, 2017). As três principais categorias de sistemas de radar para automóveis são radar de médio alcance (MRR), o radar de longo alcance (LRR), e o radar de curto alcance (SRR). Os fabricantes de VA utilizam SRR para assistência de viagem e aviso de proximidade de colisão, MRR para sistema anti colisão lateral/traseiro e deteção de pontos cegos e LRR para aplicações de controlo de cruzeiro e deteção precoce (Shahian Jahromi et al., 2019). Em geral, os sensores de radar são um dos sensores mais conhecidos e são normalmente utilizados em VA para fornecer uma perceção fiável e precisa de obstáculos tanto de dia como na noite devido à sua capacidade de funcionar independentemente da iluminação e condições atmosféricas adversas. Fornece informação adicional, tal como a velocidade dos obstáculos em movimento e pode realizar cartografia em curto, médio ou longo alcance. O sensor do radar, no entanto, não é geralmente adequado para reconhecimento de objetos devido à sua baixa resolução, em comparação com câmaras. Por conseguinte, os desenvolvedores de VA frequentemente fundem a informação do radar com outros dados sensoriais, tais como câmara e LiDAR, para compensar as limitações dos sensores de radar.

2.2.3 *Lidar*

LiDAR é uma tecnologia de sensores que funciona com o princípio de emissão de impulsos de feixes infravermelhos ou luz laser que reflete objetos fora do alvo. Estas reflexões são detetadas pelo instrumento e o intervalo entre a emissão e a receção do pulso de luz permite estimar a distância. Como o LiDAR digitaliza o seu ambiente, gera uma representação 3D da cena na forma de uma nuvem de pontos (Wang et al., 2020a). As três variantes primárias dos sensores LiDAR que podem ser aplicados numa vasta gama de aplicações incluem LiDAR 1D, 2D e 3D. Os sensores LiDAR emitem dados como uma série de pontos, também conhecidos como dados de nuvem de pontos (*Point Cloud Data (PCD)*) em espaços 1D, 2D e 3D, mapeando desta forma a informação a nível da proximidade dos objetos ao radar. Para sensores LiDAR 3D, o PCD contém as coordenadas x, y, z. Para aplicações de VA, os sensores LiDAR com 64 ou 128 canais são normalmente utilizados para originar imagens laser (ou dados de nuvens de pontos) em alta resolução, segundo (KODORS, 2017).

- Os sensores 1D ou unidimensionais medem apenas a informação da distância (coordenadas x) de objetos nas imediações.
- Os sensores 2D ou bidimensionais fornecem informação adicional sobre o ângulo (coordenadas y) dos objetos detetados.
- Os sensores 3D ou tridimensionais disparam feixes laser no eixos vertical para medir a elevação (coordenadas z) dos objetos.

Os sensores LiDAR podem ainda ser categorizados como LiDAR mecânicos ou LiDAR de estado sólido ou Solid state Lidar (SSL). O LiDAR mecânico é a solução mais popular de digitalização de ambientes de longo alcance no campo da investigação e desenvolvimento de VA. Utiliza a ótica de alta qualidade e possui um sistema rotativo de lentes acionadas por um motor elétrico para dirigir os feixes laser e captar o campo desejado de vista (*Field of View (FoV)*) em torno do VA. As lentes rotativas podem alcançar uma cobertura horizontal de FoV de 360 graus no meio envolvente do veículo. Ao contrário, os *Solid State Lidar (SSL)* eliminam a utilização de lentes rotativas e, assim, evitam falhas mecânicas. Os SSL utilizam uma multiplicidade de micro estruturas de guias de ondas para direcionar os feixes laser para a perceção do meio envolvente. Estes LiDARs ganharam interesse nos últimos anos como uma alternativa aos LiDARs mecânicos devido à sua robustez, fiabilidade, e por geralmente estarem a custos mais baixos do que as suas contrapartes mecânicas. No entanto, eles têm um FoV horizontal menor, tipicamente 120° ou menos, do que o LiDAR mecânico tradicional. Os feixes laser refletidos são observações discretas registadas quando um pulso laser é detetado. LiDARs podem recolher múltiplos retornos a partir do mesmo laser e sensores modernos podem gravar até cinco retornos de cada pulso laser. Ao receber a reflexão de um laser de Lidar anteriormente emitido, o sensor analisa as luzes recebidas do feixe laser numa direção para determinar a distância e a intensidade da informação. Subsequentemente utiliza os dados obtidos para determinar qual foi o último retorno ou o retorno mais forte do feixe laser. Em contraste, os sensores em modo de configuração de duplo retorno devolverão tanto o sinal de retorno mais forte como as últimas medições, as mais tardias. No entanto, as medições de segunda mais forte regressarão como os mais fortes se as medições de retorno mais fortes forem as últimas medições de retorno. Em geral, atualmente, os LiDARs 3D são mais aplicados na condução de dia e de

noite, devido à sua maior abrangência, para proporcionar uma perceção fiável e precisa de um campo de visão, maior alcance de deteção e perceção de profundidade. Os dados adquiridos no formato de nuvem de pontos fornecem uma representação espacial 3D densa (ou "(imagem) laser") do ambiente à volta do VA. Os sensores LiDAR não fornecem informação a cores do meio envolvente em comparação com os sistemas de câmara e esta é uma das razões pelas quais o PCD é frequentemente fundido com dados de diferentes sensores usando algoritmos de fusão sensoriais, de acordo com (Yeong et al., 2021a).

2.2.4 *Sensores ultrassónicos*

Os sensores ultrassónicos são usados para medir distâncias curtas (alguns metros) e costumam ser usados para a tarefa de estacionamento, automático ou não. Funcionam segundo o princípio de reflexão de ondas ultrassónicas emitidas pelo sensor e refletidas por um alvo cuja frequência é superior à frequência máxima audível, 20 kHz.

2.2.5 *Sensores odométricos*

Os sensores odométricos são uma categoria de sensores usados para estimar a posição do veículo a que estão acoplados assim como a sua velocidade.

2.2.6 *Comparação entre os sensores*

É possível comparar as vantagens e desvantagens de cada sensor usado em várias dimensões:

- Medição de profundidade
- Resolução
- Capacidade de classificação de veículos automóveis
- Medição da velocidade
- Número de imagens por segundo
- Independência das condições meteorológicas
- Independência da hora do dia
- Robustez face a interferências

Como demonstra a figura 6, a combinação câmara mais radar é tão complementar que maximiza todas as dimensões, daí ser a fusão sensorial escolhida.

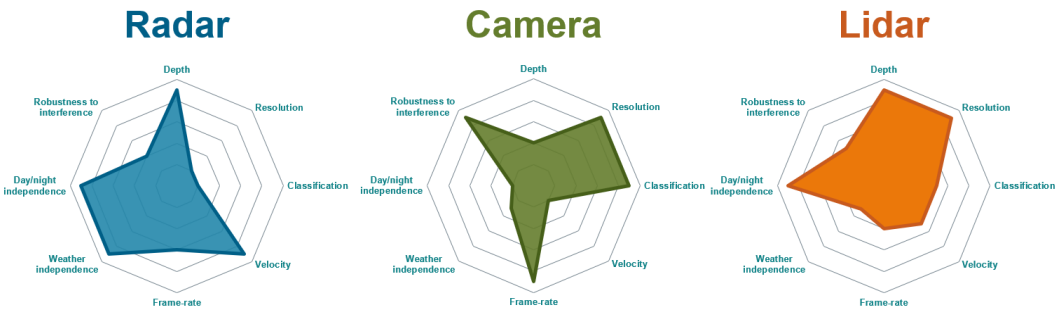


Figura 6: Comparação entre diferentes sensores. Tabela retirada de (Kaur, 2021).

2.3 Unidades de processamento a escolher para processamento das deteções dos sensores

Além de sensores, é necessário usar unidades para processamento de informação sensorial, bem como para emitir sinais de saída para os atuadores do veículo, como os travões, ou então para estes serem sinalizados em um monitor. A tabela 1 estabelece a relação entre as maiores empresas na indústria dos VA e os seus produtos.

Empresa	**Produtos para veículos autónomos**
Intel	Atom E3900 Processor Atom E3800 Processor Xeon processors Arria 10 FPGA Mobileye 5 Series Mobileys 6 Series Mobileye Shield+
Nvidia	NVIDIA DRIVE PX2 NVIDIA DGX-1 Xavier supercomputer chip
Qualcomm	Snapdragon 820A RFCMOS (radio frequency complementary metal-oxide-semicondutor) radar system QCA6696
AMD	Radeon Instinct MI6 accelerator
Xilinx	Zynq UltraScale+ MPSoC (multiprocessor system-on chip)
Tesla	FSD Chip

Tabela 1: Tabela onde estão mapeadas as principais empresas de desenvolvimento de unidades de processamento para VA, retirada de (far, 2014).

2.4 Fusão sensorial

Quanto à fusão de dados sensoriais, esta pode ser efetuada de três modos diferentes: fusão baixo nível, nível médio ou alto nível. Também possuem outra designação como a que está indicada na figura 7. No primeiro caso, os dados sensoriais são associados e fundidos antes de lhes serem extraídos as *features*.

Seguidamente, é efetuada uma atribuição de ID ao objeto classificado. No segundo caso, para cada sinal de cada sensor são extraídas as *features* e associadas a seguir para consequentemente serem fundidas e lhes ser atribuído um ID ao resultado da fusão. Por último, na fusão de alto nível, as *features* são uma vez mais extraídas em primeiro seguidas da atribuição de um ID e a sua associação é feita antes de ser efetuada uma fusão no nível final, de decisão. De notar que, apesar de não ser explícito na figura 7, costuma ser efetuado um pré-processamento do sinal proveniente dos sensores, nomeadamente para remoção/atenuação de ruído ou para efetuar uma amostragem do sinal. Tais operações permitem uma melhor deteção, ou pelo menos servem para que o sinal vá de encontro às especificações do algoritmo detetor, (Uusitalo, 2001).

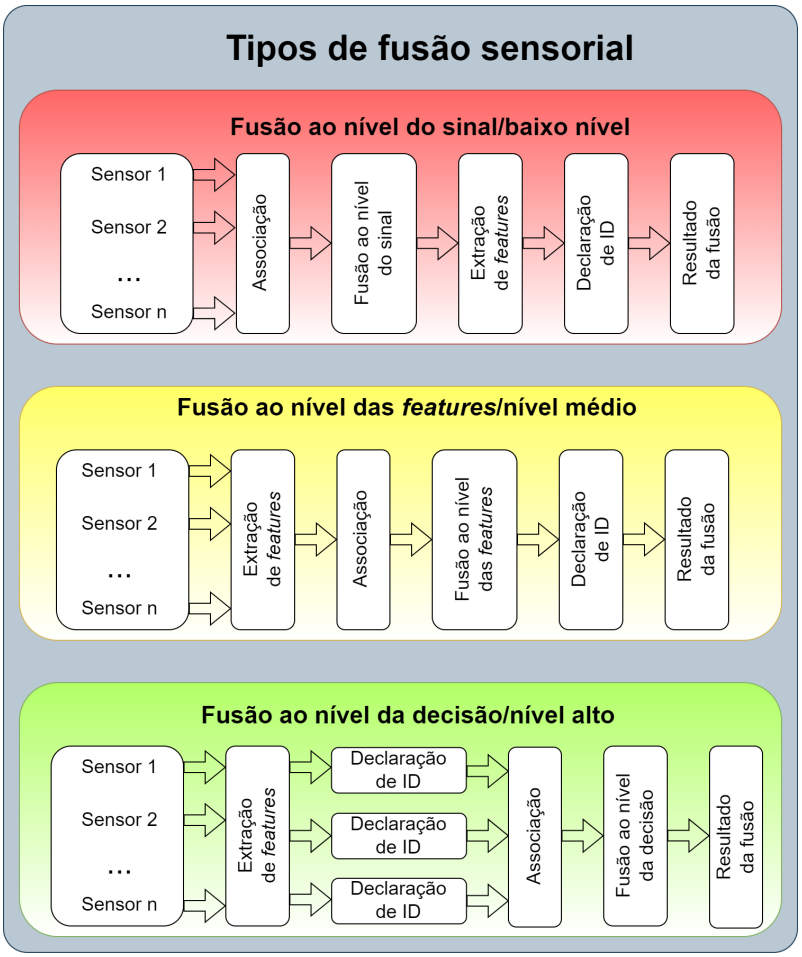


Figura 7: Tipos de fusão sensorial. A associação consiste na etapa da junção dos dados recolhidos por diferentes tipos de sensores.

Como se pode verificar na tabela 2, (Elmenreich, 2002), face às várias vantagens e desvantagens de cada modo de fusão sensorial, a **fusão de nível intermédio** é a mais indicada para o âmbito desta tese. A mesma técnica resulta de um compromisso entre a elevada sensibilidade na deteção obtida por métodos de fusão nível baixo e a flexibilidade de adicionar novas modalidades sensoriais ao modelo garantida por métodos de fusão nível alto, tendo em conta os benefícios da redundância já discutidos anteriormente.

	Vantagens	Desvantagens
Fusão de nível baixo	Fornece dados sensoriais com menor SNR, aumentando o desempenho. Reduz a latência ao atuar sobre os dados sem esperar pelo processamento sensorial, aumentando a performance.	Gera grandes quantidades de dados, podendo surgir problemas de memória ou largura de banda Existe redundância neste modo, levando a baixa eficiência de fusão de dados.
Fusão de nível intermédio	Requer menos poder computacional que fusão de baixo nível. Possibilita o uso de variados algoritmos de deteção de *features* e subconjuntos destes, aumentando a precisão de classificação.	Requer grandes conjunto de dados de treino para achar o subconjunto de *features* mais relevante. Requer calibração sensorial bastante precisa antes da extração e fusão de *features* de cada sensor
Fusão de nível alto	Baixo custo computacional, leve uso de recursos de comunicação e pouca complexidade. Permite abstração do algoritmo de fusão, não sendo necessário um conhecimento aprofundado dos algoritmos de processamento de sinal para a sua implementação.	Fornece classificações com confiança mais baixa que poderão se descartadas. Efetuar o fine-tuning do algoritmo não aumenta significativamente a precisão dos resultados nem diminui a latência do mesmo

Tabela 2: Vantagens e desvantagens dos vários tipos de fusão sensorial. Retirado de (Elmenreich, 2002).

2.5 Datasets

Na tabela 3 encontra-se representado as especificações de alguns dos conjunto de dados públicos mais conhecidos.

conjunto de dados	Ano	Radar	Nº deteções 3D	Nº classes	Localização
KITTI	2012	Não	80K	8	Germany
ApolloScape	2018	Não	70K	35	China 4x
H3D	2019	Não	1.1M	8	SF, California
nuScenes	2019	Sim	1.4M	23	Boston, SG
Argoverse	2019	Não	993K	15	PT, Miami
Waymo	2019	Não	12M	4	USA 3x
Boxy	2019	Não	2M	1	SF, California
AIODrive	2021	Sim	26M	23	Sintético

Tabela 3: Alguns conjunto de dados usados no âmbito dos VA. Tabela adaptada de (Wang et al., 2021b). Por ordem, vem: KITTI (Geiger et al., 2012), ApolloScape (Huang, 2019), H3D (Patil, 2019), nuScenes (Caesar et al., 2020), Argoverse (Wilson, 2023), Waymo (Sun, 2020), Boxy (Behrendt, 2019) e AIODrive (Weng et al., 2020).

Dos conjunto de dados apresentados, o nuScenes, (Caesar, 2020),é o mais indicado para este projeto, sendo composto por uma ampla gama de imagens *Reg Green Blue (RGB)* e de formatos PCD de radar retirados de cenários reais. O conjunto de dados KITTI, (Geiger, 2012b), também é excelente para efetuar

testes no âmbito da condução autónoma, visto que possui milhares de ficheiros de dados reais para câmara, 2D e 3D, e lidar, apesar de não possuir conjunto de dados relativos a radar. Note-se que apesar de os conjunto de dados reais serem os mais fidedignos, muitas vezes recorre-se ao uso de diferentes conjunto de dados sintéticos através de simuladores visto que as imagens sintéticas podem ser geradas em massa, permitindo assim ter um maior conjunto de dados de treino para treinar as redes neuronais. Consequentemente, é possível obter-se melhores resultados como demonstrado em (Johnson-Roberson et al., 2016). Deste modo, uma alternativa seria o uso do conjunto de dados AIODrive, (Weng, 2020), ou então, o uso de simuladores para captação de vídeo e, consequentemente, extração e processamento de imagens para criação de um conjunto de dados *ad hoc*.

2.6 Deteção de objetos

A operação de deteção de objetos envolve localização e classificação destes no campo de visão de um ou mais sensores.

Especificamente, a tarefa de classificação, que consiste na atribuição de um rótulo a uma imagem de entrada dentro de um leque de categorias, constitui um dos principais problemas de visão por computador e possui diversas aplicações práticas. No entanto, existem alguns problemas que muitas vezes dificultam a tarefa de classificação, nomeadamente:

- **Variação do ponto de vista** — um objeto pode ser orientado de várias maneiras em relação à câmara.
- **Variação de escala** — Um objeto pode assumir escalas enganosas numa imagem, o que pode dificultar a classificação.
- **Deformação** — Muitos objetos de interesse não são corpos rígidos e podem ser deformados, o que se traduz numa elevada dificuldade em classificar tais objetos.
- **Oclusão** — Os objetos de interesse podem estar ocluídos. Às vezes, apenas uma pequena parte de um objeto (com apenas alguns pixels) se encontra visível.
- **Condições de iluminação** — Os efeitos da iluminação afetam os valores dos píxeis que compõe a imagem dos objetos de interesse, quando se utiliza a câmara.

O objetivo do processo de classificação de uma imagem pode ser visto na Figura **??**, onde uma imagem com anotações entra numa rede neuronal ajustada para a aplicação em causa e a rede aprende a detetar os atributos presentes no objeto de interesse nessa mesma imagem. Ao fim dessa aprendizagem, pretende-se que a rede consiga reconhecer todo o tipo de imagens relacionadas com o objeto de interesse em questão, partindo de ficheiros de imagem desse objeto que não estão presentes no conjunto de dados de treino e validação. Na secção abaixo encontra-se alguns exemplos de redes neuronais tipicamente usadas para deteção de imagens.

2.6.1 *CNN: Redes neuronais de convolução*

Quando se fala em visão por computador, uma das técnicas de aprendizagem profunda mais famosa consiste no uso de redes CNN. As CNNs tornaram-se no principal método para resolver desafios que envolvam dados de imagens. Quando nos deparamos com imagens de grandes resoluções, o custo computacional e a consequente ocupação de memória está inerente, o uso de CNN consegue contornar estas desvantagens e originar bons resultados a este nível. As CNNs são uma das principais categorias na tarefa de reconhecimento e classificação de imagens. As classificações de imagem de uma rede CNN utilizam numa imagem de entrada, processam-na e procedem à sua classificação em determinadas categorias. O computador "vê"uma imagem como uma matriz de píxeis, esta matriz contém uma altura, largura e profundidade. Por exemplo, uma imagem de matriz 6x6x3 (em que 6x6 diz respeito à sua dimensão e o 3 refere-se aos três canais presentes em RGB) e uma imagem de matriz 4x4x1, em que o 1 significa um canal de luminosidade. Nos modelos de CNN, cada imagem passa por uma série de camadas de convolução com filtros (Kernels), Pooling, Camadas Totalmente Interligadas (CTI) e a que cada é aplicada a função Softmax para classificar um objeto com valores probabilísticos entre 0 e 1. A representação do fluxo completo numa CNN está representado na figura 8, (Prabhu, 2018).

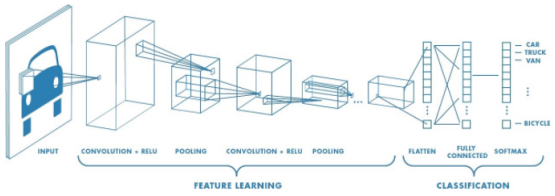


Figura 8: Representação do fluxo completo numa CNN para processar uma imagem de entrada e classificar os objetos de saída.

A etapa da convolução ocorre na primeira camada a extrair recursos de uma imagem de entrada. Esta camada, denominada de camada de convolução preserva o relacionamento entre os píxeis, e traduz-se numa operação matemática que recebe duas entradas: sendo uma a matriz que representa a imagem e a outra um filtro (*Kernel*).

Ao efetuar uma operação de convolução de uma imagem sendo *a* altura, *l* largura e *d* dimensão com um filtro de *fa* altura, *fl* largura e *fd* dimensão, obtém-se na saída o tamanho:

$$(a \times l \times d) * (fa \times fl \times fd) = (a - fa + 1) \times (l - fl + 1) \times 1 \tag{3}$$

O *Stride* corresponde à deslocação em píxeis efetuada pelo filtro por cada iteração sobre a matriz de entrada. Quando o filtro não se encaixa perfeitamente na imagem de entrada, existem duas formas de contornar o processo, a primeira é impor zeros na imagem e a segunda é descartar a parte da imagem onde o filtro não coube. Quando se aumenta uma imagem de 5x5x1 em uma imagem 6x6x1 e, em seguida, se aplica um Kernel 3x3x1 sobre a imagem aumentada, observa-se que a matriz resultante apresenta dimensões 5x5x1. Daí o nome *same padding*. Por outro lado, se a mesma operação for realizada mas desta vez sem padding, a matriz resultante apresenta as dimensões do próprio Kernel (3x3x1), denominando-se assim *valid padding*. Outro conceito importante é o de *Pooling Layer*, a sua função é reduzir o tamanho espacial do recurso

envolvido (redução dimensional) e, desta forma, reduz o poder computacional necessário para processar os dados. A Pooling Layer é muito útil na extração de características dominantes que não variam tornando assim o processo igualmente eficaz. Alguns exemplos de *Pooling*: *Max Pooling* e *Average Pooling*; na figura 9 os processos estão explicados de uma forma gráfica. Max Pooling retorna o valor máximo da imagem coberta pelo Kernel, por outro lado, o Average Pooling retorna a média de todos os valores da parte da imagem coberta pelo Kernel.

- Max Pooling - funciona como um supressor de ruído caso os seus valores sejam mais baixos que os do sinal, pois descarta completamente as ativações ruidosas e realiza a remoção do ruído junto com a redução da dimensão.
- Average Pooling - realiza apenas uma redução da dimensão.

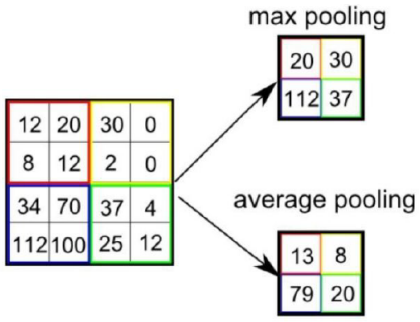


Figura 9: Alguns tipos de operações de pooling. Imagem retirada de (Saha, 2018).

ReLU significa Unidade Linear Retificada para uma operação não linear. A saída da função é dada por:

$$f(x) = max(0, x) \tag{4}$$

O objetivo da ReLU é introduzir a não linearidade. Existem outras funções não lineares não tão populares, como tanh ou sigmóide, que também podem ser usadas no lugar da ReLU. Fully Connected Layer (FCL) é outro conceito importante em redes neuronais. Estamos perante uma FCL quanto as entradas de uma camada são conectadas a todas as unidades de activação da camada seguinte. Nos modelos mais antigos de ML, as últimas camadas são FCL que compilam os dados extraídos pelas camadas anteriores formando assim a saída final, esquema 10.

Concluindo, uma CNN é um algoritmo de aprendizagem profunda que pode reconhecer e classificar certas caraterísticas de uma imagem. É uma rede neuronal de várias camadas, projetada para analisar entradas visuais e executar tarefas como classificação de imagens, segmentação e deteção de objetos, que podem ser úteis na condução autónoma. Uma CNN é composta por duas partes: a convolução que divide as várias *features* da imagem para análise, e uma FCL que usa a saída da camada de convolução para prever a melhor descrição para a imagem. A arquitetura da CNN é inspirada na organização e na funcionalidade do córtex visual e projetada para imitar o padrão de conectividade dos neurónios no cérebro humano. Os neurónios dentro de uma CNN são divididos numa estrutura tridimensional, em que cada conjunto de neurónios analisa uma pequena região ou característica da imagem. Ou seja, cada grupo de neurónios é especializado em identificar uma parte da imagem. As CNNs usam as previsões das camadas para produzir uma saída final

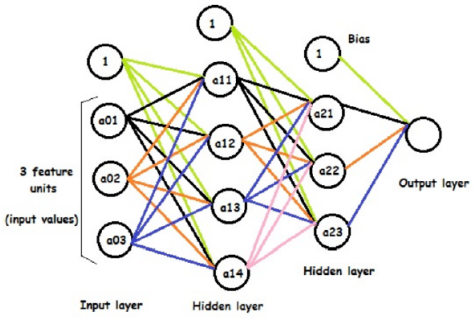


Figura 10: Arquitetura de uma rede de convolução superficial com uma camada de entrada, duas escondidas e uma de saída. Imagem retirada de (Singh, 2017).

que apresenta um vetor de pontuações de probabilidade que representa a probabilidade de uma amostra específica pertencer a uma determinada classe, (Dixe, 2020). Na Figura 11 está representada a arquitetura de uma CNN. Em suma, uma CNN é composta pelas seguintes camadas, além da camada convolucional já explicada:

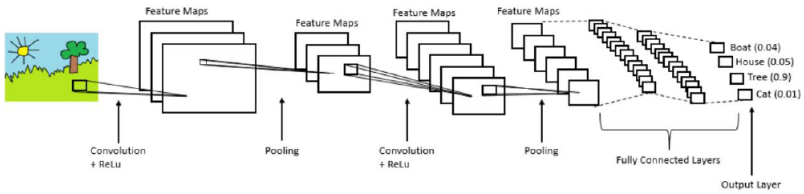


Figura 11: Etapas presentes numa arquitetura completa de CNN. Imagem retirada de (Saha, 2018).

- **Camada de pool (downsampling)** - reduz a quantidade de informações que a camada convolucional gerou para cada *feature* mantendo as informações essenciais (o processo das camadas convolucionais e de pool geralmente repetem-se várias vezes).
- **Fully connected input layer** - "nivela" as saídas geradas pelas camadas anteriores transformando-as num vetor único que pode ser usado como entrada para a próxima camada.
- **Fully connected layer** - aplica pesos sobre a entrada gerada pela análise de recursos para prever uma anotação precisa.
- **Fully connected output layer** - gera as probabilidades finais para determinar uma classe para a imagem.

2.6.2 *Arquiteturas de CNNs*

Existem várias arquiteturas de *Convolutional Neural Networks (CNN)*s disponíveis, ver Figura 12, como: LeNet-5, AlexNet, VGG-16, Inception-v1, Inception-v3, ResNet-50, Xception, Inception-v4, Resception-ResNets e ResNeXt-50. Estas redes tornaram-se tão profundas e complexas que se tornou extremamente difícil visualizar o modelo inteiro. A arquitetura de uma CNN é o fator chave para determinar o seu desempenho e eficiência. A forma como as camadas estão estruturadas, quais elementos são usados em cada camada

e como estes são projetados afetam frequentemente a velocidade e a precisão com que ela pode executar várias tarefas.

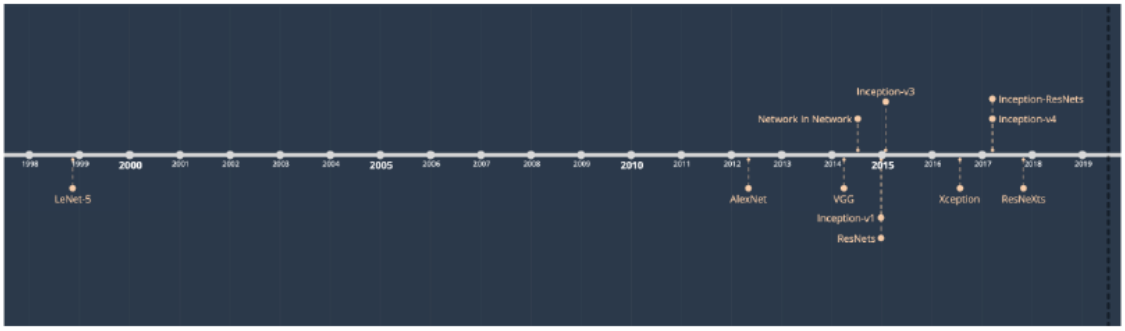


Figura 12: Evolução cronológica do aparecimento de novas arquiteturas CNN. Imagem adaptada de (Karim, 2019).

Existem várias arquiteturas populares de CNN, ver Figura 12, muitas delas ganharam reconhecimento ao obter bons resultados no *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC), segundo a plataforma de *Deep Learning (DL)* (Shenhav).

- LeNet-5 (1998) - É uma arquitetura CNN de 7 camadas, ver Figura 13, que serviu em 1998 para classificar dígitos a partir de imagens através de entrada digitalizadas em escala de cinza de 32 × 32 píxeis. Esta arquitetura foi usada por vários bancos para reconhecer os números escritos à mão nos cheques.

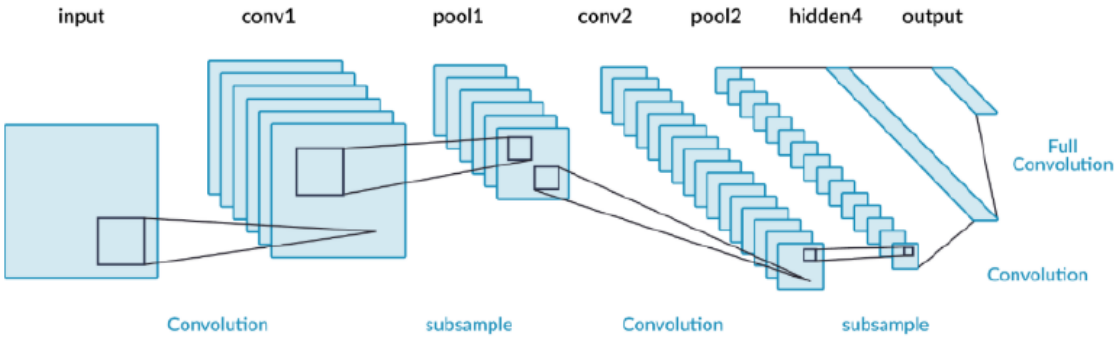


Figura 13: Arquitectura LeNet-5. Imagem adaptada de (Shenhav).

- AlexNet (2012) - A AlexNet é uma arquitetura projetada pelo grupo *SuperVision*, sendo esta semelhante à LeNet, mas mais profunda (Figura 14). Esta arquitetura apresenta mais filtros por camada, além de camadas convolucionais empilhadas. É composta por cinco camadas convolucionais, seguidas por três *fully connected layers*.
  Uma das diferenças mais significativas entre o AlexNet e outros algoritmos de classificação de objetos é o uso da ReLU para a parte não linear em vez do uso de funções Sigmond ou Tanh muito usadas em redes neuronais tradicionais. AlexNet aproveita o treino mais rápido da ReLU para tornar o seu algoritmo também mais rápido.

Os criadores da AlexNet dividiram a sua rede em dois *pipelines* porque usaram duas GPUs (Nvidia Geforce GTX 580 Graphics Processing Units) para treinar esta CNN.

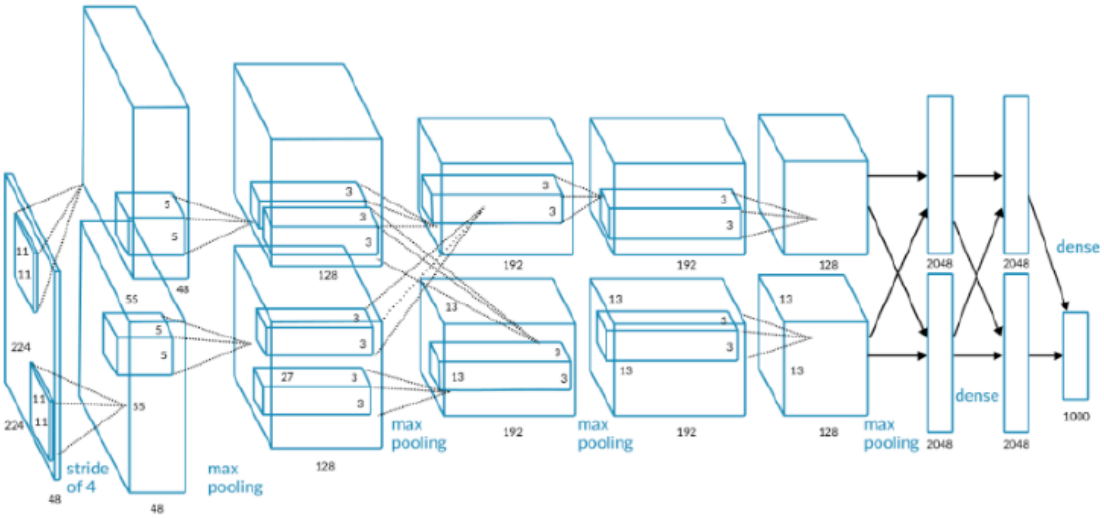


Figura 14: Arquitetura AlexNet. Imagem adaptada de (Shenhav).

- GoogleNet (2014) - É uma criação inspirada na LetNet, esta rede também denominada de Inception V1, foi criada pela Google, ver esquema na Figura 19. O GoogleNet foi o vencedor do ILSVRC 2014 e alcançou uma taxa de erro mínima de 7%, percentagem próxima do nível de desempenho humano.
  A arquitetura do GoogleNet consiste numa CNN de 22 camadas de profundidade que usa um módulo baseado em pequenas convoluções, chamado "inception module", que usa batch normalization (técnica de aprendizagem supervisionada que converte as saídas de camadas num formato padrão chamado normalização) e RMSprop (algoritmo de otimização para treino de redes neuronais baseado na propagação do erro médio quadrado).

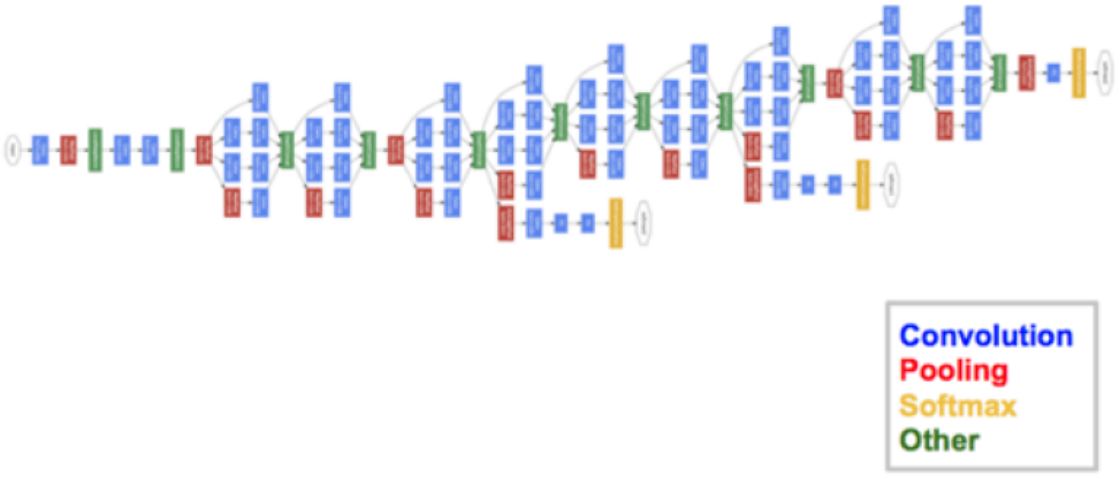


Figura 15: Arquitectura GoogleNet. Imagem adaptada de (Shenhav).

- VGGNet (2014) - A VGGNet, foi a segunda rede com melhor desempenho apresentada no ILSVRC 2014, e possui 16 camadas convolucionais, ver Figura 16. A VGGNet foi treinada em 4 GPUs por mais de

duas semanas para alcançar o seu desempenho. O problema com o VGGNet são os 138 milhões de parâmetros, 34.5 vezes mais que o GoogleNet, o que dificulta a sua execução.

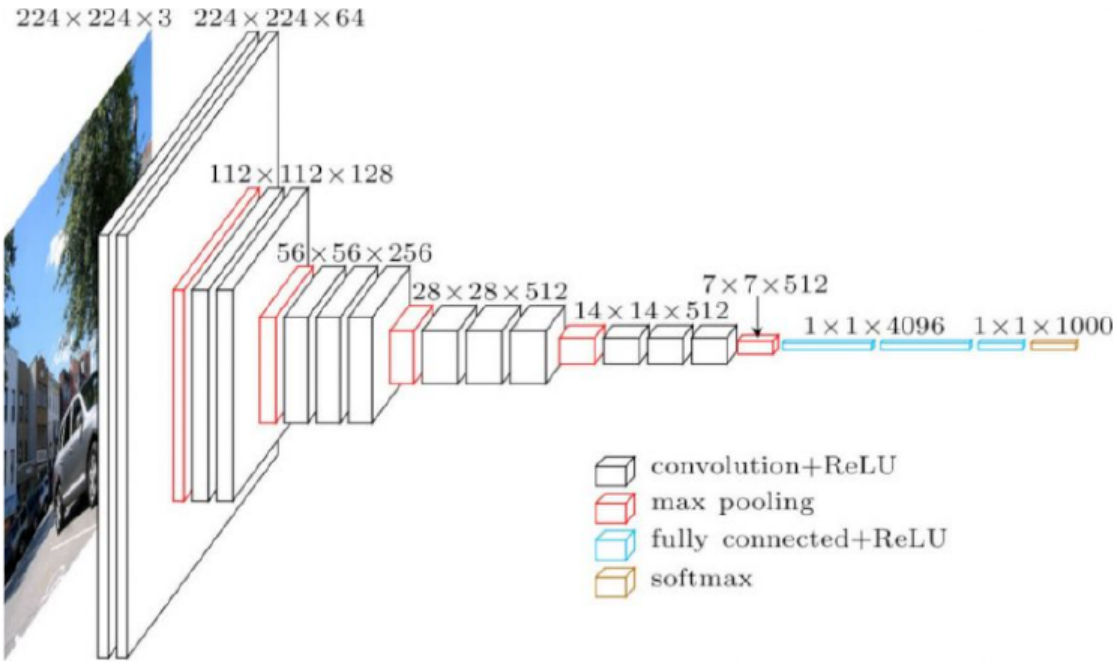


Figura 16: Arquitectura VGGNet. Imagem adaptada de (ul Hassan, 2018).

2.6.3 *Métricas*

Num qualquer modelo de ML ou DL existe a necessidade de "medir"a qualidade do modelo de acordo com o objetivo da tarefa a que se propõe. Com a necessidade de avaliar estes modelos foram utilizadas funções matemáticas que ajudam na avaliação da capacidade de erro e no acerto dos modelos. Estas funções matemáticas denominam-se de métricas.

A importância de escolher o melhor modelo de ML ou DL para uma dada tarefa é indiscutível. Mas, é a partir da escolha da métrica apropriada ao problema que conseguimos avaliar o desempenho dos modelos e consequentemente compará-los entre si.

Existem vários tipos de métricas, umas mais simples outras mais complexas, umas comportam-se melhor consoante determinadas características de *conjunto de dados*, outras são desenvolvidas consoante o objetivo dos modelos. A escolha da métrica mais acertada pode fazer a diferença na hora da avaliação do modelo, por isso, devem-se considerar fatores como proporção de dados nos *conjunto de dados* e o objetivo da previsão como: probabilidade, binário, *ranking*, etc.

O conceito de métrica perfeita não existe e nem deve existir essa comparação, todas elas são igualmente relevantes, depende unicamente da aplicação prática do modelo em que são usadas. Em certas situações especificas o ideal é que a métrica usada tenha um significado especifico para a tarefa em questão, por exemplo imagine-se a situação de sites de anúncios, uma boa métrica é a taxa de cliques, com isto, é importante interiorizar que as métricas a seguir apresentadas podem ser usadas na generalidade mas se numa situação especifica se conseguir uma métrica diretamente relacionada com o contexto, é essa que deve ser usada.

As métricas a seguir apresentadas, são utilizadas em tarefas de classificação. A tarefa de classificação é procurar prever qual é a categoria a que uma dada amostra pertence. A maioria das métricas pode ser usada tanto na tarefa de classificação binária como em classificação de várias classes.

Matriz de confusão

A matriz de confusão é uma medida de desempenho para a classificação de ML em que a saída pode ser duas ou mais classes. É uma tabela com 4 combinações diferentes de valores previstos e reais, se estes valores forem binários, ver Tabela 4.

Tabela 4: Matriz Confusão.

		Valor Verdadeiro	
Valor Previsto		Positivos	Negativos
	Positivos	**VP** **Verdadeiro Positivo**	**FP** **Falso Positivo**
	Negativos	**FN** **Falso Negativo**	**VN** **Verdadeiro Negativo**

1. Verdadeiro Positivo (VP): Valores que pertencem à classe positiva e foram classificados como positivos. Classificados corretamente.
2. Falso Positivo (FP): Valores que pertencem à classe negativa e foram classificados como positivos.
3. Falso Negativo (FN): Valores que pertencem à classe positiva e foram classificados como negativos.
4. Verdadeiro Negativo (VN): Valores que pertencem à classe negativa e foram classificados como negativos. Classificados corretamente.

As métricas a seguir apresentadas são amplamente usadas na comparação de modelos de classificação. Cada uma das seguintes métricas avalia um aspeto diferente do modelo. Para perceber as métricas é importante ter em mente a Matriz de Confusão apresentada anteriormente na tabela 4.

1. *Exatidão*: Corresponde à proporção de casos verdadeiros (Verdadeiro Positivo e Verdadeiro Negativo) entre o número total de casos examinados.

$$Accuracy = \frac{VerdadeiroPositivo(VP) + VerdadeiroNegativo(VN)}{Total} \tag{5}$$

2. *Precisão*: Número de exemplos classificados como pertencentes a uma classe, que realmente são daquela classe (Verdadeiros Positivos), dividido pela soma entre este número, e o número de exemplos classificados nesta classe, mas que pertencem a outras (Falsos Positivos).

$$Precision = \frac{VerdadeirosPositivos(VP)}{VerdadeirosPositivos(VP) + FalsosPositivos(FP)} \tag{6}$$

3. *Revocação*: Frequência com que o classificador encontra os exemplos de uma classe, ou seja, quando realmente é da classe x, o quanto frequentemente é classificado como x:

$$Recall = \frac{VerdadeiroPositivo(VP)}{VerdadeiroPositivo(VP) + FalsoNegativo(FN)} \tag{7}$$

4. *Pontuação F1*: Métrica que combina *precision* e *recall* de modo a traduzir num único valor a qualidade geral do modelo, esta métrica trabalha bem com conjuntos de dados que possuam classes desproporcionais. Quanto maior o valor, melhor o modelo.

$$F1 = \frac{2 * precision * recall}{precision + recall} \tag{8}$$

5. *Intersection over Union* (IoU): No problema de deteção de objetos esta é uma métrica muito utilizada e bastante eficiente na análise do problema. Na Figura 17, está ilustrada a equação que traduz a métrica IoU, este cálculo consiste em dividir a área de interceção entre a localização do objeto e a previsão deste pela área de união (objeto e previsão). Na Figura 18, encontram-se exemplos de diferentes valores de IoU.

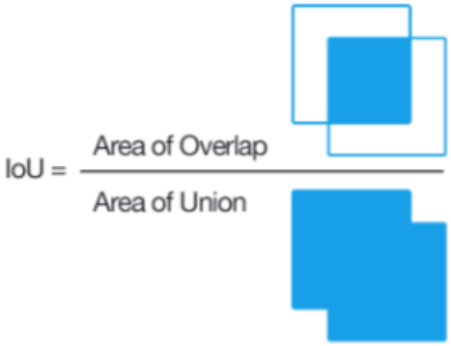


Figura 17: Representação da equação que traduz o valor de IoU. Imagem adaptada de (Rosebrock, 2016).

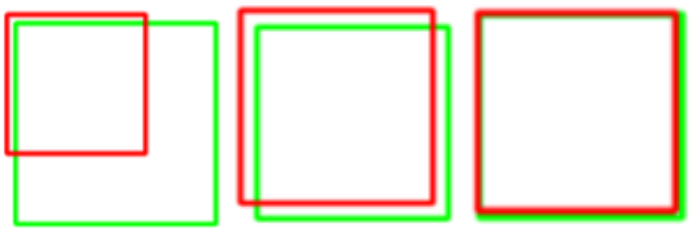


Figura 18: (a) Valor de IoU = 0.4034 (Mau), (b) Valor de IoU = 0.7330 (Bom) (c) Valor de IoU = 0.9264 (Excelente). Imagem adaptada de (Rosebrock, 2016).

6. *Mean Average Precision(mAP)*: Métrica usada para comparação entre diferentes modelos de *machine learning* que corresponde à média de todas as áreas resultantes das curvas de precisão e revocação, para cada categoria capaz de ser classificada pelo modelo. Os valores de precisão e revocação tabelados são cumulados numa tabela à medida que aumenta a dimensão da amostra, de 1

a N, sendo N a nº total de elementos do conjunto de dados. Esses valores são interpolados, criando uma curva de precisão-revocação cuja área corresponde à métrica em questão para uma dada classe, ou AP (precisão média). Obtém-se a mAP quando se efetua a média para todas as APs das classes classificadas. Quanto maior este parâmetro, melhor.

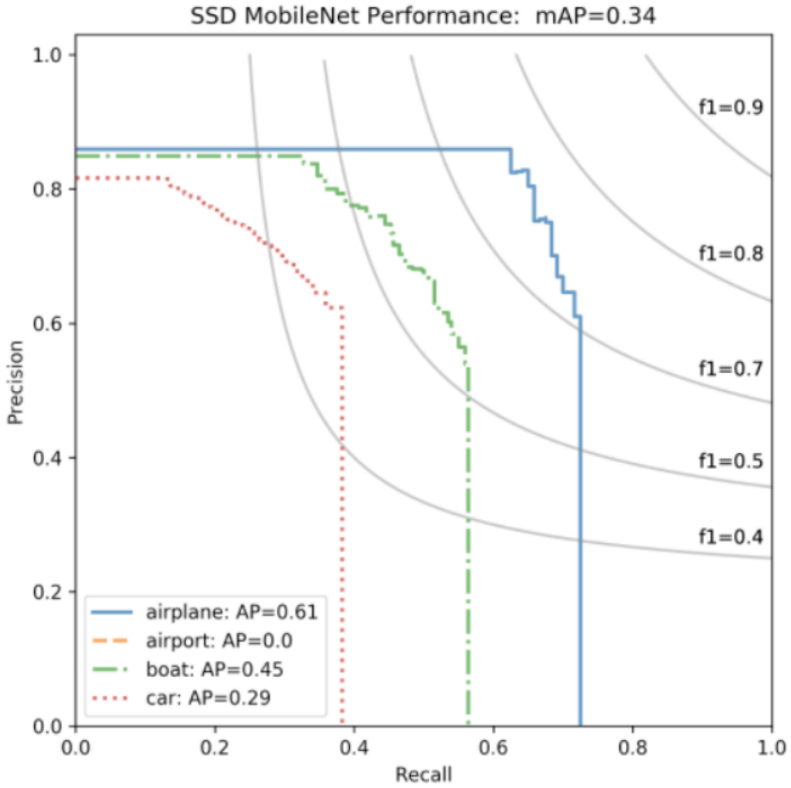


Figura 19: *Mean Average Precision* de uma rede SSD. Imagem adaptada de (Yohanandan and Pokhrel, 2020).

7. Outras métricas: Poderão ser usadas também outras métricas, sobretudo métricas de erro para diferenciação entre diferentes algoritmos com similar mAP, ou *mean Average Precision*. Idealmente, pretende-se que os seguintes parâmetros sejam 0% ou o mais baixo possível.

    - mATE - *mean average translation error*, calcula a média do erro de translação entre as predições e as caixas de *ground truth*.
    - mASE - *mean average scale error*, calcula a média do erro de escala entre as predições e as anotações reais.
    - mAOE - *mean average orientation error*, calcula a média do erro da orientação entre as predições e as caixas de *ground truth*.
    - mAVE - *mean average velocity error*, calcula a média do erro da velocidade entre as predições e as caixas de *ground truth*.
    - mAAE - *mean average attribute error*, calcula a média do erro de atribuição da correta classificação entre as predições e as caixas de *ground truth*.

2.7 Algoritmos de deteção

Alguns dos algoritmos de deteção de objetos 3D e/ou em tempo real que costumam ser usados em técnicas de fusão sensorial encontram-se detalhados na tabela 5 abaixo.

Algoritmo	Descrição	Vantagens	Desvantagens
YOLO	Detetor de apenas um estágio que efetua a deteção/deteções apenas com uma passagem de uma imagem numa CNN	Deteções em tempo real Versões mais recentes tem maior mAP que a alternativa SSD	Deteção fraca de pequenos objetos ou de conjuntos destes aglomerados
SSD	Detetor de um estágio similar ao YOLO que discretiza as deteções em caixas de dimensões variáveis às dimensões dos obstáculos detetados	Deteções em tempo real Maior rapidez de inferência que a alternativa YOLO	A rapidez vem com custos na precisão, em comparação com detetores de duplo estágio
VoxelNet	Detetor 3D end-to-end que processa de dados que recebe em formato de nuvem	Não precisa de extrair *features* manualmente	Precisa de grandes quantidades volume de dados de treino e memória
PointNet	Uma rede invariável à permutação de dados que aprende *features* de nuvens de dados em dois estágios de deteção	Lida com nuvens de pontos em qualquer ordem	Dificuldade em generalizar para configurações nunca antes vistas

Tabela 5: Comparação entre diferentes algoritmos de deteção que costumam ser usados no âmbito da fusão sensorial. Tabela adaptada de (Yeong et al., 2021b). Os dois primeiros algoritmos, o Yolo (Redmon, 2016) e o SSD (Liu and Anguelov, 2016) funcionam apenas com imagens de câmara. Já os dois últimos algoritmos, o VoxelNet (Zhou and Tuzel, 2017) e o PointNet (Qi, 2017) funcionam apenas com nuvens de pontos para lidar.

2.7.1 *Detetores que recorrem ao uso de câmaras e radares*

Poucos estudos focam-se na fusão de radares com outros sensores para aplicações de condução autónoma. RadarNet (Yang et al., 2020) combina dados de radar e LiDAR para deteção de objetos 3D. Usa um mecanismo de fusão de baixo nível para aprender representações conjuntas dos dois sensores e um mecanismo de fusão de alto nível para explorar a evidência da velocidade radial do radar e melhorar a estimativa de velocidade do objeto, figura 20. Em (Chadwick et al., 2019), são efetuadas projeções de radar para o plano da imagem a fim de aumentar a precisão de deteção de objetos para objetos distantes. Em (Nabati and Qi, 2020) autores usam deteções de radar para originar propostas de objetos 3D primeiro e, em seguida, projeções no plano da imagem para realizar a junção de deteção de objetos 2D e estimativa de profundidade. CRF-Net (Nobis et al., 2019) também projeta deteções de radar para o plano da imagem, mas representa como linhas verticais onde os valores de pixel correspondem à profundidade de cada ponto de deteção. Os dados da imagem são então aumentados com as informações do radar e usado uma rede de convolução para realizar deteção de objetos 2D.

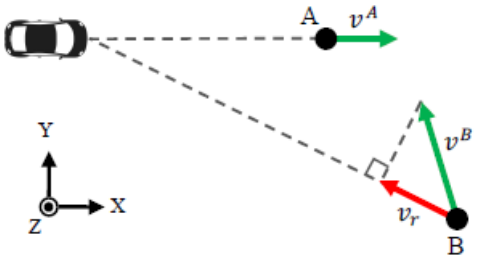


Figura 20: Diferença entre velocidade atual e radial. No caso A, são iguais; no caso B, a velocidade atual encontra-se a verde e a radial a vermelho, em direção ao ego veículo, (Nabati and Qi, 2020).

Face a estes métodos, pretende-se deste modo efetuar uma abordagem do estado de arte para a temática de detetores 3D, começando pelos preliminares, isto é, por introduzir a noção da nuvem de pontos do radar e pelo algoritmo de deteção 3D, Centernet, (Zhou et al., 2019).

2.7.2 *Nuvem de pontos de radar*

Os radares retornam geralmente os objetos detetados como pontos 2D em *Birds' Eye View (BEV)*, fornecendo o ângulo de azimute e a distância radial para o objeto. Para cada deteção, o radar também retorna a velocidade instantânea do objeto na direção radial. Primeiro, representa-se cada deteção de radar como um ponto 3D no sistema de coordenadas egocêntrico, e o mesmo é parametrizado como P = (x; y; z; vx; vy) onde (x; y; z) é a posição e (vx; vy) é a velocidade radial relatada do objeto nas direções x e y. A velocidade radial é compensada pelo movimento do ego veículo. Para cada cena, agrega-se 3 amostragens da nuvem de pontos de radar (deteções nos últimos 0,25 segundos). O conjunto de dados nuScenes fornece os parâmetros de calibração necessários para mapear as nuvens de pontos de radar do sistema de coordenadas do mesmo para as coordenadas do ego veículo e a sua câmara.

2.7.3 *CenterNet*

O Centernet (Zhou, 2019) corresponde ao estado de arte de deteção de objetos usando uma única câmara. Como entrada recebe uma imagem, I, de dimensões, $WxHx3$, e cria um mapa de pontos de interesse, que varia entre [0,1] e possui dimensões $\frac{W}{R}x\frac{H}{R}xC$ onde W e H correspondem à largura e altura da imagem, R à razão de decimação e C ao número de categorias de objetos. Assim, uma predição onde um ponto no mapa de pontos de interesse (descritores) é 1, significa que existe um objeto detetado centrado nas coordenadas desse ponto e pertence a uma classe específica também descrita por esse descritor. Também é efetuado o mesmo processo para as ground-truth onde um mapa de descritores é originado. Sabendo que para cada centro de cada caixa de deteção $p_i \in \Re^2$ de categoria c, o valor que existe em cada ponto $q \in \Re^2$ é definido como:

$$Y_{qc} = max_i[e^{-\frac{(p_i-q)^2}{2\sigma_i^2}}] \tag{9}$$

onde σ_i corresponde ao desvio-padrão adaptado ao tamanho de cada objeto detetado.

Para originar caixas de deteção 3D são usadas camadas do topo de uma rede para achar a profundidade, dimensões e orientação diretamente dos pontos de centro.

2.7.4 *Centerfusion*

Tendo em conta o algoritmo anterior, o Centerfusion (Nabati, 2020) é proposto um mecanismo de fusão intermédio que associa deteções de radar para com o centro do seu objeto correspondente e explora tanto o radar como as *features* de imagem para melhorar as deteções preliminares, reavaliando a sua profundidade, velocidade, rotação e atributos. A chave deste mecanismo de fusão é a associação exata de deteções de radar a objetos. A rede de deteção do ponto central de objetos gera um mapa térmico para cada categoria de objeto na imagem. Os picos no mapa de calor representam possíveis pontos centrais dos objetos, e as *features* da imagem nesses são utilizados para estimar outras propriedades dos objetos. Para explorar a informação do radar neste cenário, *features* baseadas no radar precisam de ser mapeadas para o centro do seu correspondente objeto na imagem, o que requer uma associação precisa entre as deteções de radar e os objetos em cena.

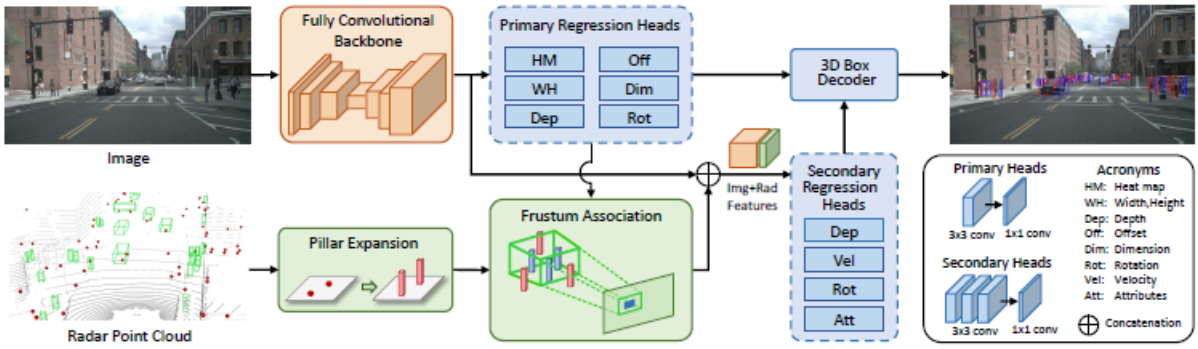


Figura 21: Pipeline do Centerfusion. Da imagem são obtidas as caixas de deteção 3D através da componente *fully convolutional backbone* que extrai as *features*. A associação do tronco da caixa de deteção com as deteções do radar, expandidas em pilares, permite originar *features* conjuntas da câmara e radar. Estas são concatenadas com as deteções preliminares para recalcular a profundidade e rotação, assim como estimar a velocidade e outros atributos dos objetos detetados. Imagem retirada de (Nabati and Qi, 2020).

Deteção do centroide da imagem

Com o CenterNet gera-se as deteções preliminares para a arquitetura do Centerfusion, exemplificada na figura 21. As *features* são primeiramente extraídas por uma rede convolucional codificadora-descodificadora, *backbone* do tipo *Deep Layer Aggregation (DLA)*, (Yu et al., 2017). As *features* permitem obter as coordenadas do centro (centróide) assim como as dimensões 2D do objeto, profundidade, rotação e dimensões 3D do mesmo. Cada cabeça primária de regressão possui uma camada de convolução 3x3 com 256 canais e uma camada de convolução 1x1 para originar a saída desejada, obtendo-se assim uma caixa de deteção 2D e uma preliminar 3D para cada objeto no cenário.

Associação ao radar

A rede de deteção de pontos centrais utiliza apenas as *features* da imagem no centro de cada objeto para regressão de todas as outras propriedades destes. Para explorar plenamente os dados do radar neste processo, precisamos primeiro de associar as suas deteções aos seus objetos no plano da imagem. Para realizar isto, uma abordagem pouco cautelosa seria o mapeamento de cada deteção de radar para o plano da imagem e associá-lo a um objeto se o ponto cartografado ficasse dentro da caixa de delimitação 2D desse objeto. Esta não é uma solução muito robusta, visto que não existe um cartografia um-a-um entre deteções de radar e objetos na imagem; muitos objetos na cena geram múltiplas deteções de radar, e há também deteções de radar que não correspondem a nenhum objeto. Além disso, porque a dimensão z da deteção do radar não é exata (ou não existe de todo), a deteção do radar mapeada pode acabar fora da caixa de delimitação 2D do seu objeto correspondente. Além disso, deteções de radar obtidas a partir de objetos obstruídos mapeariam para a mesma área geral na imagem, o que dificulta a sua diferenciação no plano da imagem 2D.

Mecanismo de associação de frustum

É um método de associação que utiliza a caixa de deteção 2D assim como a sua profundidade e tamanho estimado para criar uma caixa 3D, ou seja, uma Região de Interesse (RoI) chamada frustum do objeto. A partir de uma caixa de delimitação 2D para cada alvo, cria-se um frustum para esse objeto, como mostra na Fig. 3. Isto permite reduzir as deteções de radar que precisam de ser verificadas para associação, pois qualquer ponto fora deste frustrum pode ser ignorado. Se houver múltiplas deteções de radar dentro deste RoI, toma-se o ponto mais próximo como a deteção de radar correspondente a este objeto. Na fase de treino, utiliza-se a caixa de deteção tridimensional do objeto para criar uma RoI em forma de frustum e associar deteções de radar para o objeto. Na fase de teste, o frustum é calculado a partir da estimação da caixa de deteção 3D. Neste caso, utiliza-se um parâmetro para controlar o tamanho da RoI, como mostra a figura 22. Isto explica a falta de exatidão na profundidade estimada, visto que a profundidade do objeto nesta fase é apenas determinada utilizando as características baseadas na imagem. Aumentar a região frustum utilizando este parâmetro aumenta a possibilidade de incluir as deteções de radar correspondentes dentro do frustum, mesmo se a profundidade estimada for ligeiramente inferior. O valor deve ser cuidadosamente selecionado, dado que uma grande RoI pode incluir deteções radar de objetos próximos. A abordagem enunciada elimina o problema dos objetos sobrepostos, pois cada um possui um frustum próprio e resolve o problema da múltipla deteção, dado que apenas a deteção de radar mais próxima dentro da RoI está associada ao objeto. Não ajuda, no entanto, com o problema da dimensão z, visto que as deteções do radar podem estar fora do ROI do seu objeto correspondente devido à informação inexata sobre a altura.

Expansão de pilares

Para abordar a informação inexata sobre o problema da altura, introduziu-se um pré-processamento de nuvens de pontos de radar chamado expansão de pilares, onde cada ponto de radar é expandido para um pilar de tamanho fixo, como ilustrado na figura 23. Os pilares criam uma melhor representação para os objetos físicos detetados pelo radar, visto que estas deteções estão agora associadas com uma dimensão no espaço 3D.

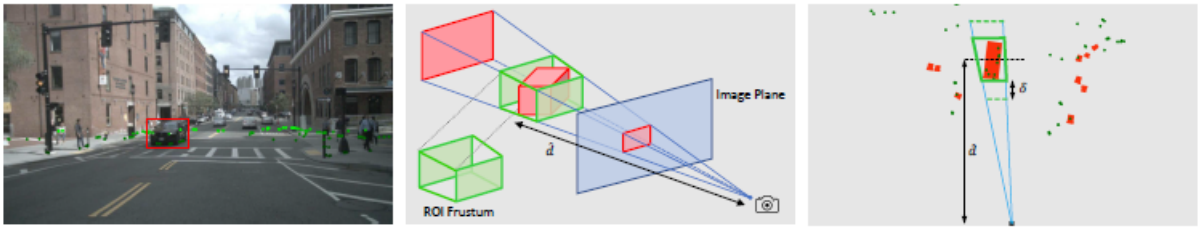


Figura 22: Associação ao tronco do RoI 3D. Na esquerda, pode-se observar o objeto a ser detetado pelas *features* da imagem. Na imagem do meio o frustum originado através da caixa de delimitação 3D do objeto, sendo d a profundidade e a sua estimativa no treino e teste, respetivamente. Na direita, o parâmetro usado aumentar o tamanho do frustum na fase de testes. Imagem retirada de (Nabati and Qi, 2020).

Deste modo, considera-se apenas as deteções do radar aquelas cujos pilares intersetam as regiões frustum, total ou parcialmente.

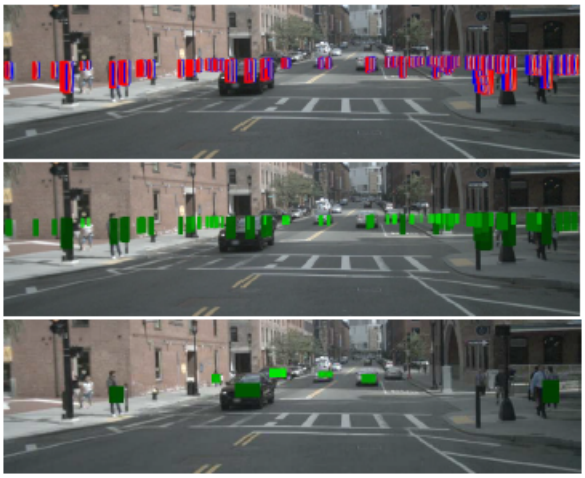


Figura 23: A expansão de pilares a partir dos pontos de deteção permite corrigir o problema de associação que ocorre quando as deteções em ponto encontram-se diretamente acima ou abaixo, na direção da altura (z), das regiões frustum. Na imagem de topo, pode-se ver a expansão de pilares e na imagem do meio o seu mapeamento direto com o centro dos objetos, resultando num cálculo de valores errados de profundidade. Na imagem de baixo, verifica-se o seu mapeamento com o frustum dos objetos, obtendo-se uma leitura precisa da profundidade destes. Imagem retirada de (Nabati and Qi, 2020).

Extração de features de radar

Após associar as deteções de radar aos objetos, utiliza-se a profundidade e a velocidade das deteções do radar para criar características complementares para a imagem. Particularmente, cada deteção radar associada a um objeto origina três canais de mapa centrados e dentro de uma caixa de delimitação 2D do objeto, como mostra a figura 23. A a largura e a altura dos mapas térmicos são proporcionais às caixas 2D do objeto, sendo controlados por um parâmetro. Os valores do mapa térmico são a profundidade normalizada do objeto (d) e também os componentes x e y da velocidade radial (vx e vy) no sistema de coordenadas egocêntrico:

$$F^j_{x,y,i} = \frac{1}{M_i} \begin{cases} f_i \left| x - c^j_x \right| \leq a w^j \wedge \left| y - c^j_y \right| \leq a h^j \\ 0 \end{cases} \tag{10}$$

onde i ∈ 1,2,3 no canal de mapa de *features*, M_i é um fator de normalização, f_i é o valor do *feature* que pode ser d, v_x, ou v_y; c_x^j e c_y^j são as coordenadas do centro do objeto j e w^j e h^j a largura e altura da caixa de deteção 2D. Se dois objetos tiverem zonas sobrepostas no mapa térmico, o que tiver uma profundidade menor domina, uma vez que apenas o objeto mais próximo é totalmente visível na imagem. Os mapas de calor originados são então concatenados com as *features* da imagem como canais extra. Estas *features* são utilizadas como entradas para as cabeças de regressão secundárias para recalcular a profundidade e rotação do objeto, bem como a velocidade e os atributos. As cabeças de regressão de velocidade estimam as componentes x e y da velocidade real do objeto segundo o sistema de coordenadas do veículo. As estimativas das cabeças de regressão de atributos permitem estimar se um carro está parado apenas ou estacionado, por exemplo, e consistem em três camadas de convolução com filtros 3x3 seguidas de uma camada 1x1 para originar a saída desejada. As camadas extra convolucionais em comparação com as cabeças de regressão primárias ajudam na aprendizagem de características de nível superior a partir dos mapas de *features* do radar. O último passo é descodificar os resultados da cabeça de regressão em caixas de delimitação 3D. O descodificador de caixas utiliza a profundidade, velocidade, rotação e atributos estimados das cabeças de regressão secundárias, e obtém outras propriedades dos objetos a partir das cabeças primárias.

Tendo em conta que esta é uma solução relativamente simples, recente (2020) e até bastante popular (abordagem com maior pontuação no sítio web *www.paperswithcode.com*) torna-se assim a indicada de entre as demais já mencionadas.

2.8 Ambientes de simulação

A elaboração de testes de condução autónoma ao vivo na via pública é uma atividade pouco segura, custosa e nem sempre passível de ser efetuada. Consequentemente, para além de testes, utilizam-se outras metodologias para treino de redes neuronais usadas para a condução de automóveis que recorrem ao uso de determinadas ferramentas de simulação para obtenção de imagens de jogos como o *Grand Theft Auto V*. No entanto, com o aumento da popularidade desta indústria, surgiram simuladores especializados e construídos de raíz para o propósito de fomentar a investigação na área. Nesta secção são delineados os requisitos-chave que os simuladores devem ter, abordados alguns dos simuladores mais usados e, no final, é efetuada uma escolha do melhor após comparação entre os demais. Como se pode verificar na figura 24, existem requisitos a nível funcional dos VA como a capacidade de resolver problemas de geometria em múltiplas perspetivas, resultado da projeção de diversos ângulos do cenário em diferentes câmaras usadas por um VA.

Tal habilidade permite ao veículo simultaneamente localizar-se e efetuar o mapeamento do ambiente, *Simultaneous Localization and Mapping (SLAM)*. Outro requisito prende-se com uma adequação realista não só da representação em 3 dimensões dos objetos encontrados na estrada, como também especificamente de elementos reguladores de trânsito e a sua representação semântica e ainda a emulação dos mais diversos cenários de condução, que encobre desde o comportamento de peões até à geração de diferentes condições climatéricas, capacidade de uso de diferentes tipos de sensores, simulação de acidentes rodoviários, entre outros. Além disso, um simulador adequado para VA deve possibilitar um leque de aplicações como o rótulo em 2D/3D de *ground truth* para treino de redes neuronais para a função de

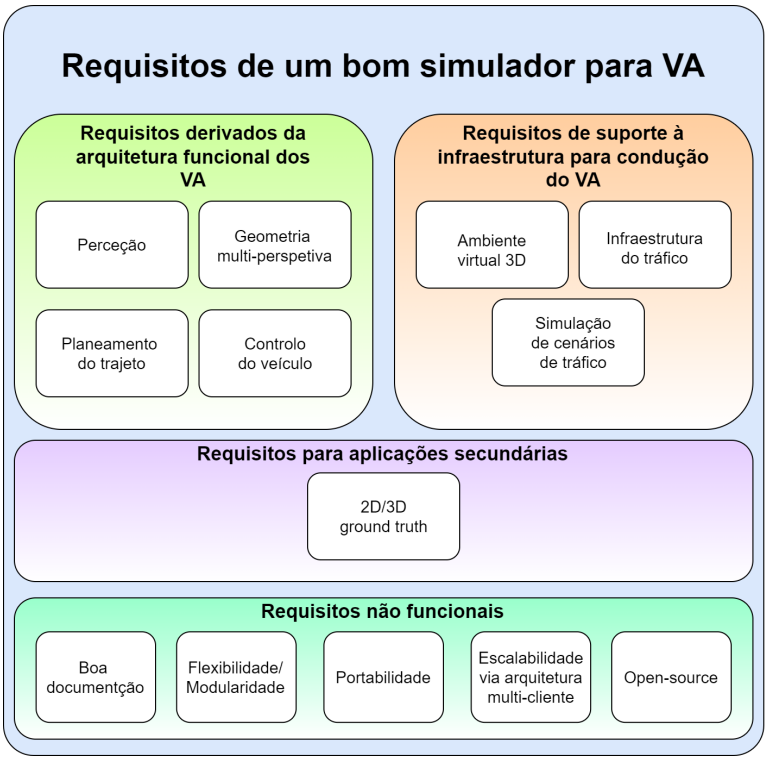


Figura 24: Requisitos de um bom simulador para VA

.

perceção, obtendo assim dados sintéticos que apesar de não possuírem o mesmo valor que imagens de conjunto de dados reais, é possível a sua geração em massa a custo monetário e temporal negligível, para além de não ter eventuais problemas legais associados à permissão de captação de vídeo em determinadas zonas existentes no mundo real. Por fim, existem requisitos não funcionais que devem ser alcançados por simuladores, nomeadamente, ser flexível na personalização de diversos agentes, sensores ou cenários; ser portável, permitindo a sua execução em diferente sistemas operativos; ser escalável para permitir o controlo em simultâneo de diferentes agentes através de uma arquitetura multi-cliente.

Alguns simuladores a salientar são:

- **Matlab/Simulink** - através da *Automated Driving Toolbox* é possível testar funcionalides chave como perceção, planeamento do trajeto e controlo do veículo por meio de exemplos já pré-feitos e de boa documentação. Permite testar e implementar em diagrama de blocos, *Simulink*, originar código em C/C++ e possui interfaces com softwares como *RoadRunner* para exportação de cenários.

- **CarSim** - outro simulador, não open-source, que possui várias interfaces com outros softwares como Matlab e LabVIEW, ampla documentação e possui um portfólio extensivo de até 200 objetos de tráfego com localizações e movimentos independentes.

- **PreScan** - possui uma framework para o design de sistemas *Advanced Driving Assistance Systems (ADAS)* e veículos de condução autónoma. Oferece em particular uma funcionalidade de veículo *Hardware-In-the-Loop*, *VeHIL*, permitindo utilizadores combinarem sistemas sintéticos e reais onde

o veículo de teste é equipado com sensores realistas, permitindo assim ter uma simulação detalhada para aplicações ADAS como já referido.

- **Carla** - simulador open-source com API de suporte para treino e validação de sistemas ADAS assim como para VA. Através da sua arquitetura servidor e multi-cliente, permite a geração de cenários complexos onde cada cliente é responsável por controlar uma função específica como o tempo, o número de carros ou o programa teste a executar e o servidor como a componente que possui o cenário simulado alterado segundo comandos dos clientes. Possui grande diversidade de sensores, mapas, elementos de tráfego e encontra-se em rápido e constante atualização e melhoramento com o surgimento de novas versões para os sistemas operativos Windows e Linux.

- **Gazebo** - simulador open-source e multi-robô para aplicações 3D, especificamente para implementar soluções que utilizem robôs de vários graus de liberdade para use-cases na indústria em tarefas de pick-place, por exemplo. Apesar de não ter sido construído diretamente para simular VA e cenários para estes, em conjunto com *Robotic Operative System (ROS)*, é possível emular um modelo complexo de um Prius Hybrido numa cidade sintética M-city.

- **LGSVL** - plataforma de desenvolvimento de multi-robôs de VA que permite exportação em 3D e em HD de mapas, similar ao CARLA. Contudo, este software não permite diretamente a gravação de vídeos como o CARLA, sendo o problema contornado com o uso de *drivers* da NVIDIA.

O simulador CARLA é o mais completo dos demais com base no artigo (Kaur et al., 2021). Por isso, serviu de ponto de partida para encontrar conjuntos de dados sintéticos como o CADET - "CArla Dataset Extraction Tool", (Brekke, 2019) de forma a poder realizar experiências cujos conjuntos de dados eram compostos por dados reais e obtidos via simuladores. Mais tarde acabou-se por optar pelo conjunto de dados KITTI e Virtual KITTI dado a sua popularidade e compatibilidade com os modelos treinados, apesar de que os dados obtidos por simuladores como o Carla não deixam de ser uma mais valia, desde que se consiga obter compatibilidade com os modelos selecionados para alvo de experiências.

2.9 Deteção multi-modal de objectos 3d a partir de pré-treino

No trabalho descrito por (Brekke et al., 2019), estudou-se o efeito do treino e fine-tuning de algoritmos multi-modais, compatíveis com dados de câmara e lidar, em conjunto de dados simulados que foram originados pelo simulador Carla. A ferramenta usada para extração desses dados, CADET, encontra-se no repositório Github dos autores. A ferramenta CADET foi utilizada para originar 10000 amostras com dados obtidos no simulador Carla para treinar o objeto de deteção 3D AVOD-FPN. Posteriormente, avaliou-se o desempenho deste modelo na deteção de carros e pessoas quando treinado e testado no conjunto de dados real KITTI, treinado e testado no CADET, treinado no CADET e testado no KITTI, e treinado no CADET e efetuado o fine-tuning e teste no conjunto de dados KITTI. Assim foi possível observar os diferentes efeitos nas performances de um dado algoritmo quando este é treinado, validado, testado e efetuado o processo de fine-tuning com conjuntos de dados reais, sintéticos e híbridos. Concluiu-se que apesar de na última experiência o modelo não ter atingido os resultados obtidos quando o mesmo fora treinado e testado no conjunto de dados real

Requirements	MATLAB (Simulink)	CarSim	PreScan	CARLA	Gazebo	LGSVL
Perception: Sensor models supported	Y	Y	Y	Y(1)	Y(2)	Y(3)
Perception: support for different weather conditions	N	N	Y	Y	N	Y
Camera Calibration	Y	N	Y	Y	N	N
Path Planning	Y	Y	Y	Y	Y	Y
Vehicle Control: Support for proper vehicle dynamics	Y	Y	Y	Y	Y	Y(3)
3D Virtual Environment	U	Y	Y	Y, Outdoor (Urban)	Y, Indoor and Outdoor	Y, Outdoor (Urban)
Traffic Infrastructure	Y, allows to build lights model	Y	Y	Y, Traffic lights, Intersections, Stop signs, lanes	Y, allows to manually build all kinds of models	Y
Traffic Scenario simulation: Support of different types of Dynamic objects	Y	Y	Y	Y	N(2)	Y
2D/3D Ground Truth	Y	N	N	Y	U	Y
Interfaces to other software	Y, with Carsim, Prescan,ROS	Y, with Matlab(Simulink)	Y, with MATLAB(Simulink)	Y, with ROS, Autoware	Y, with ROS	Y, with Autoware, Apollo, ROS
Scalability via a server multi-client architecture	U	U	U	Y	Y	Y
Open Source	N	N	N	Y	Y	Y
Well-maintained/Stable	Y	Y	Y	Y	Y	Y
Portability	Y	Y	Y	Y, Windows, Linux	Y, Windows, Linux	Y, Windows, Linux
Flexible API	Y	Y	U	Y (2)	Y	Y

Y=supported, N = not supported, U = unknown
(1) See section 4.4 for details about Carla.
(2) See section 4.5 for details about Gazebo
(3) See section 4.6 for details about LGSVL

Figura 25: Comparação entre diferentes simuladores, retirado de (Kaur, 2021).

KITTI, estes resultados ficaram muito próximos. Daí que, neste trabalho pretende-se provar até que ponto os dados sintéticos poderão ser uma mais valia e uma boa alternativa no caso de haver falta de dados reais para a tarefa de deteção de veículos no domínio da condução autónoma, mas em vez do foco ser para dados de lidar obtidos pelo CARLA, verificar se o mesmo ocorre para o conjunto de dados KITTI e virtual KITTI. E também provar em que tipo de modelos poderão estes ser mais eficazes, se nos modelos multi-modais ou de modalidade única.

3

ESTUDO DE SELEÇÃO DOS RECURSOS PARA EXECUÇÃO DAS EXPERIÊNCIAS

Para iniciar não só o processo de treino como de execução torna-se necessário configurar o computador com as "drivers"ou controladores das placas gráficas dos mesmos.

3.1 Ambiente de execução

Utilizou-se o conjunto de ferramentas da CUDA para a realização das experiências de treino, validação e teste. Este contém bibliotecas em linguagem C que dão uso às potencialidades de aceleração de execução de código através das GPUs e ainda possui ferramentas de desenvolvimento bem como um compilador próprio. A versão utilizada corresponde à mais recente, 11.7, e o compilador é denominado assim de cuda-11.7.

Deste modo, torna-se necessário verificar as compatibilidades entre o hardware e o software. Neste caso, a placa gráfica associada ao computador que realizará as experiências corresponde à Nvidia GeForce RTX 3060. Como esta é compatível com a versão mais recente e estável do Cuda, versão 11.7, após uma formatação do computador e instalação do sistema operativo Ubuntu 18.04, instalou-se essa mesma versão do acelerador de hardware, que inclui o controlador Nvidia próprio da placa gráfica.

Posto isto, instalou-se o gestor e desenvolvimento de pacotes de software Anaconda. O mesmo permite a criação de diversos ambientes isolados uns dos outros onde o utilizador pode instalar apenas os pacotes de software (como o Python) e as respetivas versões para, por exemplo, replicar um ambiente compatível com o código que deseja executar. A mesma empresa do Anaconda inclui uma versão minimalista, a Miniconda, que ocupa menos espaço e inclui apenas o Conda (gestor de pacotes), Python, pacotes inter-dependentes e uma parte de outros pacotes disponíveis para instalação em comparação com o Anaconda.

3.2 Estruturas de software utilizadas

Após instalação do software mencionado acima, utilizou-se diferentes plataformas específicas para treinar, validar e ainda testar diferentes modelos de deteção de veículos, nomeadamente:

- CenterPoint
- CenterFusion
- OpenPCDet
- MMDetection3D

Das alternativas estudadas, a MMDetection3d é a framework mais indicada para as experiências a realizar dado que:

- Suporta detetores de modalidade múltipla/singular, isto é, permite a utilização de algoritmos que utilizam imagens e/ou nuvens de pontos para treino e teste. De entre alguns exemplos encontram-se: MVXNet, VoteNet e PointPillars.
- Suporta a deteção 3D interna/externa. De entre exemplos de conjunto de dados in-door vêm: ScanNet e SUNRGB-D. Quanto a conjunto de dados out-door temos, exemplificando, Waymo, nuScenes, Lyft e KITTI.
- Permite a integração com a deteção 2D que possui mais de 300 modelos e métodos de mais de 40 publicações.
- Possui elevada eficiência visto que o treino efetuado nesta plataforma é mais rápido que noutras, utilizando os mesmos modelos e conjunto de dados.
- É modular e pode ser utilizada como biblioteca de suporte para os mais diversos projetos.

Além disso, a implementação da MMDetection3d possui mais de 1000 commits no Github, é atualizada várias vezes por ano, possui forte suporte da comunidade dada a cotação de mais de 2830 estrelas, e, por fim, as questões colocadas pelos utilizadores da mesma são brevemente respondidas em dias. Ou seja, existe um rápido feedback na resposta a problemas levantados pelos utilizadores.

3.3 Algoritmos utilizados

Quanto aos algoritmos utilizados, dos compatíveis com a framework descrita acima, destacam-se:

- HV_PointPillars_FPN, que possui modalidade singular, podendo apenas processar nuvens de pontos
- Fcos3D, que possui modalidade singular, podendo apenas processar imagens RGB
- MXVNet, de múltipla modalidade que consegue processar imagens RGBs e nuvens de pontos simultaneamente

O modelo HV_PointPillars_FPN (hv - HardSimpleVFE, ou Codificador de Caraterísticas de Vóxeis, ou píxeis com o elemento de volume) surgiu de uma resposta ao problema de codificar dados em 3D agrupados em nuvem de pontos, figura 27. Existem dois tipos de codificadores; codificadores fixos, que tendem a ser rápidos, mas que sacrificam na precisão, e codificadores mais precisos, mas mais lentos. Com o modelo PointPillars, é possível aprender uma representação de nuvens de pontos organizada em colunas verticais (pilares) utilizando a estrutura PointNets, figura 26. Após terem sido efetuados bastantes testes concluiu-se que o PointPillars supera os codificadores anteriores em relação à velocidade e precisão por uma grande margem. Apesar de usar apenas lidar, a pipeline de deteção supera significativamente o estado da arte, mesmo entre os métodos de fusão, em relação aos benchmarks KITTI 3D e de visão aérea. Este desempenho de deteção é alcançado durante a execução a 62 Hz: uma melhoria de tempo de execução de 2 a 4 vezes. Existe ainda uma versão

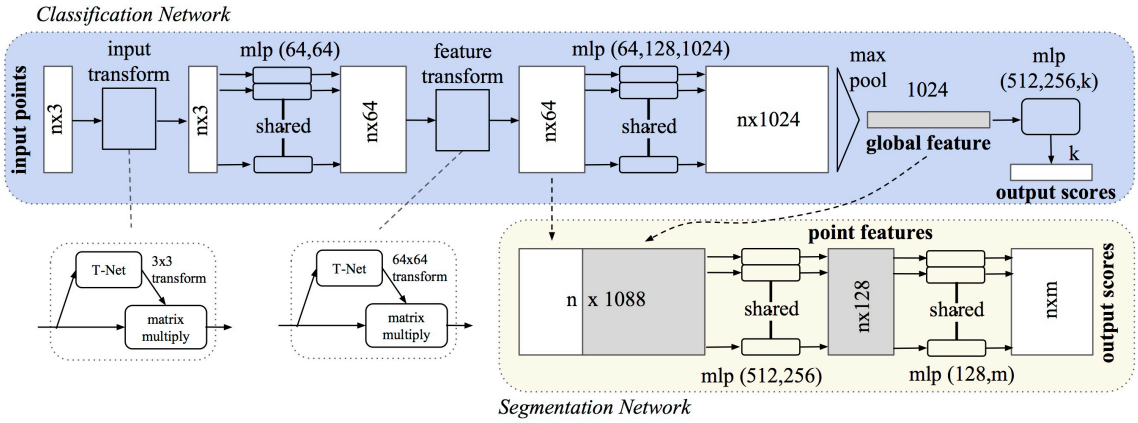


Figura 26: Arquitetura do modelo Pointnet. A rede de classificação recebe n pontos como entrada, aplica transformações de entrada e de recurso e, em seguida, agrega as nuvens de pontos pelo maior agrupamento. A saída é a pontuação de classificação para m classes. A rede de segmentação é uma extensão da rede de classificação. Ele concatena recursos e resultados globais e locais por pontos. Os MLP ou os percetores multi-camada são classes de rede neuronais totalmente conectados sendo que os números entre parênteses são os seus tamanhos de camada. A normalização em lote, Batchnorm, é uma técnica usada para tornar os treinos mais rápidos e estáveis através do redimensionamento e recentralização das camadas neuronais e é usada para todas estas que possuem um retificador, ReLU. Também são usadas camadas de dropout, isto é, redes de neurónios que se desligam de forma aleatória no processo de treino para evitar over-fitting para o último percetor multi-camada na rede de classificação.

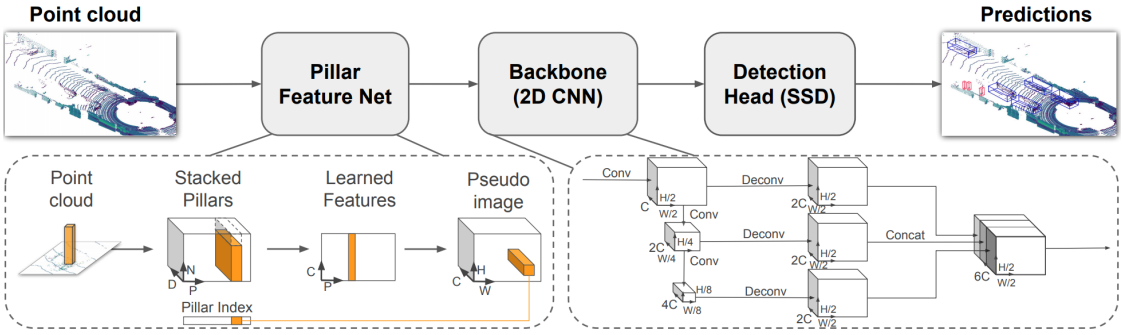


Figura 27: Arquitetura do modelo hv_pointpillars_fpn. Do ficheiro de entrada de nuvem de pontos até à obtenção do ficheiro de saída com as anotações das deteções, o algoritmo processa os dados numa primeira fase, upstream, em que as features extraídas são organizadas em pilares (FPN). Posteriormente, os resultados são pós-processados por uma rede convolucional 2D (Backbone) que os concatena antes de enviar para o estágio de downstream, que efetua a classificação final com uma rede SSD, ou single-shot-detection, obtendo-se as deteções pretendidas. Fonte: (Lang et al., 2019).

ainda mais rápida do método que corresponde ao estado de arte em 105 Hz. O modelo PointPillars é pois uma codificação apropriada para deteção de objetos em nuvens de pontos.

A estrutura geral do modelo FCOS3D foi criada para tentar resolver o problema da falta de informação quanto à profundidade nos métodos de deteção 3D que utilizam apenas uma câmara (monoculares). O algoritmo transforma os alvos 3D 7-DoF definidos no domínio da imagem e os desacopla como atributos 2D e 3D. Em seguida, os objetos são distribuídos para diferentes níveis de recursos considerando suas escalas 2D e atribuídos apenas de acordo com o centro 3D projetado para o treino. Além disso, a centralidade é redefinida com uma distribuição gaussiana 2D baseada no centro 3D para se adequar à formulação do alvo 3D. A estrutura é assim simples e eficaz por conseguir eliminar qualquer deteção 2D ou anterior de

correspondência 2D-3D. Esta solução alcança o 1° lugar de todos os métodos somente de visão no desafio de deteção 3D nuScenes do NeurIPS 2020. Fonte: (Wang et al., 2021a).

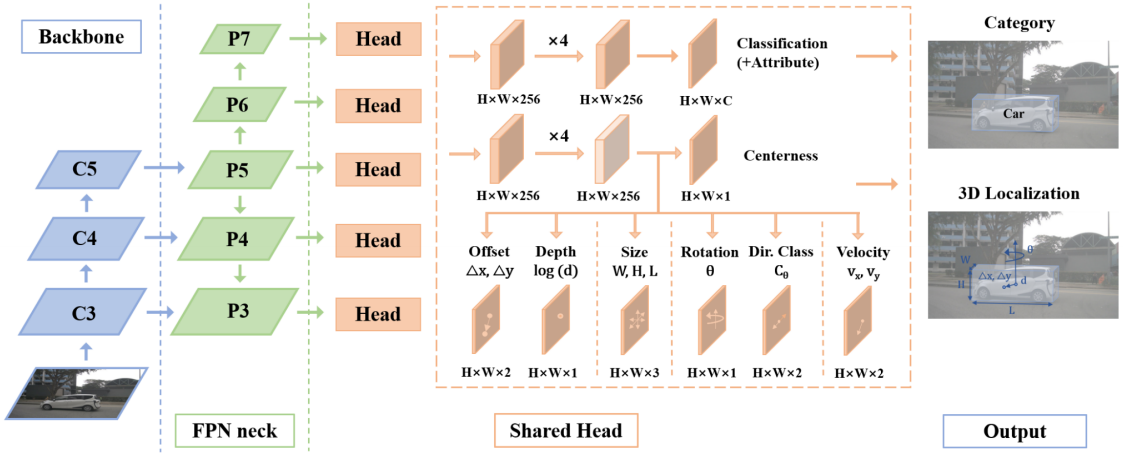


Figura 28: Arquitetura do modelo FCOS3d. Para aproveitar os extratores de *features* 2D, o algoritmo utiliza o design do *backbone* ou tronco e *neck* ou pescoço dos detetores 2D. Para a cabeça, reformulou-se os alvos 3D com o paradigma baseado no centro para desacoplar de uma aprendizagem multi-tarefas. As estratégias de múltiplos níveis de atribuição de alvo e amostragem de centro são posteriormente ajustadas em concordância de forma a equipar a framework com melhor capacidade de lidar com a sobreposição de *ground truths* e problemas de variância de escala. Imagem retirada de (Wang, 2021).

A arquitetura Multimodal Voxel-Net (MVX-Net) para deteção de objetos 3D foi criada a fim de resolver o problema das redes convencionais não terem em consideração a interação das diferentes modalidades de dados nos estágios iniciais. As últimas limitam-se a processar apenas os dados de uma modalidade, ou quando muito, de forma sequencial e tardia os dados de múltiplos tipos de sensores. Os modelos MVX-Net apresentam duas abordagens para a fusão precoce de dados, PointFusion e VoxelFusion, que combinam as modalidades RGB e nuvem de pontos. A PointFusion involve a projeção de pontos 3D numa imagem a partir de uma matriz de calibração, seguida de um processo de extração de caraterísticas através de uma rede CNN em 2D pré-treinada e uma concatenação de caraterísticas de imagens ao nível dos pontos. A Voxelfusion involve a projeção em 3D dos vóxeis na imagem, seguida de uma extração de caraterísticas em 2D RoIs e concatenação de agregados de caraterísticas de imagens ao nível do vóxel. A avaliação no conjunto de dados KITTI demonstra melhorias significativas no desempenho em relação às abordagens que usam apenas dados de nuvem de pontos. Além disso, o método proposto fornece resultados competitivos com os algoritmos multimodais de última geração, alcançando a classificação top 2 em cinco das seis categorias de visão aérea e deteção 3D no benchmark KITTI, usando uma rede simples de estágio único. Fonte: (Sindagi et al., 2019).

3.4 Datasets utilizados

Estudaram-se vários conjunto de dados e efetuaram-se várias experiências piloto com o propósito de escolher os que mais se adequam para o treino dos algoritmos acima mencionados. De entre os demais, destacam-se:

- CADET
- Kitti-Carla

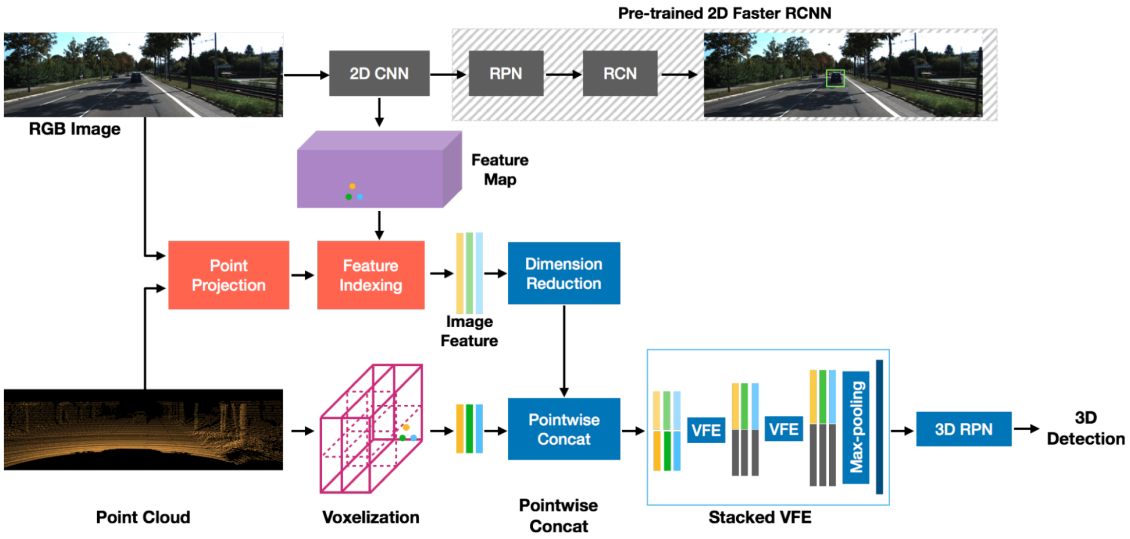


Figura 29: Visão geral do método MVX-Net PointFusion proposto. O método usa filtros convolucionais de RCNN 2D pré-treinados mais rápidos para calcular o mapa de recursos da imagem. Observe que o RPN e o RCN (mostrados no retângulo sombreado) não fazem parte do pipeline de inferência 3D. Os pontos 3D são projetados para a imagem usando as informações de calibração e os recursos de imagem correspondentes são anexados aos pontos 3D. As camadas VFE e o RPN 3D processam os dados agregados e produzem as deteções 3D. Retirado de (Vishwanath et al., 2019).

- NuScenes
- Kitti
- virtual KITTI 1.3.1
- Vkitti3D-conjunto de dados

Apenas o KITTI e o virtual KITTI foram de facto utilizados posteriormente para as experiências a realizar visto que:

- são compatíveis com a framework Mmdetection3d
- o KITTI é compatível com o modelo MVX-Net
- o virtual KITTI é uma adaptação sintético do KITTI, possibilitando a sua formatação para o conjunto de dados Kitti

3.4.1 *KITTI*

O conjunto de dados KITTI atua como um benchmark de deteção de objetos e estimativa de orientação destes composto por 7.481 imagens de treino e 7.518 imagens de teste, possuindo ao todo 80.256 objetos rotulados. Todas as imagens são coloridas e guardadas como ficheiros com extensão "PNG". Para a avaliação, calculou-se as curvas de "precision-recall"para deteção de objetos e curvas "similar-orientation-recall"para deteção conjunta de objetos e estimativa de orientação. Neste último caso, não apenas a caixa delimitadora do objeto

deve ser localizada corretamente, mas também a estimativa de orientação na visão aérea é avaliada. Para classificar os métodos, calculou-se a precisão média e a similaridade de orientação média. Abaixo na tabela 6 podemos verificar que dificuldade é atribuída a cada bounding box, consoante a altura desta em píxeis, nível de oclusão e nível de truncação, ambas em percentagem. A oclusão mede o quão o objeto de interesse detetado pela bounding box se encontra visível sob a perspetiva da câmara, dado que estes podem ser escondidos se se puseram outros objetos à frente dos mesmos. A truncação corresponde à porção do objeto que se encontra fora da bounding box em relação à totalidade do objeto. De notar que uma deteção só é válida se a IoU ou interseção sobre a união entre a bounding box criada pelo algoritmo e a da ground truth, anotada de antemão, for igual ou superior a 0.7 para carros e 0.5 para as demais categorias como pedestres ou ciclistas. Múltiplas deteções ou bounding boxes para o mesmo objeto de interesse contam como falsos positivos. Avaliou-se o desempenho da deteção conjunta de objetos e estimou-se a sua orientação 3D utilizando a medida designada por semelhança de orientação média, AOS (Liu, 2019), definida da seguinte forma:

$$AOS = \frac{1}{11} \sum_{r \in 0,0.1,...1} max_{\tilde{r}:\tilde{r} \geqslant r} s(\tilde{r}) \tag{11}$$

Onde a variável r é definida como:

$$r = \frac{TP}{TP + FN} \tag{12}$$

Sendo que $s \in [0,1]$ na medida de recall r, isto é, o número de corretas predições sobre o número total de predições. Esta variável s é então uma variante do coseno similarmente definida como:

$$s(r) = \frac{1}{|D(r)|} \sum_{i \in D(r)} \frac{1 + cos\Delta_{\theta}^{(i)}}{2} \delta_i \tag{13}$$

Onde D(r) significa o conjunto de todas as deteções de objetos na taxa de recolha r e $\Delta_{(i)}^{\theta}$ corresponde à diferença de ângulo entre a estimativa e a orientação da deteção da ground truth i. Para penalizar múltiplas deteções associadas a um mesmo objeto, definimos $\delta_i = 1$ se a deteção foi atribuída a um limite de deteções ground truth com sobreposições de pelo menos 50% e $\delta_i = 0$ se não tiver sido atribuído. Retirado de (Andreas and Raquel, 2012).

KITTI classification difficulty level	**Min. Bounding Box Height (px)**	**Max. Occlusion Level**	**Max. Truncation (%)**
Easy	40	Fully Visible	15
Moderate	25	Partly Occluded	30
Hard	25	Difficult to see	30

Tabela 6: Nesta tabela podemos saber quais as condições que se têm de realizar para o conjunto de dados KITTI reconhecer a dificuldade de classificar uma dada bounding box como Easy, Moderate, ou Hard - Fácil, Moderado ou Difícil. Dados retirados de (Geiger, 2012a).

3.4.2 *Virtual Kitti*

O conjunto de dados virtual KITTI é um conjunto de dados de vídeo sintético foto-realista concebido para aprender e avaliar modelos de visão por computador para as várias tarefas de compreensão de vídeo: deteção de objetos e rastreio multiobjeto, segmentação semântica a nível de cenário e de instância, fluxo ótico que mapeia em vetores de deslocamento os movimentos ocorridos entre duas imagens consecutivas, e estimativa de profundidade.

O KITTI sintético contém 50 vídeos monoculares de alta resolução (21.260 frames) originados a partir de cinco mundos sintéticos diferentes em ambientes urbanos sob diferentes condições de imagem e meteorológicas. Estes mundos foram criados utilizando o motor de jogo Unity e um novo método de clonagem real-para-sintético. Estes vídeos sintéticos foto-realistas são anotados automaticamente para rastreio de multi-objetos 2D e 3D e a nível de pixel com anotações ao nível de classificação, instância, fluxo ótico, e profundidade. Retirado de (Gaidon et al., 2016).

3.5 Parâmetros de treino/teste

Para executar o treino e teste dos algoritmos, de todos os parâmetros configuráveis presentes por defeito na framework mmdetection3D para os modelos de MVX-Net e Fcos3d, apenas se modificaram:

Parâmetros de treino/teste	**Valor por defeito**	**Valor estabelecido**
samples_per_gpu	4	1
workers_per_gpu	4	1
learning_rate	0.003	0.002
with_attr_label	True	False
input_modality	use_lidar=True, use_camera=True	use_lidar=False, use_camera=True
class_names	['car', 'truck', 'trailer', 'bus', 'construction_vehicle', 'bicycle', 'motorcycle', 'pedestrian', 'traffic_cone', 'barrier']	['Pedestrian', 'Cyclist', 'Car']
keys	['img', 'gt_bboxes', 'gt_labels', 'gt_bboxes_3d', 'gt_labels_3d', 'attr_label', 'centers2d', 'depths']	['img', 'gt_bboxes', 'gt_labels', 'gt_bboxes_3d','gt_labels_3d', 'centers2d', 'depths']

Tabela 7: Parâmetros modificados para execução das experiências. O parâmetro with_attr_label foi colocado a false para não considerar as anotações na fase de testes. O parâmetro use_lidar passou a false para considerar apenas os dados de câmara.

3.5.1 *Otimizador*

Foi utilizado o otimizador AdamW ao fazer passar quarenta vezes (*epoch*=40) o conjunto de dados de treino pelo algoritmos MVX-Net e Fcos3d. O modelo baseado em PointPillars acabou por não pertencer . Os optimizadores são algoritmos ou métodos utilizados para alterar os atributos de uma rede neural, tais como

os parâmetros e a taxa de aprendizagem, a fim de reduzir as perdas. O optimizador AdamW é a solução mais adequada para a generalidade dos casos pois possui melhores tempos computacionais e requer menos parâmetros para o fine tuning.

3.5.2 *Ambiente de execução padrão*

Os ficheiros utilizados tanto para treino como para validação e teste estão agrupados numa pasta denominada ImageSets composta por:

- train.txt: contém a listagem dos números dos ficheiros para treino. 5237 ficheiros usados para treino (70%).
- val.txt: contém a listagem dos números dos ficheiros para validação. 1496 ficheiros no total ou 20%. Os ficheiros de validação permitem uma avaliação do modelo após uma etapa do treino para verificar o seu desempenho e, ao mesmo tempo, é utilizado para afinar os hiperparâmetros do modelo.
- trainval.txt: possui a listagem dos números dos ficheiros para treino e validação.
- test.txt: possui a listagem dos números dos ficheiro para teste para avaliar o desempenho do modelo após terminar o treino e as etapas de validação num conjunto de dados diferente do usado nos outros dois conjuntos. Total de 748 ficheiros usados, o que corresponde a 10% do número total de ficheiros existentes para efetuar a experiência com o conjunto de dados KITTI. O KITTI possui um conjunto de dados de teste com 7518 ficheiros de dados que não foi possível ser obtido, mesmo após ter justificado o uso para a finalizar esta dissertação. Em vez disso, foi recomendado pelas mesmas autoridades do KITTI que utilizasse um subconjunto do conjunto de dados de treino para formar a partição de testes. Deste modo, escolheu-se e isolou-se de forma aleatória os ficheiros de teste provenientes da partição de treino.

Para efetuar o treino ou teste, basta introduzir na linha de comandos: python <ficheiro de python de configuração geral de treino ou teste> <ficheiro de python de configuração do algoritmo em si> [ficheiro de python de um checkpoint de um modelo] [–eval mAP]. Por cada passagem do conjunto de dados de treino inteiro pelo algoritmo, é efetuada uma validação e os resultados das métricas de precisão do algoritmo são reportados na consola ao fim de cada época, ou *epoch*.

O programa automaticamente efetua multi-threading caso sejam detetados múltiplos cores não ocupados para acelerar o processo de treino.

O processo de validação é linear, isto é, é utilizado o mesmo subset de validação para todas as experiências.

3.6 Planeamento e estruturação de código para as experiências a realizar

Seguindo a estrutura do artigo (Brekke et al., 2019), pretende-se para este projeto provar os seguintes pontos:

- Comparar as diferenças a nível de precisão dos modelos pré-treinado da literatura, MVX-Net e Fcos3D, e os mesmos treinados (e validados) de raíz.

- Comparar as métricas de precisão entre detetores de modalidade única (Fcos3D) e modalidade múltipla (MVX-Net, usa dados de câmara e lidar).
- Verificar o efeito em treinar um modelo multi-modal (MVX-Net) e um modelo de modalidade única (Fcos3D, usa apenas dados de câmara) em conjunto de dados com diferentes proporções de dados reais e sintéticos (originados num simulador).
- Verificar a eficácia do pré-treino de algoritmos em conjunto de dados mistos.

3.6.1 *Seleção da modalidade: câmara/lidar*

Os modelos estudados são de modalidade singular (Fcos3D) e múltipla (MXV-NET). Deste modo, os testes realizados que envolvem dados de câmara, reais e sintéticos, foram efetuados em ambos os modelos. Quanto aos testes que envolveram câmara e lidar, foi utilizado o modelo MXV-Net. Devido à falta de conjunto de dados de lidar sintéticos em geral, os testes que envolveram apenas dados de lidar, que foram efetuados pelo modelo baseado em PoinPillars que utiliza apenas dados em forma de nuvens de pontos, não apresentaram resultados significativos, visto que o número total de dados obtidos para treino, validação e teste não passou de 90, daí não serem reportados neste documento.

3.6.2 *Diagrama de Experiências*

Os diagramas das figuras 30, 31, 32 e 33 representam de modo simplificado o planeamento das experiências. Na primeira e segunda, cinco instâncias dos modelos Fcos3D e MVX-Net são treinados durante 40 épocas com conjuntos de dados 100% sintéticos, 100% reais, e com diferentes partições de dados reais e sintéticos (75/25%, 50/50% e 25/75% respetivamente). 70% do número total de dados correspondem ao treino e são integrados a estes volumes 20% dos dados totais que contam para a validação, perfazendo os 90% de dados ou 6733 ficheiros presentes em cada um dos cinco volumes. O volume de teste apenas possui 10% do número total de dados e estes são 100% reais e provenientes do KITTI. Tanto os ficheiros de dados sintéticos como os reais foram alocados de forma aleatória, por meio do código setup_datasets.py, de forma a obedecer à estrutura abordada, mais detalhadamente descrita nas imagens que se seguem.
Já as experiências 3 e 4 foram efetuadas para os modelos que atingiram melhor desempenho dos conjuntos de modelos obtidos após cada uma das 40 épocas das experiências anteriores. Utilizando então as instâncias pré-treinadas do Fcos3D e MVX-Net com a melhor performance, respetivamente, utilizou-se desta vez um conjunto de dados 100% real para treino e validação, durante 8 épocas, e voltou-se a testar os modelos com um conjunto de dados de teste real, tudo proveniente do KITTI.

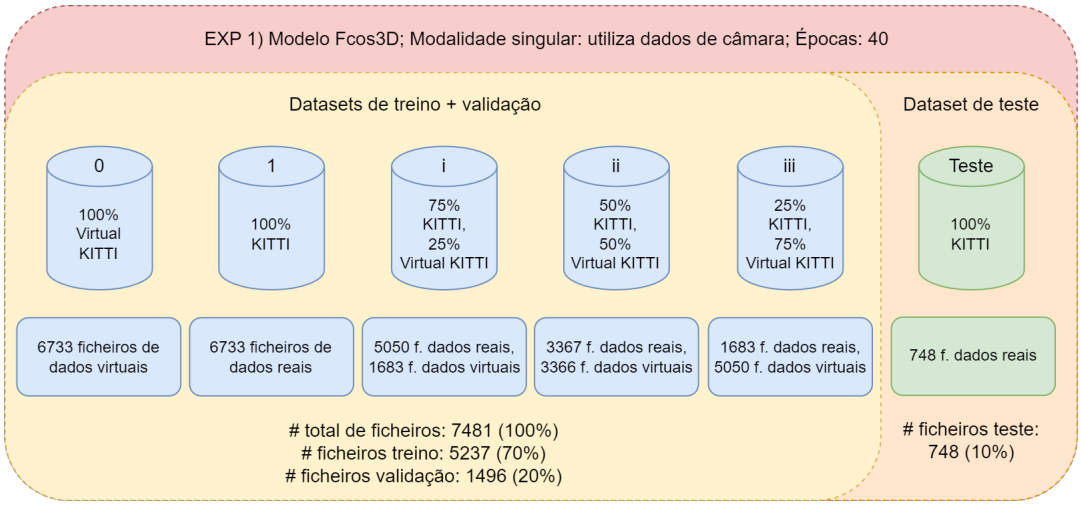


Figura 30: Experiência número 1: Treino, validação e teste do modelo Fcos3D nos conjunto de dados KITTI e virtual KITTI.

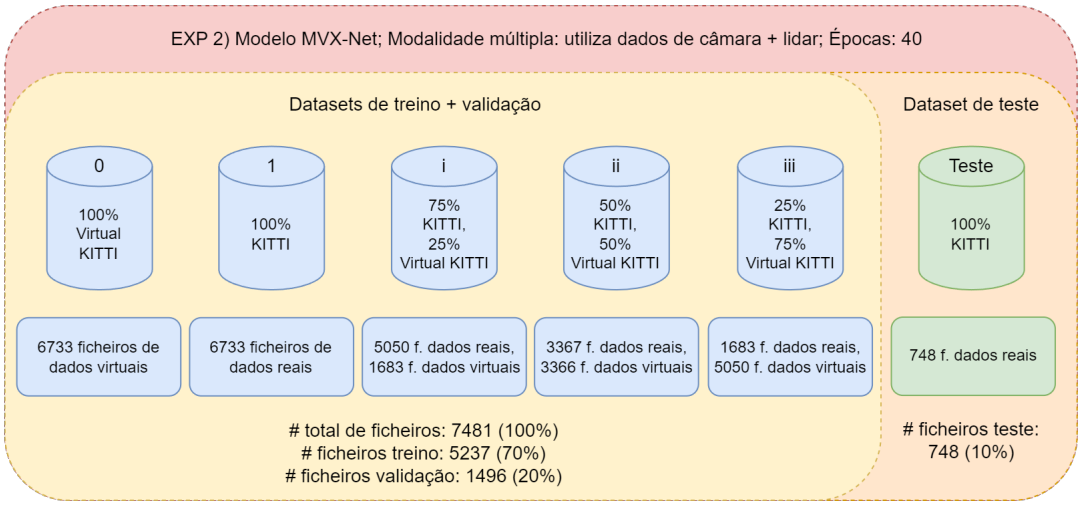


Figura 31: Experiência número 2: Treino, validação e teste do modelo MVX-Net nos conjunto de dados KITTI e virtual KITTI.

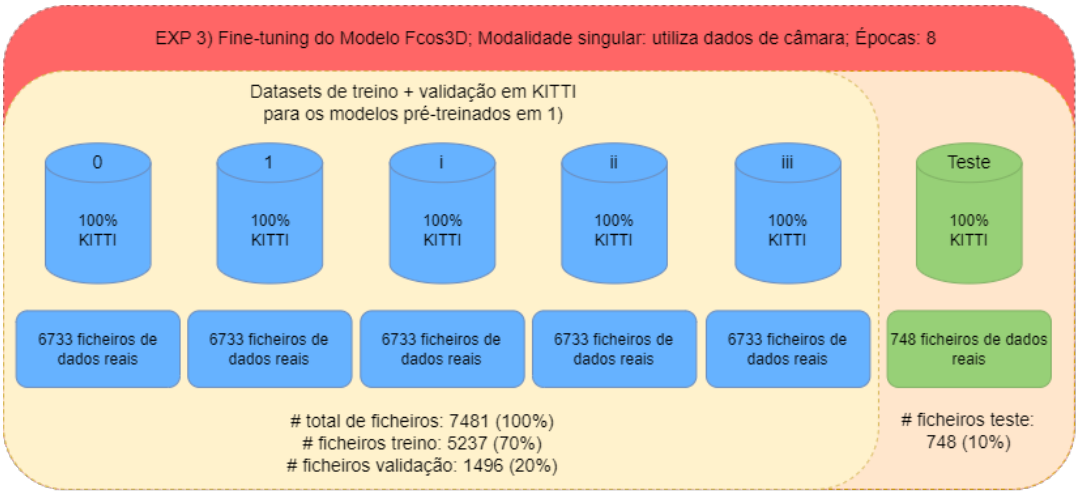


Figura 32: Experiência número 3: Treino, validação e teste dos modelos Fcos3D pré-treinados da experiência 1) no conjunto de dados KITTI. A diferença entre cada uma das 5 partes prende-se com a origem do modelo pré-treinado obtido na experiência 1.

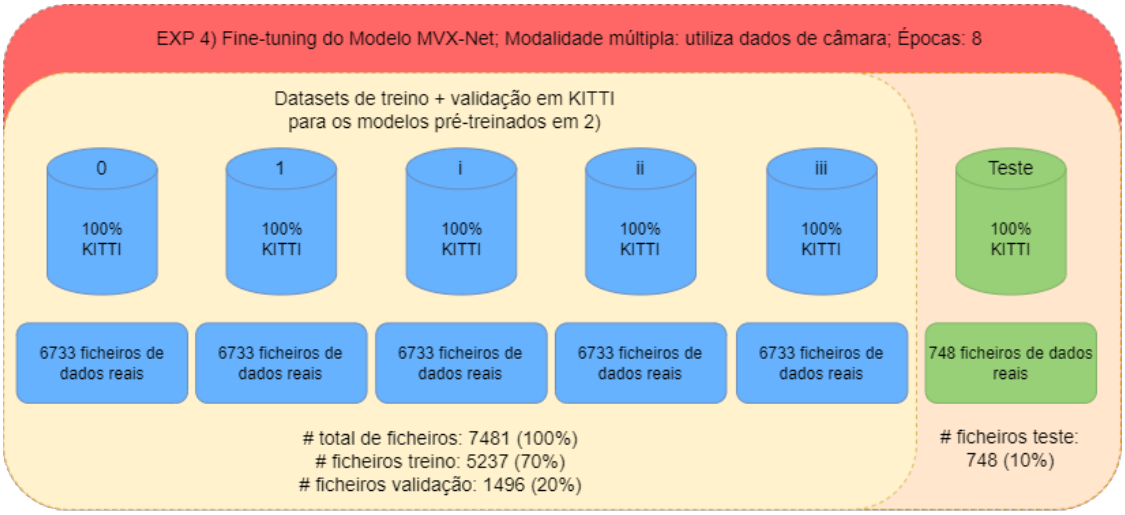


Figura 33: Experiência número 4: Treino, validação e teste dos modelos MVX-Net pré-treinados da experiência 2) no conjunto de dados KITTI. A diferença entre cada uma das 5 partes prende-se com a origem do modelo pré-treinado obtido na experiência 2.

4

IMPLEMENTAÇÃO E PREPARAÇÃO DO CENÁRIO DE EXECUÇÃO

Neste capítulo é apresentado o processo de preparação dos conjunto de dados Kitti e virtual KITTI para treino, validação e teste. Para tal, estudou-se o formato do conjunto de dados KITTI de forma a reformatar o conjunto de dados virtual KITTI para um formato igual ao KITTI. Só assim se pôde treinar, validar e testar este novo conjunto de dados personalizado na framework mmdetection3D, como se se tratasse do conjunto de dados KITTI. Desde analisar a ordem dos elementos presentes no conjunto de dados real, verificar a localização da coordenada de origem nos ficheiros de dados e até a completa reformatação de um conjunto de dados personalizado, bem como a sua integração em diferentes conjunto de dados híbridos (parte dos dados provenientes do KITTI, parte provenientes do virtual KITTI - conjunto de dados i, ii e iii), todos os seguintes tópicos serão abordados por ordem.

4.1 Estrutura do conjunto de dados kitti

O conjunto de dados KITTI possui a seguinte disposição de parâmetros.

Os dados são armazenados em ficheiros de texto, *labels*, que são organizadas desde 000000.txt até 007480.txt. Abaixo seguem-se alguns exemplos da forma como os dados são organizados nos ficheiros de texto.

car 0.00 0 -1.58 587.01 173.33 614.12 200.12 1.65 1.67 3.64 -0.65 1.71 46.70 -1.59

cyclist 0.00 0 -2.46 665.45 160.00 717.93 217.99 1.72 0.47 1.65 2.45 1.35 22.10 -2.35

pedestrian 0.00 2 0.21 423.17 173.67 433.17 224.03 1.60 0.38 0.30 -5.87 1.63 23.11 -0.03

4.2 Estrutura do conjunto de dados virtual kitti versão 1.3.1

O conjunto de dados virtual KITTI dispõe de 9 ficheiros comprimidos, nomeadamente:

- vkitti_2.0.3_rgb.tar - conjunto de dados de imagens RGB
- vkitti_2.0.3_depth.tar - conjunto de dados depth ou de profundidade
- vkitti_2.0.3_classSegmentation.tar - conjunto de dados de segmentação de classes

Nº Elementos	Posição	Nomes dos parâmetros	Descrição	Tipo	Gama de valores	Exemplo
1	0	Nome da classe	A classe do objeto	Cadeia de carateres	N/A	Pessoa, Carro
1	1	Truncação	Quanto é que o objeto saiu das fronteiras	Flutuante	0.0, 0.1	0.0
1	2	Oclusão	Estado de oclusão (0=totalmente visível, 1=parcialmente visível, 2=amplamente oculto, 3=totalmente oculto)	Inteiro	[0,3]	2
1	3	Alpha	Ângulo de observação do objeto	Flutuante	[-pi,pi]	0.146
4	4,5, 6,7	Coordenadas da bounding box	Localização do objeto na imagem	Flutuante	[0, image_width], [0, image_height], [top left, image_width], [bot right, image_height]	100 120 180 160
3	8, 9, 10	Dimensão 3D	Altura, Largura, Comprimento (metros)	Flutuante	N/A	1.65, 1.67, 3.64
3	11,12, 13	Localização	Localização 3D em coordenadas de câmara (metros)	Flutuante	N/A	-0.65,1.71,46.7
1	14	Rotação_y	Rotação ry em torno do eixo Y em coordenadas de câmara (metros)	Flutuante	[-pi,pi]	-1.59

Tabela 8: Parâmetros do KITTI

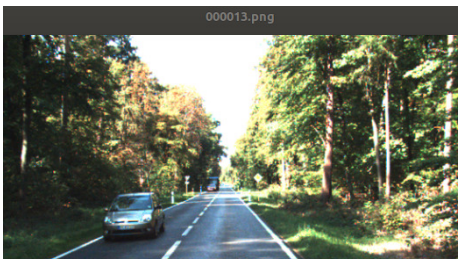
- vkitti_2.0.3_instanceSegmentation.tar - conjunto de dados de segmentação de instâncias
- vkitti_2.0.3_textgt.tar.gz - anotações ou *ground truths*
- vkitti_2.0.3_forwardFlow.tar - conjunto de dados de fluxo dianteiro
- vkitti_2.0.3_backwardFlow.tar - conjunto de dados de fluxo traseiro
- vkitti_2.0.3_forwardSceneFlow.tar - conjunto de dados óptico/fluxo de cena dianteiro
- vkitti_2.0.3_backwardSceneFlow.tar - conjunto de dados óptico/fluxo de cena traseiro

Para as experiências a realizar, apenas foi necessário utilizar o conjunto de dados RGB e as respetivas anotações. Além disso, os dados de simulação do virtual KITTI foram obtidos a partir de cinco localizações diferentes, sendo que para cada localização existem dez variações diferentes de perspetiva e ambiente. Ou seja, após descarregar este conjunto de dados, verifica-se uma organização de ficheiros onde existem cinco pastas com os nomes das cinco localizações, 01, 02, 06, 18 e 20. Para cada uma delas existem dez ficheiros representantes das variações das condições em como os dados foram obtidos: *15-deg-left, 15-deg-right, 30-deg-left, 30-deg-right, clone, fog, morning, overcast, rain e sunset*. Os dados são representados com cinco dígitos em vez de seis, como o KITTI, e vão desde 00000.txt até 00446.txt, 00892.txt ou 00223.txt dependendo de qual das dez variantes de ficheiros é analizada. As anotações possuem o seguinte formato:

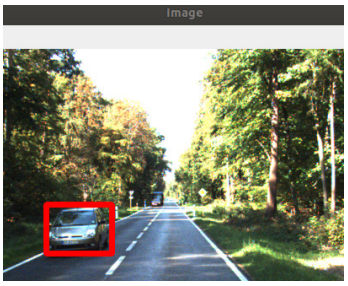
Nº elementos	Posição	Nomes dos parâmetros	Descrição	Tipo	Gama de valores	Exemplo
1	0	Frame	Índice do ficheiro	Inteiro	N/A	3
1	1	ID objeto	Identificador do objeto	Inteiro	N/A	5
1	2	Nome da classe	A classe do objeto	Cadeia de carateres	Car, Van, DontCare	Car
1	3	Truncação	Quanto é que o objeto saiu das fronteiras	Inteiro	[0,1,2, DontCare] (discreto)	1
1	4	Oclusão	Estado de oclusão	Inteiro	[0,1,2, DontCare] (discreto)	DontCare
1	5	Alpha	Ângulo de observação do objeto	Flutuante	[-pi, pi] (contínuo)	1.3
4	6,7,8,9	Coordenadas da bounding box	Esquerda, Cima, Direita, Baixo em Coordenadas de Píxel	Flutuante	[0,image_width], [0,image_height], [top left, image_width], [bot right, image_height]	132 155 177 172
3	10,11,12	Dimensão 3D	Largura, Altura, Comprimento (metros)	N/A	N/A	1.32 2.67 3.12
3	13,14,15	Localização	Localização em 3D das coordenadas de câmara (metros)	N/A	N/A	-0.26 0.65 1.76
1	16	Rotação em Y	Rotação em torno do eixo Y em coord. de câmara	Flutuante	[-pi, pi] (contínuo)	-1.54
1	17	Rotação em X	Rotação em torno do eixo X em coord. de câmara	Flutuante	[-pi, pi] (contínuo)	1.85
1	18	Rotação em Z	Rotação em torno do eixo Z em coord. de câmara	Flutuante	[-pi, pi] (contínuo)	1.45
1	19	Rácio de truncatura	Indicador de truncatura em contínuo	Flutuante	[0,1] (contínuo)	0.1
1	20	Rácio de ocupação	Indicador de ocupação em contínuo	Flutuante	[0,1] (contínuo)	0.8
1	21	Anotação de origem	Anotação original removendo a regra do DontCare	Cadeia de carateres	N/A	Van
1	22	Movendo	Indica se o objeto está em repouso ou movimento	Inteiro	[0,1] (discreto)	1
1	23	Modelo	Nome do modelo em 3D para renderizar o objeto	Cadeia de carateres	N/A	Sedan
1	24	Cor	Nome da cor do objeto	Cadeia de carateres	N/A	Blue

Tabela 9: Parâmetros do virtual KITTI

Depois de se ter comparado os dois conjunto de dados, foi necessário identificar em ambos onde se localizava as coordenadas de origem em cada imagem, bem como codificar um ficheiro de código para extraír os dados do virtual KITTI que pertencem ao KITTI, colocando-os na mesma ordem que no conjunto de dados real. Para isso, um dos truques usados foi desenhar um retângulo vermelho com as coordenadas de uma amostra do conjunto de dados do virtual KITTI e verificar se a predição (o carro neste caso) se encontra dentro



(a) Uma dada amostra do conjunto de imagens sem deteções por cima.



(b) A mesma amostra do conjunto de imagens em que foi desenhado um retângulo vermelho a envolver o objeto de interesse - carro.

Figura 34: Com a biblioteca cv2 é possível desenhar retângulos dando apenas as coordenadas x,y de quatro pontos. Se uma delas for 0,0 um dos vértices do retângulo encontrar-se-à na origem do referencial, que corresponde ao canto superior esquerdo.

do retângulo que atua como "bounding box", indicadora que o algoritmo o detetou, como indicado na figura 34b.

4.3 Localizar as coordenadas de origem

Antes de reformatar o conjunto de dados virtual KITTI para um formato igual ao do KITTI, teve de se verificar se as coordenadas de origem dos ficheiros de imagem de ambos os conjunto de dados coincidiam. Se esta etapa fosse ignorada e caso o virtual KITTI usasse uma origem do referencial diferente do do KITTI, as anotações do virtual KITTI ao serem justapostas com as do KITTI seriam incongruentes e originariam deteções nos sítios errados na imagem por causa de um sistema de coordenadas diferente. Isso levaria a que as experiências até então efetuadas seriam invalidadas. Então, para isso, selecionou-se uma imagem do KITTI e do virtual KITTI onde só havia um veículo. De seguida, desenhou-se um retângulo com essas coordenadas através de uma função da biblioteca cv2. Em ambos os casos, verificou-se que os retângulos ou *bounding boxes* criadas envolviam os veículos tanto no KITTI como no virtual KITTI. Depois, colocou-se uma dessas coordenadas com os valores 0,0 para x,y e constatou-se que o retângulo desenhado possuía um dos vértices no canto superior esquerdo da imagem. Para confirmar, verificou-se que no website do virtual KITTI, versão 1.3.1, o canto superior esquerdo corresponde às coordenadas da origem e assim se provou que ambos os conjunto de dados possuem as mesmas coordenadas de origem do referencial de *bounding boxes*. O teste foi possível efetuar através do ficheiro draw_rectangle.py, descrito em anexo em 1.

Concluiu-se assim que não foi necessário recalcular as coordenadas das *bounding boxes* do virtual KITTI uma vez que estas possuem as mesmas coordenadas de origem que o conjunto de dados KITTI.

4.4 Reformatar o conjunto de dados

Depois de assegurar que as coordenadas de origem eram as mesmas, tornou-se necessário reformatar o conjunto de dados virtual KITTI de modo a que os parâmetros fossem os mesmos e tivessem a mesma ordem que no conjunto de dados KITTI. Para isso, codificou-se o ficheiro vkitti_2_kitti_conv_ficheiro.py, descrito em

anexo em 2, que utiliza como entrada um ficheiro de anotações do virtual KITTI, gerando um ficheiro de saída, também de anotações, apenas com os parâmetros do conjunto de dados Kitti, de acordo com as tabelas acima descritas. Esses parâmetros são os que possuem na tabela do conjunto de dados do virtual KITTI as posições de 2 a 16, excluindo a 1, ou ID do objeto. O parâmetro da posição zero também é guardado. Isto deve-se ao facto de o ficheiro de texto resultante possuir todas as anotações em formato KITTI de todos os ficheiros virtual KITTI. Tornou-se assim necessário originar um segundo ficheiro, label_creation_files.py, descrito em anexo em 3, que aloca as anotações do mesmo ficheiro, descrito pelo parâmetro na posição 0, num ficheiro correspondente em formato KITTI, repetindo o processo até todas as anotações do virtual KITTI estarem nos respetivos ficheiros numerados da mesma maneira que no KITTI (000000.txt até 020140.txt). Por fim, para ordenação de ficheiros recorreu-se ao ficheiro order_img_files.py, também descrito em anexo, em 4.

4.5 Estruturar os conjunto de dados híbridos

Após se obterem os conjunto de dados sintéticos no formato KITTI, organizou-se cinco conjunto de dados diferentes com dados de câmara para as seguintes experiências a realizar no modelo MVX-Net e Fcos3D descritas nas figuras 30 e 37. Para todas utilizaram-se 7481 ficheiros de dados, 6733 dos quais para treino mais validação. Os restantes 748 ficheiros de dados foram reservados para teste que são retirados do conjunto de dados KITTI. Deste modo, dos 6733 ficheiros, nas experiências 0 e 1 estes foram obtidos dos conjunto de dados sintético e real, respetivamente. Para as restantes experiências i, ii e iii os dados tiveram de ser obtidos de forma aleatória e em partições de 75/25%, 50/50% e 25/75% de conjunto de dados real e sintético. Para isso, desenvolveu-se o ficheiro setup_conjunto de dados.py, decrito em anexo em **??**. O ficheiro trata de escolher aleatoriamente quais os ficheiros que vão para treino, validação e teste, gerando os ficheiros train.txt, val.txt e test.txt dentro da pasta ImageSets, a pasta cujos conteúdos são lidos pelo modelo a treinar ou a testar para obter a sua precisão. De seguida, para cada experiência, são coletadas diferentes proporções de dados reais e sintéticos e os ficheiros de dados e a suas contrapartes de anotações são copiados para os respetivos ficheiros de destino. A explicação do ficheiro encontra-se detalhada em anexo. A falha tecnológica está do lado do MMDetection3D. Os autores reconheceram o problema e prontamente anunciaram que o erro das métricas irem todas a zero irá desaparecer para o modelo MVX-Net, não sendo necessário acrescentar manualmente o prefixo pts_ do lado do utilizador final.

4.6 Configuração adicional para testar com sucesso o modelo mvx-net

Após ter sido executado o teste do modelo MVX-NET depois de treinado no conjunto de dados KITTI, verificou-se que este apresentava como precisão média (ou mAP) o valor zero. Todos os parâmetros possuiam uma classificação de zero o que não era normal. A solução para este problema consistiu em alterar o nome das camadas de neurónios que constituem o modelo MVX-NET. Para isso, desenvolveu-se o ficheiro convert_model_parameters.py, descrito em anexo em 6, que carregava o modelo mais recente guardado pelo processo de treino num ficheiro pth - latest.pth ou epoch_40.pth, visto que se efetuou quarenta épocas. Para isso, utilizou-se a função torch.load(path) da biblioteca torch que, neste caso, carrega o path ou diretório do ficheiro .pth codificador dos parâmetros (pesos e offsets) do modelo treinado. De seguida, efetuou-se uma

transposição das várias camadas do modelo com a função torch.transpose e, em cada uma delas, adicionou-se o prefixo pts_ antes de se salvar o modelo com a função torch.save(new_model, "adjusted_latest.pth").

4.7 Configuração adicional para treinar modelos fcos3d no conjunto de dados kitti

Também surgiram problemas ao tentar treinar o modelo Fcos3D, que só suporta dados de câmara, no conjunto de dados KITTI. Foram necessárias alterar algumas configurações presentes no ficheiro kitti-mono-3d.py de forma a facultar o treino. Das alterações que se fizeram, vêm as seguintes:

- Alteração dos parâmetros presentes na normalização da média de configuração de imagem: o primeiro e terceiro parâmetros permutam entre si
- Alteração do desvio-padrão da normalização da configuração da imagem: os três valores de desvio-padrão passam para zero
- Substituição do parâmetro dicionário LoadImageFromFileMono3D por t no pipeline de treino
- A escala de imagem passou a assumir os valores (1242, 375)
- O fator de escala passou a assumir um fator de escala 1.0
- Os nomes das classes a detetar passaram apenas a: Pedestrian, Cyclist e Car

O ficheiro final de configuração usado para treinar o Fcos3D encontra-se em anexo.

4.8 Configurações adicionais para efetuar o fine-tuning dos modelos

Para efetuar os fine-tuning dos modelos obtidos nas experiências 1) e 2) foi necessário efetuar algumas alterações no código base.

```

'--load-from', action='store_true', help='the checkpoint file to load from')
    parser.add_argument('--auto-resume',
        action='store_true',
        help='resume from the latest checkpoint automatically')


```

Listing 4.1: Incluir no ficheiro train.py dentro de tools

```
if args.load_from is None: #Original: not None
    cfg.load_from = args.load_from
```

Listing 4.2: Editar no ficheiro train.py dentro de tools

4.9 Experiências com conjunto de dados de lidar sintético e radar

Foram também efetuadas experiências com dados de lidar obtidos em simulação provenientes do conjunto de dados virtual KITTI dataset 3D (Engelmann et al., 2017). Contudo, dado o número insignificativo de amostras, 90, em comparação com as 6733 amostras necessárias para se efetuar experiências sob as mesmas condições que as anteriores, não se registou neste documento o resultado destas. Os últimos commits efetuados na plataforma tinham sido feitos há mais de 5 anos e não havia qualquer feedback por parte do autor. Nem tampouco havia outros conjuntos de dados disponíveis de lidar sintético tais que servissem para a realização do conjunto de experiências abordadas nos capítulos anteriores. Além disso, não existem na literatura conjunto de dados que envolvessem apenas dados de radar para deteção de veículos. Por isso, apenas foram registadas as experiências que envolvessem dados de câmara para os modelos Fcos3D e MVX-Net dado a falta de dados sintéticos de lidar e radar.

5

RESULTADOS

Neste capítulo pretende-se comprovar através dos resultados das experiências propostas o efeito do treino de redes de modalidade única (Fcos3D) e multi-modal (MVX-Net). Para isso, foram analisadas os parâmetros bbox 2D, bbox 3D (consoante se trabalha ora com deteções 2D ora com deteções 3D) e orientação "aos"para cada tipo de dificuldade de deteção de veículos - easy, moderate and hard, ou fácil, médio e difícil, respetivamente. Analisou-se a diferença entre o resultado dos testes após o treino inicial e após o fine-tuning para perceber se existe alguma melhoria significativa em todos ou alguns dos parâmetros, para cada tipo de conjunto de dados. Também se verificou as diferenças de resultados para cada um dos cinco conjunto de dados e para cada um dos modelos. Os valores que se encontram apenas em bold são os resultados das classificações das categorias treinadas nos cinco conjunto de dados descritos no capítulo anterior que tiveram melhor desempenho que os valores citados pela literatura, pelo mesmo modelo. Já os valores em bold e com fundo verde correspondem aos resultados de fine-tuning com classificação superior a 40% que ultrapassaram os valores de literatura para um dado modelo.

5.1 Desempenho dos modelos fcos3d e mvx-net na literatura

Nas tabelas abaixo podemos verificar o desempenho dos modelos Fcos3D e MVX-Net reportados na literatura:

Fcos3D Test Set AP40 results overall	**AP40 @easy**	**AP40 @moderate**	**AP40 @hard**
bbox 2D	53.3167	44.9526	41.0082
bbox 3D	3.3643	2.2743	1.9977
orientação	43.19	36.75	33.06

MVX-Net Test Set AP40 results overall	**AP40 @easy**	**AP40 @moderate**	**AP40 @hard**
bbox 2D	89.1262	81.8567	78.3773
bbox 3D	81.0503	70.6455	65.8158
orientação	78.84	71.71	68.64

Tabela 10: Literatura: Resultados obtidos após teste no conjunto de dados KITTI dos algoritmos pré-treinados Fcos3D e MVX-Net (Kumar, 2022).

5.2 Modelo fcos3d

Test Set AP40 results overall	AP40 @easy	AP40 @moderate	AP40 @hard	Fine tuned results overall	@easy	@moderate	@hard
bbox 2D	2.9115	2.0271	1.7180	**bbox 2D**	**70.0579**	**61.3488**	**56.6686**
bbox 3D	0.0095	0.0046	0.0053	**bbox 3D**	5.0721	3.8426	3.5244
orientação	2.86	1.95	1.63	**orientação**	**66.55**	**57.91**	**53.31**

Tabela 11: EXP 1.0 e 3.0: Resultados obtidos após teste do modelo Fcos3D com dados puramente sintéticos. A tabela à direita, a do fine-tuning, corresponde aos resultados do modelo Fcos3D treinado em dados sintéticos quando colocado sobre dados puramente reais após o primeiro teste, e testado novamente depois de 8 épocas de treino em dados desta vez 100% reais.

Test Set AP40 results overall	AP40 @easy	AP40 @moderate	AP40 @hard	Fine tuned results overall	@easy	@moderate	@hard
bbox 2D	**71.2840**	**64.8770**	**61.5631**	**bbox 2D**	**78.6175**	**67.9925**	**63.8979**
bbox 3D	12.3638	8.9829	8.2792	**bbox 3D**	7.9231	4.7890	4.2356
orientação	**67.11**	**61.06**	**57.92**	**orientação**	**72.90**	**62.52**	**58.78**

Tabela 12: EXP 1.1 e 3.1: Resultados obtidos do teste e fine tuning do modelo Fcos3D com dados puramente reais.

Test Set AP40 results overall	AP40 @easy	AP40 @moderate	AP40 @hard	Fine tuned results overall	@easy	@moderate	@hard
bbox 2D	**71.5550**	**64.1831**	**60.9450**	**bbox 2D**	**75.9532**	**69.7813**	**65.6015**
bbox 3D	12.3772	8.8884	8.3489	**bbox 3D**	7.6324	4.9366	4.6415
orientação	**67.57**	**60.40**	**57.24**	**orientação**	**72.19**	**66.02**	**61.81**

Tabela 13: EXP 1.i e 3.i: Resultados obtidos no teste do modelo Fcos3D com dados reais e sintéticos, proporção 75/25, respetivamente, e fine tuning com dados reais. Os resultados de desempenho do modelo treinado neste conjunto híbrido de dados para os parâmetros de bbox 2D e 3D e para a dificuldade moderada e difícil do KITTI são superiores aos resultados obtidos pelo mesmo modelo quando é treinado em conjuntos de dados 100% reais, depois de ser efetuada a etapa de "fine-tuning"em ambos os casos.

Test Set AP40 results overall	AP40 @easy	AP40 @moderate	AP40 @hard	Fine tuned results overall	@easy	@moderate	@hard
bbox 2D	**69.5597**	**63.5842**	**59.8291**	**bbox 2D**	**72.5382**	**64.6395**	**60.6912**
bbox 3D	11.6705	8.2391	7.4217	**bbox 3D**	2.3783	2.2498	2.0754
orientação	**64.08**	**58.52**	**54.73**	**orientação**	**68.55**	**60.40**	**56.72**

Tabela 14: EXP 1.ii e 3.ii: Resultados obtidos no teste do modelo Fcos3D com dados reais e sintéticos, proporção 50/50, respetivamente, e fine tuning com dados reais.

5.3 Modelo mvx-net

Test Set AP40 results overall	AP40 @easy	AP40 @moderate	AP40 @hard	Fine tuned results overall	@easy	@moderate	@hard
bbox 2D	**64.7780**	**56.8055**	**52.3878**	**bbox 2D**	**68.7099**	**60.0393**	**55.6325**
bbox 3D	7.6071	4.9195	4.3524	**bbox 3D**	7.7883	5.1107	4.6113
orientação	**56.94**	**49.81**	**45.77**	**orientação**	**55.58**	**50.39**	**46.41**

Tabela 15: EXP 1.iii e 3.iii: Resultados obtidos no teste do modelo Fcos3D com dados reais e sintéticos, proporção 25/75, respetivamente, e fine tuning com dados reais.

Test Set AP40 results overall	AP40 @easy	AP40 @moderate	AP40 @hard	Fine tuned results overall	@easy	@moderate	@hard
bbox 2D	14.0071	8.8206	7.7761	**bbox 2D**	87.1355	80.5850	77.1994
bbox 3D	1.3891	1.0404	1.0044	**bbox 3D**	72.4752	63.1295	58.2946
orientação	9.07	5.68	5.00	**orientação**	66.85	59.57	57.17

Tabela 16: EXP 2.0 e 4.0: Resultados obtidos do teste do modelo MVX-Net cujo treino fora efetuado com dados puramente sintéticos. A tabela à direita, a do fine-tuning, corresponde aos resultados do modelo MVX-Net treinado em dados sintéticos quando posteriormente colocado em treino sobre dados puramente reais após o primeiro teste, e testado novamente depois de 8 épocas de treino em dados desta vez 100% reais.

Test Set AP40 results overall	AP40 @easy	AP40 @moderate	AP40 @hard	Fine tuned results overall	@easy	@moderate	@hard
bbox 2D	**90.1767**	**85.1476**	**82.0891**	**bbox 2D**	**92.1822**	**85.1069**	**82.9483**
bbox 3D	**87.1855**	**77.8197**	**74.7331**	**bbox 3D**	**84.4535**	**76.4712**	**73.7395**
orientação	**83.66**	**76.96**	**73.91**	**orientação**	**84.87**	**75.91**	**73.64**

Tabela 17: EXP 2.1 e 4.1: Resultados obtidos do teste e fine tuning do modelo MVX-Net com dados puramente reais.

Test Set AP40 results overall	AP40 @easy	AP40 @moderate	AP40 @hard	Fine tuned results overall	@easy	@moderate	@hard
bbox 2D	86.8166	78.5973	75.8453	**bbox 2D**	**91.4362**	**87.5891**	**84.8230**
bbox 3D	**82.8675**	**70.7324**	65.7611	**bbox 3D**	**84.5423**	**75.9541**	**71.8603**
orientação	66.51	60.05	57.95	**orientação**	**81.01**	**76.18**	**73.40**

Tabela 18: EXP 2.i e 4.i: Resultados obtidos no teste do modelo MVX-Net com dados reais e sintéticos, proporção 75/25, respetivamente, e fine tuning com dados reais. Os resultados de desempenho do modelo MVX-Net treinado neste conjunto híbrido de dados para os parâmetros de bbox 2D e 3D e para a dificuldade moderada e difícil do KITTI são superiores aos resultados obtidos pelo mesmo modelo quando é treinado em conjuntos de dados 100% reais, depois de ser efetuada a etapa de "fine-tuning"em ambos os casos. A única exceção ocorre apenas no valor de desempenho do atributo "aos"em dificuldade difícil e depois da etapa de "fine-tuning"onde o modelo atinge um resultado ligeiramente superior (diferença de décimas) com o conjunto de dados real do que com o conjunto de dados híbrido presente na experiência 4.i.

Test Set AP40 results overall	AP40 @easy	AP40 @moderate	AP40 @hard	Fine tuned results overall	@easy	@moderate	@hard
bbox 2D	84.0669	78.1533	74.5966	**bbox 2D**	**89.6567**	**82.7058**	**80.0161**
bbox 3D	78.8856	68.7473	63.4153	**bbox 3D**	**83.1232**	**73.9982**	**70.2474**
orientação	74.51	67.91	64.42	**orientação**	**82.38**	**73.79**	**71.41**

Tabela 19: EXP 2.ii e 4.ii: Resultados obtidos no teste do modelo MVX-Net com dados reais e sintéticos, proporção 50/50, respetivamente, e fine tuning com dados reais.

Test Set AP40 results overall	AP40 @easy	AP40 @moderate	AP40 @hard	Fine tuned results overall	@easy	@moderate	@hard
bbox 2D	81.8081	70.9454	66.8487	**bbox 2D**	**91.4643**	**85.5164**	**81.8950**
bbox 3D	70.3306	56.6511	51.2416	**bbox 3D**	**82.8697**	**73.3320**	**68.6929**
orientação	56.74	49.95	46.67	**orientação**	74.50	68.82	65.47

Tabela 20: EXP 2.iii e 4.iii: Resultados obtidos no teste do modelo MVX-Net com dados reais e sintéticos, proporção 25/75, respetivamente, e fine tuning com dados reais.

Analisando os resultados obtidos, conclui-se que é possível atingir-se resultados de classificação superiores aos de literatura se se treinar o modelo Fcos3d com um conjunto de dados composto por pelo menos 25% de dados reais. É ainda possível obter resultados de classificação superiores aos de literatura para o modelo Fcos3d mesmo para conjunto de dados de treino puramente sintéticos se o modelo for posteriormente submetido a um fine tuning num conjunto de dados 100% real, bastando apenas 8 épocas de treino adicional para o fine-tuning.

Quanto ao modelo MVX-Net, verifica-se que é possível obter resultados superiores aos de literatura em todos os pós-treinos de fine-tuning, após 8 épocas, com a exceção do treino inicial num conjunto de dados 100% sintético. Além disso, é possível fazê-lo também com o treino em 40 épocas num conjunto de dados 100% real, o que torna uma mais valia o uso deste tipo de dados sinteticamente gerados na falta de dados reais.

Quanto ao modelo Fcos3d, verifica-se que este obtém melhores resultados (em bbox e aos) após fine-tuning que antes. O parâmetro 3d parece piorar quando efetuado o fine-tuning e o parâmetro aos não melhora nem piora quando o algoritmo é treinado num conjunto de dados 3.iii, com 75% dos dados sintéticos e 25% dos dados reais, e é efetuado o seu fine-tuning num conjunto de dados com 100% dos dados reais.

Quanto ao modelo MVX-Net, como seria de esperar, este obtém melhores resultados em termos de bbox, 3d e aos com treino em conjunto de dados com maior proporção de dados reais que sintéticos. Mas o mesmo não se verifica quando se efetua 8 épocas de fine-tuning. Entre os testes de fine-tuning, conclui-se que os resultados melhoram substancialmente na experiência com o conjunto de dados puramente sintético, havendo pequenas melhorias nos parâmetros bbox, 3d e aos nas experiências 4.i, 4.ii e 4.iii, para todos os níveis de classificação: easy, moderate e hard. No entanto, não se verificou diferenças significativas após o fine-tuning da experiência 4.1, nem entre o fine-tuning das experiências 4.i, 4.ii e 4.iii, para todas os níveis de classificação e parâmetros. Exceto apenas na experiência 4.iii em que o parâmetro aos é consideravelmente

inferior à experiência 1, em relação ao fine-tuning. Deste modo, prova-se também que os dados sintéticos por si só não conseguem treinar com a qualidade desejada os modelos de deteção usados nas experiências. Todavia, quando esses dados sintéticos são colocados em partições com maior percentagem de dados reais, os modelos conseguem atingir uma performance satisfatória e, em alguns casos, como os abordados nas tabelas das experiências, acima da performance atingida quando os conjuntos de dados de treino são constituídos por dados 100% reais.

Entre os dois modelos, o modelo multi-modal MVX-NET obtém melhores resultados em todos os parâmetros e níveis de classificação que o Fcos3d. Ambos, contudo, obtém as melhores classificações depois de fine-tuning em conjunto de dados puramente reais, 8 épocas, nas experiências 4.1 e 4.i, isto é, em conjunto de dados 100% dados reais e 75% dados reais com 25% dados sintéticos, respetivamente.

5.4 Teste dos modelos em produção

Nas figuras abaixo verifica-se uma correta deteção efetuada pelos modelos Fcos3d e MVX-Net:

Figura 35: Deteção efetuada pelo modelo Fcos3D, sob perspetiva 3D.

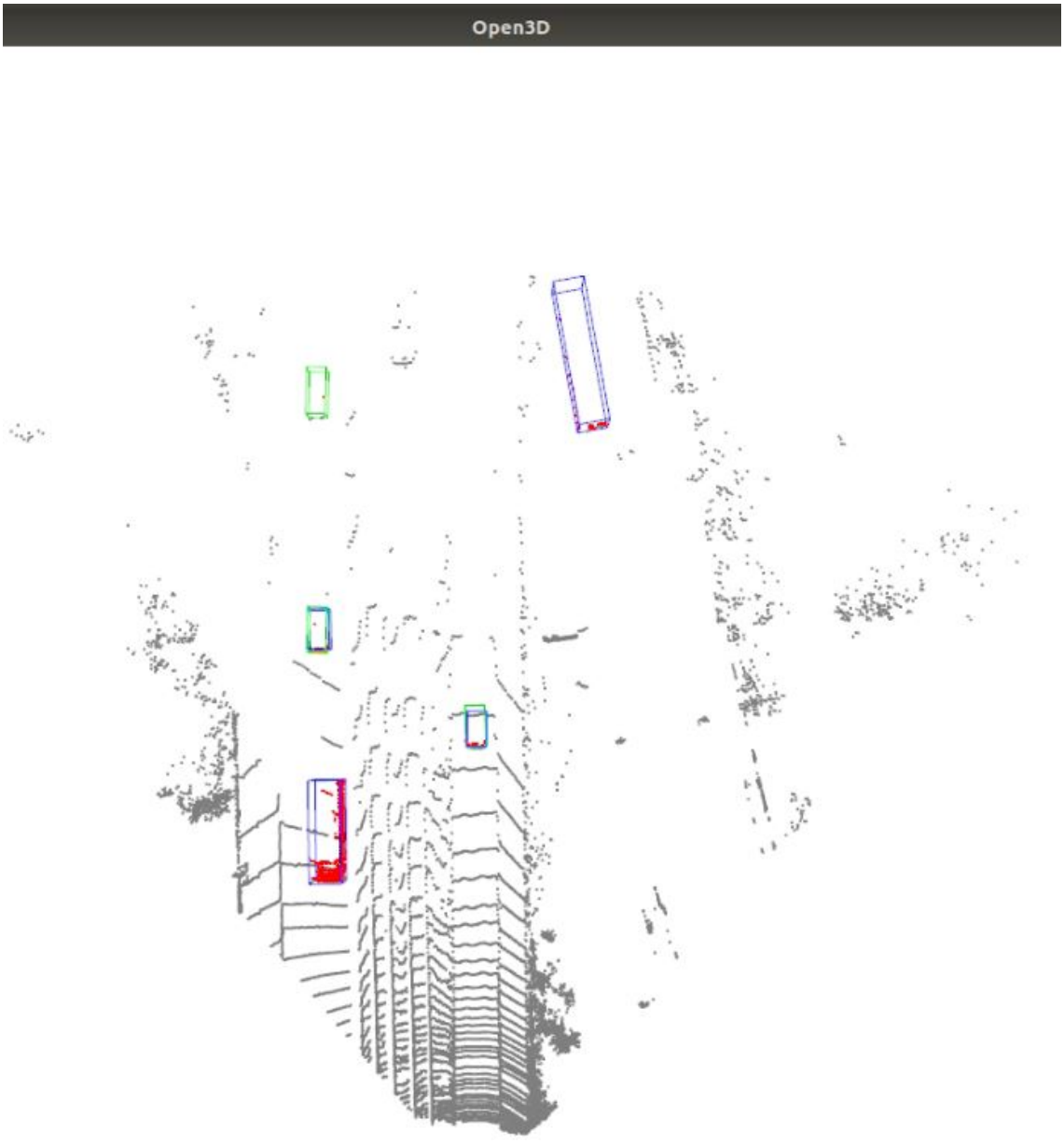


Figura 36: Deteção efetuada pelo modelo MVX-Net, sob perspetiva bird's-eye-view.

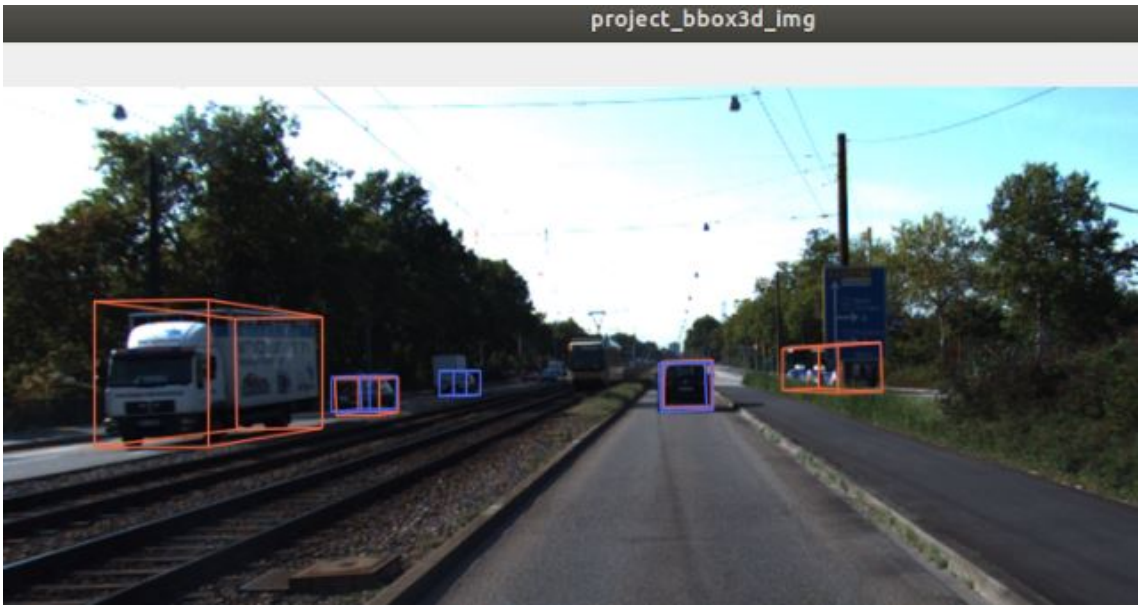


Figura 37: Deteção efetuada pelo modelo MVX-Net, sob perspetiva 3D.

6

CONCLUSÃO E TRABALHO FUTURO

Neste capítulo resumem-se as conclusões obtidas durante o tempo dedicado à elaboração desta dissertação e definem-se algumas linhas de trabalho e sugestões para trabalho futuro. O desenvolvimento desta dissertação teve como objetivo verificar o efeito do treino em conjunto de dados híbridos nos modelos de modalidade singular e multi-modal, e comparar o seu desempenho face a serem treinados em conjunto de dados com dados 100% reais. Deste modo, pretendeu-se averiguar a fiabilidade de dados sintéticos no treino de modelos quando existe falta de dados reais ou estes são demasiados dispendiosos para serem obtidos, por questões de tempo, dinheiro ou, possivelmente, autorizações relacionadas com a permissão de captação de dados em determinadas zonas. Assim sendo, começou-se por estudar os algoritmos do estado de arte de deteção de veículos, acabando-se por selecionar os modelos que apresentavam melhores resultados e que eram compatíveis com o conjunto de dados KITTI e com a framework mmdetection3D, atualizada mensalmente com novos conjunto de dados, modelos e ficheiros de treino. De entre os demais modelos, foi selecionado para alvo de experiências os modelos Fcos3D para câmara, modalidade singular e o MVX-Net para modalidade múltipla. Os resultados obtidos para o modelo HV_PointPillars (lidar) modalidade singular não foram significativos dada a falta de dados de lidar sintéticos pelo que não fora considerado as experiências do mesmo. A escolha do conjunto de dados KITTI teve a ver com a existência de contrapartes sintéticas como o virtual KITTI (câmara) que permitiram a formação dos conjunto de dados híbridos, para além do facto de o KITTI ser o conjunto de dados mais popular para treino de detetores de veículos. O mmdetection3d foi escolhido por ser a framework com maior diversidade de modelos e conjunto de dados compatíveis, assim como disponibiliza diversas opções de treino/teste e permite diversas opções de configuração extra como o hiperparâmetro de learning rate, ou ritmo de aprendizagem. Depois de selecionados os modelos, procedeu-se à formulação dos vários conjunto de dados. Para tal, começou-se por descarregar tanto o KITTI como o virtual KITTI. De notar que também se tentou efetuar testes com dados de lidar sintético, contudo, os dados obtidos (90 ficheiros de nuvens de pontos) eram insuficientes para o treino, por isso, não se efetuaram testes envolvendo esse tipo de dados. De seguida, elaborou-se ficheiros que permitem a criação dos conjunto de dados híbridos de forma aleatória para efetuar as experiências retratadas, após reformatar o conjunto de dados virtual KITTI, de forma a ficar com o mesmo formato que o KITTI, e criar as respetivas labels para as imagens reais ou sintéticos. Também se teve de efetuar algumas alterações no modelo como acrescentar o prefixo "pts_"nas camadas de neurónios do modelo MVX-Net para permitir que o teste devolvesse resultados não nulos, uma falha tecnológica do lado da estrutura de software MMDetection3D que entretanto fora reconhecida pelos autores e encontra-se em fase de resolução no momento desta escrita. Por fim, depois da obtenção dos resultados e descrição destes no capítulo anterior, concluiu-se que ambos os modelos superaram em termos

de precisão os resultados registados na literatura quando submetidos a um fine tuning. O modelo MVX-NET de modalidade múltipla, em particular, obteve melhores resultados que o Fcos3D de modalidade singular. Finalmente, a obtenção de resultados muito próximos no fine-tuning de ambos os modelos quando treinados com um conjunto de dados inicial de treino com dados 100% reais e conjunto de dados com 75% de parte real e 25% parte sintético permite comprovar que os dados sintéticos poderão ser usados em treino na falta de dados reais, mantendo a precisão em todas as modalidades de deteção e parâmetros no KITTI, desde que haja três vezes mais dados reais que sintéticos e dados reais afinados para a etapa de "fine-tuning".

As conclusões acima levam assim à formulação das seguintes hipóteses a serem investigadas em trabalhos futuros:

- Modelos multi-modais, em geral, obtém melhores resultados num todo que modelos de modalidade única, quando ambos são treinados num conjunto de dados misto.
- conjunto de dados de treino mistos poderão em certos casos originar melhores resultados que conjunto de dados puramente reais, desde que haja maior proporção de dados reais que de sintéticos no primeiro caso (idealmente 75/25 % de dados reais/sintéticos) e se, de seguida, for efetuado nos modelos pré-treinados o fine-tuning em conjuntos de dados puramente reais, com a mesma quantidade de ficheiros para treino, validação e teste.

Com base nestes resultados, surge também outra linha de investigação a seguir: extrapolar se se verifica as mesmas conclusões quando os dados são de nuvens de pontos, apenas, e não de câmara. Ou, ainda, uma mistura de dados de nuvens de pontos e de câmara (RGB).

Por fim, poder-se-ia também efetuar um estudo não só com diferentes conjunto de dados, mas também com diferentes modelos e, ainda, diferente número de épocas de treino e fine-tuning e/ou diferentes proporções/quantidades de dados reais e sintéticos para conjunto de dados de treino, validação e teste.

Anexos

```
# Import cv2 module
import cv2

# Path to location of image in local storage
location = r'/home/mola/data/sintético_kitti_1_3_1/training/image_2/0001/15-deg
    -left/00000.png'

# Read the image in default mode
image = cv2.imread(location)

# Name of window
window = 'Image'

#455.70 183.86 533.81 241.91 -- kitti
#897 171 984 236 -- vkitti

# Set the coordinates of top left corner of rectangle
top_left = (40, 100)

# Set the coordinates of bottom right corner of rectangle
bottom_right = (984, 200)

# Set the line color to red in BGR
colour = (0, 0, 255)

# Set line thickness to 5px
thickness = 5

# Draw a rectangle with red borders and thickness of 5 px
image = cv2.rectangle(image, top_left, bottom_right, colour, thickness)

# Display the image
cv2.imshow(window, image)

# Show the image until any key is pressed on keyboard
cv2.waitKey(0)
```

Listing 1: Ficheiro de desenho de um retângulo para localização da origem do referencial nas imagens KITTI e virtual KITTI: draw_rectangle.py

```


# Taking "gfg input file.txt" as input file
# in reading mode

#2 3 4 5 6-7-8-9 11-10-12 13-14-15 16 (vkitti data on kitti format and
    positioning)

#2 - either car, van or don't care
#21 - other alternatives: ...
```

```
10
# import required module
12 import os

# assign directory
14 directory = '..'
count = 0

16
# iterate over files in
18 # that directory
lst = os.listdir(directory)
20 lst.sort()
lst.pop()
22 reading_first_line=True
#/home/mola/data/vkitti3D-conjunto de dados/0002_data.txt
24 ##    with open("../"+lst[count], "r") as input:

26
while(count<len(lst)):
28

    with open("/home/mola/data/data_preparation_files/other/to_process_to_final
    /clone/0020_label.txt", "r") as input:
30

        # Creating "gfg output file.txt" as output
32      # file in write mode
        with open("0020_data.txt", "w") as output:
34

            # Writing each line from input file to
36          # output file using loop
            for line in input:
38              if(reading_first_line):
                    reading_first_line=False
40                  continue
                #output.write(line)
42              values = line.split()
                output.write(values[0])
44              output.write(' ')
                output.write(values[2])
46              output.write(' ')
                output.write(values[3])
48              output.write(' ')
                output.write(values[4])
50              output.write(' ')
                output.write(values[5])
52              output.write(' ')
                output.write(values[6])
54              output.write(' ')
                output.write(values[7])
56              output.write(' ')
                output.write(values[8])
```

```
                output.write(' ')
                output.write(values[9])
                output.write(' ')
                output.write(values[11])
                output.write(' ')
                output.write(values[10])
                output.write(' ')
                output.write(values[12])
                output.write(' ')
                output.write(values[13])
                output.write(' ')
                output.write(values[14])
                output.write(' ')
                output.write(values[15])
                output.write(' ')
                output.write(values[16])
                output.write('\n')
                #"""
            count+=1
            reading_first_line=True
```

Listing 2: Ficheiro de extração dos valores das labels do conjunto de dados virtual KITTI para KITTI: vkitti_2_kitti_conv_ficheiro.py.

```

import os
from tokenize import Double
import copy
#testing original: 0 1 2 3 4 5 6 7 8 9 10 11 33 38 40 53

#INPUT: filename_label.txt
#OUTPUT: .txt with all labels orderly

directory = "/home/mola/data/sintético_kitti_1_3_1/training/image_2"
count = 0
before_frame_value=0

lst = os.listdir(directory)
lst.sort()
first_time=True

#formated_count= str(count).zfill(6)

#while(count<len(lst)):

with open("filename_label.txt", "r") as input:

        # Creating "gfg output file.txt" as output
        # file in write mode

```

```
    while(count<len(lst)):
28
        with open("format_labels/"+lst[count][:-4]+".txt", "w") as output:
30
            # Writing each line from input file to
32          # output file using loop

34          if first_time:
                first_time=False
36          elif not first_time:
                final_line.pop(0)
38              output.write(" ".join(final_line))
                output.write('\n')
40
            for line in input:
42              #output.write(line)
                values = line.split()
44              final_line=copy.deepcopy(values)
                if int(values[0]) == int(before_frame_value):
46                  final_line.pop(0)
                    output.write(" ".join(final_line))
48                  output.write('\n')
                    before_frame_value = int(values[0])
50              #"""
                else:
52                  count+=1
                    before_frame_value = int(values[0])
54                  break
```

Listing 3: Ficheiro de criação de labels KITTI a partir das labels virtual KITTI: label_creation_files.py

```
2 import os
  from tokenize import Double
4
  #            "args": ["/home/mola/backup/mmdetection3d_backup/configs/mvxnet/
      BACKUPdv_mvx-fpn_second_secfpn_adamw_2x8_80e_kitti-3d-3class.py /home/mola/
      backup/mmdetection3d_backup/terminal_copy_after_train_test/EXP1/EXP1_a/
      epoch_40.pth --eval kitti"]
6
  #testing original: 0 1 2 3 4 5 6 7 8 9 10 11 33 38 40 53
8 #On launch file: "program": "${file}",

10 #label_2_for_test
  directory = "/home/mola/data/sintético_kitti_1_3_1/training/image_2"
12 dir="/home/mola/data/sintético_kitti_1_3_1/training/"
  count = 0
14
  lst = os.listdir(directory)
16 lst.sort()
```

```
folder = "image_2"
for count, filename in enumerate(lst):
    formated_count= str(count).zfill(6)
    dst = f"{formated_count}.png"
    src =f"{folder}/{filename}"  # foldername/filename, if .py file is outside
    folder
    dst =f"{folder}/{dst}"

    # rename() function will
    # rename all the files
    os.rename(dir+src, dir+dst)

#os.rename(src, dst)
```

Listing 4: Ficheiro de ordenação dos labels criados: order_image_files.py

```
import os
import shutil
import random

#Test subset - DESTINATION
dir_3='/home/mola/data/kitti/testing/image_2/'
dir_3_l='/home/mola/data/kitti/testing/label_2/'

#MIX - DESTINATION
dir_2='/home/mola/data/sintético_kitti_1_3_1/training/EXP1/iii/image_2/'
dir_2_l='/home/mola/data/sintético_kitti_1_3_1/training/EXP1/iii/label_2/'

#sintético - SOURCE
dir_1='/home/mola/data/sintético_kitti_1_3_1/training/image_2/'
dir_1_l='/home/mola/data/sintético_kitti_1_3_1/training/label_2/'

#REAL - SOURCE
dir_0='/home/mola/data/kitti/training/image_2/'
dir_0_l='/home/mola/data/kitti/training/label_2/'

#From train_val total amount of conjunto de dados samples
path = '../data/kitti/ImageSets/'
file_dir = path + 'trainval.txt'
file_train_dir = path + 'train.txt'
file_val_dir = path + 'val.txt'

def split_filelist(f_dir):

    filelist=[]
    with open(f_dir) as f:
        filelist=f.readlines()
        random.shuffle(filelist)
        s_train=filelist[:5237]      #70%
```

```
        s_val=filelist[5237:6733]    #20%
        s_test=filelist[6733:]       #10%
        return filelist, s_test

split_train, split_val, split_test = split_filelist(file_dir)
split_val = split_filelist(file_val_dir)


with open(os.path.join(os.path.dirname(path),'train'+'.txt'), 'w') as f:
    f.writelines(split_train)

with open(os.path.join(os.path.dirname(path),'val'+'.txt'), 'w') as f:
    f.writelines(split_val)

with open(os.path.join(os.path.dirname(path),'test'+'.txt'), 'w') as f:
    f.writelines(split_test)


def copy_files(files, src_dir_data, src_dir_label, dest_dir_data,
    dest_dir_label, fileformat):
    files=files+fileformat
    shutil.copy2(src_dir_data+files, dest_dir_data)
    anno = files[:-4]
    anno = anno + '.txt'
    shutil.copy2(src_dir_label+anno, dest_dir_label)

def mix_conjunto de dados(train_val, mix_ratio):
    if len(train_val) == 6733:                  # #Train + Val Files
        if mix_ratio == '1':
            real_subset=train_val               #100% real
        if mix_ratio == '0':
            sintético_subset=train_val            #100% sintético
        if mix_ratio == 'i':
            real_subset=train_val[:5050]        #75% real
            sintético_subset=train_val[5050:]     #25% sintético
        if mix_ratio == 'ii':
            real_subset=train_val[:3367]        #50% real
            sintético_subset=train_val[3367:]     #50% sintético
        if mix_ratio == 'iii':
            real_subset=train_val[:1683]        #25% real
            sintético_subset=train_val[1683:]     #75% real

        for rfiles in real_subset:
            copy_files(rfiles[:-1], dir_0, dir_0_l, dir_2, dir_2_l, '.png')

        for vfiles in sintético_subset:
            copy_files(vfiles[:-1], dir_1, dir_1_l, dir_2, dir_2_l, '.png')

    else:
        return False
```

```
    return True

for test_files in split_test:
    copy_files(test_files[:-1], dir_0, dir_0_l, dir_3, dir_3_l, '.png')
                # Camera files

split_train_val = split_train + split_val
print('\n')
print(mix_conjunto de dados(split_train_val, '0'))
```

Listing 5: Ficheiro de geração de conjunto de dados: Setup_conjunto de dados.py

```
import torch

path = '/home/mola/mmdetection3d/work_dirs/literature/fcos3d_normal.pth'
model = torch.load(path)

model['state_dict']['pts_middle_encoder.conv_input.0.weight'] = torch.transpose
    (model['state_dict']['pts_middle_encoder.conv_input.0.weight'],0,1)
model['state_dict']['pts_middle_encoder.conv_input.0.weight'] = torch.transpose
    (model['state_dict']['pts_middle_encoder.conv_input.0.weight'],1,2)
model['state_dict']['pts_middle_encoder.conv_input.0.weight'] = torch.transpose
    (model['state_dict']['pts_middle_encoder.conv_input.0.weight'],2,3)
model['state_dict']['pts_middle_encoder.conv_input.0.weight'] = torch.transpose
    (model['state_dict']['pts_middle_encoder.conv_input.0.weight'],3,4)

model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer1.0.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer1.0.0.weight'],0,1)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer1.0.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer1.0.0.weight'],1,2)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer1.0.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer1.0.0.weight'],2,3)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer1.0.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer1.0.0.weight'],3,4)

model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.0.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer2.0.0.weight'],0,1)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.0.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer2.0.0.weight'],1,2)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.0.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer2.0.0.weight'],2,3)
```

```
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.0.0.weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.0.0.weight'],3,4)

model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.1.0.weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.1.0.weight'],0,1)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.1.0.weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.1.0.weight'],1,2)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.1.0.weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.1.0.weight'],2,3)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.1.0.weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.1.0.weight'],3,4)

model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.2.0.weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.2.0.weight'],0,1)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.2.0.weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.2.0.weight'],1,2)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.2.0.weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.2.0.weight'],2,3)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.2.0.weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer2.2.0.weight'],3,4)

model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer3.0.0.weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer3.0.0.weight'],0,1)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer3.0.0.weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer3.0.0.weight'],1,2)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer3.0.0.weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer3.0.0.weight'],2,3)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer3.0.0.weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer3.0.0.weight'],3,4)

model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer3.1.0.weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer3.1.0.weight'],0,1)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer3.1.0.weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer3.1.0.weight'],1,2)
```

```
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer3.1.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer3.1.0.weight'],2,3)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer3.1.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer3.1.0.weight'],3,4)

model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer3.2.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer3.2.0.weight'],0,1)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer3.2.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer3.2.0.weight'],1,2)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer3.2.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer3.2.0.weight'],2,3)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer3.2.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer3.2.0.weight'],3,4)

model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer4.0.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer4.0.0.weight'],0,1)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer4.0.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer4.0.0.weight'],1,2)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer4.0.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer4.0.0.weight'],2,3)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer4.0.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer4.0.0.weight'],3,4)

model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer4.1.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer4.1.0.weight'],0,1)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer4.1.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer4.1.0.weight'],1,2)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer4.1.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer4.1.0.weight'],2,3)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer4.1.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer4.1.0.weight'],3,4)

model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer4.2.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer4.2.0.weight'],0,1)
```

```
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer4.2.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer4.2.0.weight'],1,2)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer4.2.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer4.2.0.weight'],2,3)
model['state_dict']['pts_middle_encoder.encoder_layers.encoder_layer4.2.0.
    weight'] = torch.transpose(model['state_dict']['pts_middle_encoder.
    encoder_layers.encoder_layer4.2.0.weight'],3,4)

model['state_dict']['pts_middle_encoder.conv_out.0.weight'] = torch.transpose(
    model['state_dict']['pts_middle_encoder.conv_out.0.weight'],0,1)
model['state_dict']['pts_middle_encoder.conv_out.0.weight'] = torch.transpose(
    model['state_dict']['pts_middle_encoder.conv_out.0.weight'],1,2)
model['state_dict']['pts_middle_encoder.conv_out.0.weight'] = torch.transpose(
    model['state_dict']['pts_middle_encoder.conv_out.0.weight'],2,3)
model['state_dict']['pts_middle_encoder.conv_out.0.weight'] = torch.transpose(
    model['state_dict']['pts_middle_encoder.conv_out.0.weight'],3,4)

torch.save(model, "adjusted_fcos3d_normal.pth")
```

Listing 6: Ficheiro de alteração das layers do modelo fcos3d para obtenção de resultados: Convert_model_parameters.py.

7

BIBLIOGRAFIA

Equity research. semiconductor industry primer november 14, semiconductors, 2014. URL http://docplayer.net/29874320-Equity-research-semiconductor-industry-primer-november-14-semiconductors.html.

Andreas and Raquel. Are we ready for autonomous driving? the kitti vision benchmark suite, 2012.

Karsten Behrendt. Boxy vehicle detection in large images. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2019.

Asmund Brekke. Multimodal 3d object detection from simulated pretraining. 2019.

Åsmund Brekke, Fredrik Vatsendvik, and Frank Lindseth. Multimodal 3d object detection from simulated pretraining. *arXiv preprint arXiv:1905.07754*, 2019.

Holger Caesar. nuscenes: A multimodal dataset for autonomous driving. 2020.

Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11618–11628, 2020. doi: 10.1109/CVPR42600.2020.01164.

Sean Campbell, Niall O'Mahony, Lenka Krpalcova, Daniel Riordan, Joseph Walsh, Aidan Murphy, and Conor Ryan. Sensor technology in autonomous vehicles : A review. In *2018 29th Irish Signals and Systems Conference (ISSC)*, pages 1–4, 2018. doi: 10.1109/ISSC.2018.8585340.

Simon Chadwick, Will Maddern, and Paul Newman. Distant vehicle detection using radar and vision. 1 2019.

Sandra Dixe. Damage detection inside a car using computer vision, 2020.

W Elmenreich. Sensor fusion in time-triggered systems, 2002.

Francis Engelmann, Theodora Kontogianni, Alexander Hermans, and Bastian Leibe. Exploring spatial context for 3d semantic segmentation of point clouds. In *IEEE International Conference on Computer Vision, 3DRMS Workshop, ICCV*, 2017.

EUParliamentNews. Self-driving cars in the eu: from science fiction to reality, 2020.

EuropeanCommission. 2019 road safety statistics, 2020.

EuropeanParliamentNews. Safer roads: new eu measures to reduce car accidents, 2020.

Daniel J. Fagnant and Kara M. Kockelman. Dynamic ride-sharing and fleet sizing for a system of shared autonomous vehicles in Austin, Texas. *Transportation*, 45(1):143–158, jan 2018. ISSN 15729435. doi: 10.1007/s11116-016-9729-z.

Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis, 2016.

A. Geiger. The kitti vision benchmark suite, 2012a.

A. Geiger. The kitti vision benchmark suite, 2012b.

Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

Xinyu Huang. The apolloscape open dataset for autonomous driving and its application. 2019.

Apoorva Joglekar, Devika Joshi, Richa Khemani, Smita Nair, and Shashikant Sahare. Depth estimation using monocular camera. 2011.

Matthew Johnson-Roberson, Charles Barto, Rounak Mehta, Sharath Nittur Sridhar, Karl Rosaen, and Ram Vasudevan. Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks? 10 2016.

Raimi Karim. Illustrated: 10 cnn architectures, 2019. URL https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d. Último acesso em 23 de Janeiro de 2020.

Prabhjot Kaur. A survey on simulators for testing self-driving cars. 2021.

Prabhjot Kaur, Samira Taghavi, Zhaofeng Tian, and Weisong Shi. A survey on simulators for testing self-driving cars. 1 2021.

Sergejs KODORS. Point distribution as true quality of lidar point cloud, 2017.

Kumar. Fcos3d train on kitti dataset, 2022.

Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12697–12705, 2019.

Elizabeth Levy Paluck Rebecca Littman. The cycle of violence: Understanding individual participation in collective violence. 2015.

Patrick Langechuan Liu. Orientation estimation in monocular 3d object detection, 2019.

Wei Liu and Dragomir Anguelov. Ssd: Single shot multibox detector. 2016.

Ramin Nabati. Centerfusion: Center-based radar and camera fusion for 3d object detection. 2020.

Ramin Nabati and Hairong Qi. Centerfusion: Center-based radar and camera fusion for 3d object detection. 11 2020. doi: 10.1109/WACV48630.2021.00157.

Felix Nobis, Maximilian Geisslinger, Markus Weber, Johannes Betz, and Markus Lienkamp. A deep learning-based radar and camera sensor fusion architecture for object detection. In *2019 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, pages 1–7, Oct 2019. doi: 10.1109/SDF.2019.8916629.

Niall O' Mahony, Sean Campbell, Lenka Krpalkova, Daniel Riordan, Joseph Walsh, and Aidan Murphy. Computer vision for 3d perception a review. 10 2018.

Abhishek Patil. The h3d dataset for full-surround 3d multi-object detection and tracking in crowded urban scenes. 2019.

Prabhu. Understanding of convolutional neural network (cnn) — deep learning, 3 2018.

Charles R. Qi. Pointnet: Deep learning on point sets for 3d classification and segmentation. 2017.

Dragan Radovanovic and Danielle Muoio. This is what the evolution of self-driving cars looks like, 10 2016.

Sumit Ranjan and S. Senthamilarasu. *Applied deep learning and computer vision for self-driving cars : build autonomous vehicles using deep neural networks and behavior-cloning techniques*. ISBN 9781838646301.

Joseph Redmon. You only look once: Unified, real-time object detection. 2016.

Adrian Rosebrock. Intersection over union (iou) for object detection, 2016. URL https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. Último acesso em 12 de Fevereiro de 2020.

Sumit Saha. A comprehensive guide to convolutional neural networks — the eli5 way, 12 2018.

Babak Shahian Jahromi, Theja Tulabandhula, and Sabri Cetin. Real-time hybrid multi-sensor fusion framework for perception in autonomous vehicles. *Sensors*, 19(20), 2019. ISSN 1424-8220. URL https://www.mdpi.com/1424-8220/19/20/4357.

Gal Shenhav. Convolutional neural network architecture: Forging pathways to the future. URL https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-architecture-forging-pathways-future/. Último acesso em 12 de Junho de 2020.

Vishwanath A Sindagi, Yin Zhou, and Oncel Tuzel. Mvx-net: Multimodal voxelnet for 3d object detection. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7276–7282. IEEE, 2019.

Surya Pratap Singh. Fully connected layer: The brute force layer of a machine learning model, 2017.

VB Staff. Moving autonomous vehicles from rd to mass production is closer than you think, 7 2021.

Ihor Starepravo. How sensor fusion for autonomous cars helps avoid deaths on the road. 12 2021.

Pei Sun. Scalability in perception for autonomous driving: Waymo open dataset. 2020.

Muneeb ul Hassan. Vgg16 – convolutional network for classification and detection, 2018. URL https://neurohive.io/en/popular-networks/vgg16/. Último acesso em 12 de Junho de 2020.

Jani Uusitalo. Using desktop ocr and adaptive preprocessing for recognizing text on extruded surfaces. *Human Friendly Mechatronics*, pages 247–252, 2001. doi: 10.1016/B978-044450649-8/50042-5.

A. Sindagi Vishwanath, Zhou Yin, and Tuzel Oncel. Mvx-net: Multimodal voxelnet for 3d object detection, 2019.

Denver Hill Walling. The design of an autonomous vehicle research platform, 2017.

Tai Wang. Fcos3d: Fully convolutional one-stage monocular 3d object detection. 2021.

Tai Wang, Xinge Zhu, Jiangmiao Pang, and Dahua Lin. FCOS3D: Fully convolutional one-stage monocular 3d object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, 2021a.

Yingjie Wang, Qiuyu Mao, Hanqi Zhu, Yu Zhang, Jianmin Ji, and Yanyong Zhang. Multi-modal 3d object detection in autonomous driving: a survey. 6 2021b.

Zhangjing Wang, Yu Wu, and Qingqing Niu. Multi-sensor fusion in automated driving: A survey. *IEEE Access*, 8:2847–2868, 2020a. doi: 10.1109/ACCESS.2019.2962554.

Zhangjing Wang, Yu Wu, and Qingqing Niu. Multi-sensor fusion in automated driving: A survey. *IEEE Access*, 8:2847–2868, 2020b.

Xinshuo Weng. All-in-one drive: A large-scale comprehensive perception dataset with high-density long-range point clouds. 2020.

Xinshuo Weng, Yunze Man, Dazhi Cheng, Jinhyung Park, Matthew O'Toole, and Kris Kitani. All-In-One Drive: A Large-Scale Comprehensive Perception Dataset with High-Density Long-Range Point Clouds. *arXiv*, 2020.

Benjamin Wilson. Argoverse 2: Next generation datasets for self-driving perception and forecasting. 2023.

Bin Yang, Runsheng Guo, Ming Liang, Sergio Casas, and Raquel Urtasun. Radarnet: Exploiting radar for robust perception of dynamic objects. 7 2020.

De Jong Yeong, Gustavo Velasco-Hernandez, John Barry, and Joseph Walsh. Sensor and sensor fusion technology in autonomous vehicles: A review. *Sensors*, 21(6), 2021a. ISSN 1424-8220. doi: 10.3390/s21062140. URL https://www.mdpi.com/1424-8220/21/6/2140.

De Jong Yeong, Gustavo Velasco-Hernandez, John Barry, and Joseph Walsh. Sensor and sensor fusion technology in autonomous vehicles: A review. *Sensors*, 21:2140, 3 2021b. ISSN 1424-8220. doi: 10.3390/s21062140.

Shivy Yohanandan and Sabina Pokhrel. map (mean average precision) might confuse you!, 6 2020.

Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. 7 2017.

Xingyi Zhou. Object as points. 2019.

Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. 4 2019.

Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. 2017.