UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
UNIDADE ACADÊMICA DO CABO DE SANTO AGOSTINHO
BACHARELADO EM ENGENHARIA ELETRÔNICA

DAVI DE ALMEIDA MELO

Sistema de classificação e compressão para nuvens de pontos geradas por sensores LIDAR

Cabo de Santo Agostinho – PE
2025

DAVI DE ALMEIDA MELO

Sistema de classificação e compressão para nuvens de pontos geradas por sensores LIDAR

Monografia apresentada ao Curso de Graduação em Engenharia Eletrônica da Unidade Acadêmica do Cabo de Santo Agostinho, campus da Universidade Federal Rural de Pernambuco, como requisito para obtenção do grau de Bacharel em Engenharia Eletrônica.

Área de concentração: Inteligência Artificial

Orientador: Professor Dr. Felipe Alberto Barbosa Simão Ferreira

Cabo de Santo Agostinho – PE

2025

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema Integrado de Bibliotecas da UFRPE
Biblioteca da UACSA, Cabo de Santo Agostinho - PE, Brasil
Bibliotecária – Rosimeri Gomes Couto – CRB-4/1395

M528s Melo, Davi de Almeida.
Sistema de classificação e compressão para nuvens de ponto geradas por sensor LIDAR. / Davi de Almeida Melo. – Cabo de Santo Agostinho, 2025.
91 f. : il.

Orientador: Felipe Alberto Barbosa Simão Ferreira.
Trabalho de Conclusão de Curso (Graduação em Engenharia Eletrônica) – Universidade Federal Rural de Pernambuco. Unidade Acadêmica do Cabo de Santo Agostinho, 2025.

Inclui referência.

1. IA. 2. Clusterização. 3. Nuvens de pontos 4. Quantização vetorial. I. Ferreira, Felipe Alberto Barbosa Simão, orient. II. Título.

CDD 621.381

DAVI DE ALMEIDA MELO

Sistema de classificação e compressão para nuvens de pontos geradas por sensores LIDAR

Monografia apresentada ao Curso de Graduação em Engenharia Eletrônica da Universidade Federal Rural de Pernambuco, como requisito parcial para a obtenção do grau de Bacharel em Engenharia Eletrônica.

Aprovada em: 19/12/2025.

BANCA EXAMINADORA

_____

UFRPE Prof. Dr. Felipe Alberto Barbosa Simão Ferreira

_____

UFRPE Prof. Dr. João Marcus Pereira Lima e Silva

_____

UFRPE Prof. Dr. Marcel Ayres de Araújo

Dedico este trabalho aos meus pais e ao meu irmão que sempre me apoiaram em todos os momentos.

AGRADECIMENTOS

Agradeço aos meus pais pelo carinho, motivação e apoio absoluto que tive durante toda minha formação. A minha querida mãe, dona Verônica, pelos conselhos e palavras de conforto em momentos difíceis desse caminho. Ao meu pai, seu Delmiro, pela fé incondicional nas minhas capacidades. Ao meu irmão, Vitor, pela união e companheirismo que me fortaleceram durante a elaboração deste trabalho e ao longo de todo o curso.

Agradeço aos meus familiares, pela compreensão diante da minha ausência e pelo amor que me motiva a persistir e continuar sempre.

Ao meu professor orientador Felipe, por me guiar no âmbito profissional e acadêmico, pela orientação deste trabalho, pela oportunidade de pesquisar e aprender, pelas aulas e principalmente pela amizade.

Aos meus amigos, especialmente Petterson e Ricardo, pela felicidade, alegria e dificuldades compartilhadas ao longo da minha formação.

RESUMO

As mídias imersivas buscam representar a realidade em modelos 3D digitais. Por sua vez, as nuvens de pontos (*Point Clouds – PC*) são parte desse conjunto capaz de digitalizar estruturas, objetos e regiões. Uma *PC* é composta pelo conjunto de pontos que descreve a cena ou objeto e cada ponto possui uma coordenada no plano cartesiano (x,y,z) além de algumas informações adicionais como reflectância, iluminância ou camadas de cores. Essa mídia é utilizada em sistemas da atualidade nas mais diversas áreas. Algumas delas são: direção autônoma, realidade aumentada, robótica, arquitetura, vendas, planejamento, gerenciamento, saúde e educação. Entretanto, nuvens possuem na maior parte dos cenários um alto volume de dados, o que dificulta a sua utilização. Esse volume impacta as etapas de processamento, armazenamento e transmissão das nuvens de pontos. Além disso, outro problema enfrentado ao lidar com nuvens de pontos é o formato não linear. Ele interfere principalmente nas técnicas de aprendizado profundo que buscam extrair informações da nuvem, como por exemplo modelos de segmentação semântica e classificação de objetos. Nesse contexto, o presente estudo tem como objetivo desenvolver e analisar um sistema de compressão e classificação de nuvens de pontos geradas por sensores LIDAR. A quantização vetorial é o método de compressão adotado para o sistema. Já o modelo de classificação de objetos foi customizado a partir de camadas de convolução baseadas no modelo PointNet++ e PPFNet. O algoritmo de quantização vetorial obtido é capaz de alcançar altas taxas de compressão enquanto mantém a qualidade das nuvens de pontos ao serem reconstruídas. Uma arquitetura central e sete diferentes modelos de classificação foram criados. Todos eles possuem acurácia acima de 84,23%. Ademais, os algoritmos de compressão e classificação podem ser executados em conjunto e em qualquer sequência de execução necessária. Por fim, o sistema foi avaliado de modo qualitativo perante diferentes cenários e quantitativamente em função da qualidade de reconstrução e acurácia dos modelos.

Palavras-chave: AI; clusterização – nuvens de pontos; quantização vetorial; classificação - objetos; nuvem de pontos.

ABSTRACT

Immersive media seek to represent reality in digital 3D models. Point clouds (PCs) are part of this set of technologies capable of digitizing structures, objects, and regions. A point cloud consists of a set of points that describes a scene or object, with each point having Cartesian coordinates (x, y, z) along with additional information such as reflectance, illuminance, or color layers. This technology is used in contemporary systems across a wide range of fields, including autonomous driving, augmented reality, robotics, architecture, sales, planning, management, healthcare, and education. However, point clouds typically involve a high volume of data, which increase the difficult to use of this technology. This large volume affects the processing, storage, and transmission of point clouds. Another challenge when working with point clouds is their non-linear structure, which particularly hinders deep learning techniques aimed at extracting information from the cloud—such as semantic segmentation and object classification models. In this context, the present study aims to develop and analyze a compression and classification system for point clouds generated by LIDAR sensors. Vector quantization is adopted as the compression method, while the object-classification model is customized from convolutional layers based on the PointNet++ and PPFNet architectures. The resulting vector-quantization algorithm achieves high compression rates while preserving point-cloud quality upon reconstruction. A central architecture and seven distinct classification models were created, all with accuracy above 84.23%. Moreover, the compression and classification algorithms can run jointly in any desired sequence. Finally, the system was evaluated qualitatively across different scenarios and quantitatively in terms of reconstruction quality and model accuracy.

Keywords: AI; point clouds - clustering; vector quantization; objects - classification; point clouds.

LISTA DE ILUSTRAÇÕES

LISTA DE EQUAÇÕES

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

6-DOF	Six Degrees Of Freedom
CAD	Computer-Aided Design
CBD	Central Business District
CNN	Convolutional Neural Network
DMLP	Dentro da Multilayer Perceptron
DFMLP	Dentro e Fora da Multilayer Perceptron
DL	Deep Learning
FH	Fator de Compressão da codificação de Huffman
FQ	Fator de Compressão de Quantização Vetorial
FQH	Fator de Compressão da Quantização Vetorial Aliado a Huffman
FMLP	Fora da Multilayer Perceptron
GNN	Graph Neural Network
IDE	Integrated Development Environment
K-NN	K Nearest Neighbors
LIDAR	Light Detection and Raging
MLP	Multilayer Perceptron
NN	Neural Network
PC	Point Cloud
PSNR	Peak Signal-to-Noise Ratio
QV	Quantização Vetorial
RGB	Red, Green, Blue
RGBD	Red, Green, Blue, Depth
RELU	Rectified Linear Unit
RMSE	Root Mean Square Error
SSIM	Structural Similarity Index
TH	Taxa de Compressão da codificação de Huffman
TQ	Taxa de Compressão de Quantização Vetorial
TQH	Taxa de Compressão da Quantização Vetorial Aliado a Huffman
VANTS	Veículos Aéreos Não Tripulados

SUMÁRIO

1 INTRODUÇÃO

Nuvem de pontos ou *PC* (*Point Cloud*) é um tipo de mídia capaz de representar a realidade em um modelo *3D* computacional. Ela é composta essencialmente pela informação das coordenadas tridimensionais de um objeto ou região. Além disso, cada ponto pode trazer consigo atributos adicionais como: cor, iluminância e refletância (Gao, 2025; Quach, 2022).

Point Clouds são capazes de descrever tanto elementos isolados como também grandes regiões ou estruturas. Alguns exemplos são: utensílios, móveis, cômodos, carros, pessoas, lavouras, fazendas e florestas. As nuvens de pontos possuem diversas áreas de aplicação como arquitetura e engenharia civil (Wang, 2019; Tommasi, 2016), agricultura (Comba, 2018; Catala-Roman, 2024), saúde e educação (Yang, D., 2024; Weber, 2024), direção autônoma e robótica (Yue, 2018; Introducing, 2025), realidade aumentada ou virtual (Bruder, 2014; Fukuda, 2018).

Portanto, trata-se de uma das principais tecnologias dentre as mídias imersivas. A aquisição ou geração de nuvens de pontos ocorre a partir de certos métodos, alguns deles são: *LIDAR* (*Light Detection and Ranging* ou detecção de luz e distância), sensores de profundidade, câmeras *RGBD* (*Red, Green, Blue, Depth*), modelagem *3D* e reconstrução *3D* com de imagens *2D* (What, 2024; Liu, 2020; Gao, 2023; Cai, 2021). Após sua geração, elas podem conter desde milhares a bilhões de pontos (Xu, 2018; Agapaki, 2020).

Dito isso, o volume de dados é um dos principais desafios enfrentados durante a aplicação e o uso de *Point Clouds*. O tamanho das nuvens implica diretamente nas etapas de processamento, armazenamento, transmissão e classificação. Cada uma dessas etapas torna-se lenta ou até inviável de acordo com o contexto do sistema (Xu, 2018; Gao, 2025). Zhang (2023), especificamente na tarefa de classificação, pontua sobre o problema de consumo excessivo de memória e perda de informação.

Logo, técnicas de compressão como a quantização vetorial (Fonseca, 2018) são essenciais para avanços no uso de nuvem de pontos (Zhou, 2024). Em trabalho recente, Flores (2025) aplica quantização vetorial sobre modelos de *machine learning* (aprendizado de máquina) para aplicações em soluções de AI *low-power* (inteligência artificial aplicada em cenário de baixo consumo). Nesse contexto a compressão é de 90% e não existem perdas expressivas na acurácia do modelo.

Outro desafio ao lidar com *Point Clouds* é o seu formato não linear. Por exemplo, um formato tabular, em grid, ou grade é aquele que pode ser organizado exatamente como uma tabela comum de duas dimensões que possui linhas e colunas. As imagens 2D convencionais

possuem formato tabular regular. Pois, um *pixel* qualquer dentro de uma imagem está a 1 unidade dos *pixels* vizinhos adjacentes. Já as nuvens são formadas por um *grid* (grade) não regular, ou seja, não existe uma distância fixa entre os pontos.

Além disso, não há ordem existente nos pontos de uma nuvem pois, não existe o primeiro segundo ou terceiro ponto. Já em uma tabela, existe a primeira célula da primeira linha da primeira coluna, segunda célula da primeira linha e segunda coluna e assim por diante. Esse formato não estruturado é apontado por Diab (2022) como um dos principais desafios na classificação de nuvem de pontos.

Diab (2022) menciona a impossibilidade de uso da *NN* (*Neural Network* ou Redes Neurais) tradicional que necessita de dados em formato de grade para fazer classificações. Assim como as *NNs* tradicionais, vários algoritmos de aprendizado de máquina clássico precisam receber dados tabulares para funcionar corretamente, alguns deles são: *Random Forest*, *Decision Tree*, *K-NN* e *Logistic Regression.*

Dessa forma, algumas técnicas, adaptações e tratamentos são necessários para classificar nuvem de pontos. Uma possibilidade é transformar nuvens de pontos em grafos e utilizar a *GNN* (*Graph Neural Network).* Por sua vez, o grafo é uma estrutura matemática discreta formalmente definida como um par ordenado G = (V, E) no qual: V é um conjunto finito e não vazio de elementos chamados vértices (ou nós) e E é um conjunto de pares não ordenados de vértices distintos chamados de arestas (ou arcos).

Outras duas abordagens comuns são a utilização de *voxel's* e o uso de imagens em formato de grade. Contudo, essas últimas duas transformações tornam o conjunto de dados ainda mais volumoso. Primeiro, o voxel é criado a partir da definição de um volume 3D fixo que contem toda a nuvem de pontos. Esse volume é subdividido em milhões de pequenos cubos (voxels). Cada voxel armazena a informação de possuir um ponto dentro dele ou não. Essa nova matriz é significativamente maior que a sequência de coordenadas de pontos da nuvem em seu formato convencional (x, y, z, atributos). Isso ocorre porque os espaços vazios agora passam a ser representados por um valor na matriz.

Já a conversão de nuvem pontos em imagens com formato de grade exige a criação de 6 diferentes vistas de um cubo. Novamente a representação do "vazio" acresce o volume da nuvem de pontos. Para resolução deste problema, o *PointNet++* (Qi, 2017) é uma arquitetura hierárquica que opera diretamente sobre conjuntos de pontos (*sets*), usando *Set Abstractions* (amostragem, agrupamento e um "mini-PointNet") para aprender *features* locais. Ademais, essa arquitetura hierárquica do *PointNet++* pode ser interpretada como uma camada *GNN* simplificada.

Pesquisas recentes de revisão (Zhang, 2023; Li, 2020; Sarker, 2024) reafirmam a relevância do *PointNet++* que se destaca ao ser comparado com modelos atuais. Em Qi (2017) alcançou 91.90% de acurácia para o banco de dados *ModelNet40* apresentado no trabalho de Wu (2015) e 85.10% de acurácia para o banco *ShapeNet* que foi desenvolvido no estudo de Chang (2015). Além disso, possui a pontuação mais alta na métrica *F1* (90.30%) dentre os modelos com métodos *Pointwise MLP* (ponto a ponto que utiliza *Multilayer Perceptron*) comparados por Sarker (2024).

Além da problemática de volume e formato das nuvens de pontos, a classificação possui suas dificuldades específicas. Algoritmos de classificação de objetos buscam identificar qual o elemento que a nuvem de pontos representa. Por exemplo, um modelo pode ser capaz de identificar se a *Point Cloud* é uma casa, poste, carro ou pessoa. Dito isso, em um contexto real de aquisição de nuvens, como no *dataset* (banco de dados) *Sydney Urban Objects* (Sydney, 2013), a quantidade de pontos que descrevem os objetos pode variar muito dependendo do ângulo de captura do sensor.

Isso significa que pessoas podem ser representadas com 23 pontos em algumas nuvens enquanto em outras são representadas por 5000 pontos. Essa heterogeneidade que existe entre as possíveis representações de uma mesma classe torna a fase de classificação mais complexa e exige algoritmos mais sofisticados. Não obstante, é justamente nesse contexto que operam os carros autônomos, *VANTs* (Veículos Aéreos Não Tripulados) de baixa altitude que registram hectares e outras aplicações semelhantes (Bracci, 2018; Li, 2020).

Partindo desse contexto, o presente estudo possui o objetivo de desenvolver um sistema de compressão e classificação para nuvens de pontos geradas por sensores do tipo *LIDAR*. O algoritmo *K-Means* será utilizado para definir os centroides de agrupamento que viabilizam a compressão. Quantização vetorial gera redundância na representação das nuvens de pontos. Ou seja, o conjunto de símbolos que representa a nuvem de pontos será reduzido e simplificado. A partir disso, a codificação de Huffman será aplicada para utilizar dessa redundância e amplificar os resultados de compressão.

Já a classificação, será executada a partir da criação de novos modelos utilizando camadas da biblioteca *Pytorch* que são inspiradas no *PointNet++*. Para ambas as etapas serão utilizadas as nuvens de pontos do banco de dados da *SydneyUrbanObjects* (De Deuge, 2013; Sydney, 2013). Esse banco de dados fornece nuvens de pontos obtido por sensores LIDAR em ambiente urbano real e foi proposto para o treinamento de modelos de classificação de objetos. Apesar do contexto de direção autônoma estar presente no banco de treinamento, a arquitetura do modelo é capaz de desempenhar qualquer cenário de classificação.

A compressão incide sobre dois tipos de informação das nuvens: suas coordenadas *3D* (a geometria) e seus atributos de cor. Após a compressão das *Point Clouds*, o impacto da codificação de Huffman utilizada em conjunto com a quantização vetorial será mensurado a partir de métricas como *RMSE (Root Mean Square Error)*, *PSNR (Peak Signal-to-Noise Ratio)* e *SSIM (Structural Similarity Index)*. Adicionalmente, será avaliado o impacto da compressão na acurácia dos modelos de classificação desenvolvidos.

Por sua vez, benefício de técnicas que melhoram o treinamento de modelos de *AI* (*Artificial Intelligence*) como *Data Augmentation* e *Dropout* será discernido durante a etapa de classificação. Também será verificado o efeito da variação dos hiperparâmetros de treinamento na acurácia dos modelos elaborados. Ainda na classificação, a arquitetura dos modelos criados foi composta por vários tipos de convolução como *PointNetConv* e *PPFconv* da biblioteca *PyTorch Geometric* (Deng, 2018; Qi, 2017; Pytorch, 2025).

Essencialmente, o sistema foi elaborado para aplicações que possuam sensores LIDAR. A compressão pode ser utilizada em qualquer contexto envolvendo *PC*. Adicionalmente, os modelos de classificação foram treinados para um cenário não ideal (presença de ruídos, obstrução, falhas). Entretanto, a arquitetura é capaz de gerar modelos de modo independente ao ambiente de aplicação. Portanto, trata-se de um sistema adaptável e flexível que não está limitado ao contexto de treinamento do banco de dados da *SydneyUrbanObjects*.

Por fim, uma breve análise comparativa será conduzida entre o sistema desenvolvido neste trabalho e outros sistemas de estudos semelhantes. As áreas de atuação de alguns deles são: direção autônoma, controle e automação, remoção de ruído e previsão de movimentos (Abbasi, 2023; Lu, 2025; Gao, 2021; Zhou, 2022). Variáveis quantitativas, como similaridade, acurácia dos modelos, e qualitativas, como aplicabilidade e adaptabilidade da arquitetura, serão comparadas entre os trabalhos.

2 OBJETIVOS

Dada a contextualização dos problemas e a motivação do trabalho, esta seção apresenta o objetivo geral e os específicos.

2.1 Objetivo Geral

Desenvolver e avaliar um sistema capaz de comprimir e classificar nuvem de pontos capturadas por sensores do tipo *LIDAR*.

2.2 Objetivos Específicos

- Avaliar a quantização vetorial como técnica de compressão de nuvem de pontos;
- Avaliar a quantização vetorial aliada ao código de Huffman;
- Tratar e transformar uma nuvem de pontos em grafo;
- Aplicar modelos de redes neurais para grafos para classificação de nuvens de pontos;
- Avaliar e comparar a acurácia da classificação das nuvens de pontos comprimidas e não comprimidas;

3 FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta a base teórica que fundamenta o desenvolvimento do sistema de compressão e classificação de nuvens de pontos. Ela se limita a duas seções principais, na primeira trata-se das nuvens de pontos, suas aplicações e os métodos para sua classificação. Já a segunda seção apresenta conceitos importantes de compressão e quantização vetorial.

3.1 Nuvens De Pontos 3D E Aplicações

Diversas áreas do conhecimento utilizam nuvens de pontos, desde Engenharia (civil, robótica, automação) até saúde, educação e gerenciamento. Ela é capaz de descrever cenas ou objetos reais em um modelo tridimensional digital. Geralmente, uma nuvem possui milhares a bilhões de pontos que, quando observados em conjunto, revelam o formato do objeto ou cena capturada.

Um ponto é definido primariamente por seus valores no espaço cartesiano (coordenadas X, Y e Z). Além das coordenadas, cada ponto pode armazenar informações adicionais como cor, reflectância ou iluminância. Há inclusive a possibilidade de que a cor de um ponto mude de acordo com o ângulo de visão que se possui da nuvem de pontos (Sandri, 2018). A adição desses

atributos (cor, reflectância) a cada um dos milhões de pontos expande significativamente o volume de dados da nuvem.

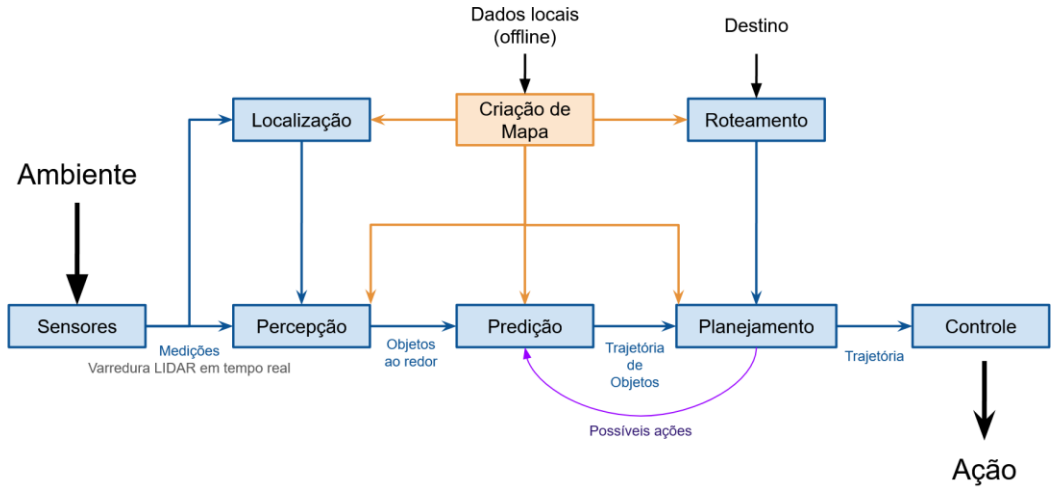
Existem diversos métodos para obter ou criar uma nuvem de pontos. Câmeras *RGB-D* e *scanners LIDAR* são dois métodos de aquisição bem estabelecidos no estado da arte. Também é possível criar nuvens artificiais a partir de modelos *CAD 3D* (*Computer-Aided Design* 3 Dimensions). A amostragem das malhas de um modelo *CAD* permite simular a ação de um *scanner LIDAR* obtendo uma nuvem de pontos sintética com características semelhantes às de uma captura real.

Dito isso, após esse detalhamento sobre nuvem de pontos, a vigente seção busca ilustrar e discutir os principais cenários de aplicabilidade dessas mídias imersivas e consequentemente do sistema desenvolvido no trabalho.

Direção autônoma

O sistema necessário para que veículos autônomos operem com exatidão e segurança é complexo, mas pode ser separado em duas funções essenciais. Primeiro, a geolocalização aliada ao correto mapeamento para definição de rotas confiáveis, ou seja, macro decisões. Em segundo, está a percepção precisa do ambiente ao redor do veículo. A partir desta função tratam-se as micro decisões instantâneas como por exemplo: passagem de faixa, diminuição da velocidade e frenagem (Chen, 2020). O diagrama na figura 1 detalha essa complexidade.

Figura 1 - Lógica de operação de sistemas de direção autônoma



Fonte: adaptado de Chen, 2020.

Nesse contexto da figura 1, as nuvens de pontos impactam na capacidade de percepção. O ambiente é captado e descrito por diversos sensores. Em seguida a informação deve ser utilizada para percepção e localização. Além disso, de forma simultânea o mapeamento funciona integrando todas as etapas intermediárias entre sensoriamento e controle.

Em termos de riscos trata-se de uma função crítica do sistema. Um erro na geolocalização pode levar o passageiro até um local incorreto. Já um erro na percepção do ambiente pode causar colisões e acidentes graves. Para obter exatidão e confiabilidade carros autônomos possuem múltiplos sensores como *scanners LIDAR* e câmeras Digitais (Li, 2020).

De forma geral, *Point Clouds* representam o sentido cognitivo da visão humana para esses sistemas. Apesar do processo de aquisição das nuvens de pontos por sensores *LIDAR* ser custoso, trata-se de uma visualização que independe das condições de iluminação e clima. Diferente das câmeras digitais, *Scanners LIDAR* possuem igual eficiência durante o dia, à noite, quando há brilho, sombras ou intempéries climáticos.

Entretanto, além do alto custo sensores *LIDAR* possuem baixa taxa de atualização 10Hz, geralmente são limitados em baixa resolução. Visando a resolução desses problemas, a fusão de imagens a nuvem de pontos é alvo de diversos trabalhos. Dito isso, esses sistemas são capazes de alcançar análise de profundidade, detecção de objetos em cenas dinâmicas e estacionárias, segmentação semântica e até mesmo a calibração automática de sensores (CUI, 2020).

Após a aquisição de uma cena em formato de nuvem de pontos, existe uma série de tarefas que precisam ser executadas de forma ágil. A sequência de tarefas pode ser estruturada em um *pipeline*: segmentar nuvens de pontos, detectar e localizar objetos na nuvem e classificar os objetos encontrados. Por exemplo, uma cena de trânsito pode ser segmentada em múltiplas nuvens de pontos, em seguida a detecção de objetos encontra um formato distinto na faixa de pedestre a 10 metros do veículo, por fim o modelo classifica e identifica que se trata de uma pessoa naquele ponto.

Nesse contexto, é essencial que cada uma dessas tarefas ocorra de forma precisa e ágil. O que exige um poder computacional adequado e modelos otimizados de detecção, segmentação e classificação. Para além dessas etapas Z. Yang (2024) aborda a criação de uma nova tarefa de pré-treinamento nomeado como *visual point cloud forecasting* ou previsão visual de nuvens de pontos. Ela é capaz de predizer nuvens de pontos futuras baseadas no histórico de capturas.

Logo, fica claro que as nuvens de pontos são essenciais no que diz respeito a área de direção autônoma. Além de participar de uma função crítica, fornece resiliência e

adaptabilidade ao sistema de direção autônoma. Ademais, a classificação de nuvens de pontos é uma tarefa central para tomada de decisão do modelo. Por fim, a compressão está diretamente relacionada com a agilidade de qualquer cenário envolvendo nuvens de pontos. Ela fornece a possibilidade de exportar dados para um servidor externo utilizando menos recursos computacionais e também pode participar de todas as tarefas sempre que o armazenamento for necessário.

Visão robótica

Sistemas robóticos precisam, de alguma forma, extrair informações do ambiente externo para tomar decisões. No geral, sensores podem coletar diversos tipos de dados como temperatura, proximidade, luminosidade e entre outros. *Scanners LIDAR* oferecem uma densa quantidade de informação que pode ser utilizada em múltiplos cenários. O trabalho desenvolvido por Duan (2021) revisa a aplicação de nuvem de pontos no contexto robótico de agarrar e manipular objetos com destreza. Esse estudo aborda mais especificamente os métodos de *Deep Learning* (DL) baseados em nuvens de pontos.

Grasping algorithms ou algoritmos para agarrar utilizavam imagens *RGB* (*red, green, blue*) em conjunto com a *CNN* (*Convolutional Neural Network*). Apesar de alcançar resultados relevantes, nuvem de pontos são capazes de conter mais informações espaciais do que as imagens devido a sua maior dimensão. Duan (2021) destaca que algoritmos de agarrar podem ser divididos em dois tipos, os que utilizam processo de tentativa e erro e aqueles que determinam a melhor postura para agarra baseado nas informações visuais. As nuvens de pontos aplicam-se a ambos, mas possui grande impacto sobre o segundo cenário.

Por sua vez, Wang (2021) explicita o uso de nuvens de pontos no planejamento e otimização da movimentação robótica em um processo industrial de usinagem. A partir do método elaborado por Wang (2021) é possível obter trajetórias suaves, ágeis e estáveis. Analogamente, Zhou (2021) utiliza nuvens de pontos para cálculo da trajetória de um braço robótico com 6 graus de liberdade. O objetivo dessa trajetória é a soldagem a arco. O método proposto é capaz de identificar as bordas entre os objetos e traçar trajetórias suaves para o processo de soldagem.

Chen (2023) também utiliza um braço robótico *6-DOF* (*six degrees of freedom* ou seis graus de liberdade) em conjunto com nuvens de pontos. O modelo *PolarNet* permite o controle guiado por linguagem do braço robótico. A abordagem mais comum na área de controle guiado por linguagem ocorre via imagens *2D*. Entretanto, as limitações e dificuldades para combinar

múltiplos ângulos de visão e inferir posições tornou-se motivação para a escolha de nuvens de pontos nesse artigo.

Por fim, Lin (2023) trata nuvens de pontos para mapear e controlar um robô que opera em ambientes internos. Em todos os trabalhos, nuvens de pontos e algoritmos de machine learning (aprendizado de máquina) são sinônimos de mais informação e exatidão. A área de robótica assim como uma série de processos industriais é diretamente impactada pelas nuvens de pontos e redes neurais.

3.2 Fundamentos da Classificação de Nuvem de Pontos

Nesta seção, os conceitos fundamentais para compreensão do processo de classificação de nuvens de pontos serão abordados. Nela há uma breve revisão histórica da evolução do modelo *PointNet++,* uma introdução a redes neurais, redes convolucionais e redes neurais para grafos.

Classificação de nuvens de pontos

A classificação de nuvem de pontos é uma etapa essencial para vários sistemas. Trata-se da tarefa de identificar e atribuir um nome ou rótulo a determinada nuvem de pontos. Aos possíveis nomes que são resultado final da classificação dá-se a nomenclatura de classe. O número de objetos, nuvem de pontos, de uma mesma classe é nomeado como instância. Por exemplo, um contexto simplista de classificação possui 2 classes como carro e pessoa. Ademais, para cada classe existem 10 instâncias. Isso significa que há 20 nuvens no banco de dados, metade delas são carros e a outra metade são pessoas. Dito isso, além da facilidade intrínseca em distinguir entre a estrutura de um carro ou de uma pessoa, a tarefa torna-se ainda mais simples devido a limitadas possibilidades. Mesmo que o modelo de classificação fosse aleatório em suas escolhas, ainda existe 50% de chance de acerto nesse contexto.

Um cenário real precisa distinguir entre várias classes. O banco *ModelNet40*, por exemplo, possui 40 categorias. Já o *dataset* da *SydeneyUrbanObjects* possui 26 classes. Outros fatores determinantes da dificuldade de classificação são: a qualidade das nuvens de pontos e o número de instâncias para cada categoria. No primeiro caso a baixa qualidade pode tornar algumas classes indistinguíveis. Já no segundo, o desbalanceamento do banco pode favorecer a escolha de classes com mais instâncias.

Outrossim, a classificação de nuvens de pontos especificamente é afetada pelas características dessa mídia. O formato não tabular e seu volume de dados podem tornar a

classificação lenta e inviável. Para sobrepujar estas e outras dificuldades novos modelos, métodos e técnicas são implementados. O trabalho de Zhang (2019) visa agilidade no treinamento utilizando um novo modelo com lógica de aprendizado não supervisionado.

Já os estudos conduzidos por Li (2020) e Zhang (2021) propõe aplicação de *data augmentation* em seus modelos *PointAugment* e *PointCutMix*. Trata-se de executar modificações nas nuvens de pontos para obter novas versões da mesma nuvem e desse modo aumentar o banco de dados. O que geralmente ocorre é o uso das técnicas de *Data Augmentation* ao banco de dados primeiro e em seguida esse novo *dataset* aumentado é entregue ao modelo.

A qualidade de uma nuvem de pontos pode ser baixa por vários motivos: ângulo e distância de aquisição do objeto, corrupção dos dados, qualidade do scanner e ataques externos. Para todos os casos, o modelo de classificação precisa ter ferramentas que contornem essas condições. Os trabalhos de Liu (2021) e Ren (2022) abordam esse tema. O primeiro propõe um novo modelo, *PointGuard*, capaz de classificar corretamente mesmo quando as nuvens são corrompidas. Já o estudo de Ren (2022) avalia a tenacidade de alguns modelos.

Entre eles está o *PointNet++*, o modelo alcança bom desempenho geral quando se avalia a acurácia diante de diferentes bancos de dados. Além disso, no trabalho de Ren (2022) o *PointNet++* também desempenha bem diante dos vários tipos de corrupção de uma nuvem de pontos: transformação de escala, perda de pontos locais e globais, adição de pontos locais e globais, rotação e adição de ruído branco a posição dos pontos.

O histórico do modelo *PointNet++*

Em seu trabalho intitulado "*PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*" Qi (2017) apresenta o modelo *PointNet*. Capaz de classificar e segmentar nuvens de pontos. Esse estudo introduz de forma pioneira a capacidade de treinar um modelo utilizando diretamente as nuvens de pontos. Antes dessa abordagem as aplicações de classificação e segmentação precisavam transformar as nuvens de pontos em *voxels 3D* ou em conjuntos de imagens.

A desvantagem dessas transformações é o alto custo computacional exigido para tratar as nuvens de pontos antes de entregar como entrada para o modelo de classificação. Além de solver este problema, Qi atribui 3 propriedades básicas aos pontos de uma nuvem para elaborar o modelo *PointNet*. São eles: pontos não possuem ordem, pontos possuem relação de distância entre si, pontos são invariantes a transformações.

A primeira propriedade é a não regularidade ou inexistência de ordem do conjunto de pontos. Não existe primeiro, segundo ou último ponto de uma nuvem. Desse modo, o modelo deve ser invariante a permutações. Ou seja, não importa em que sequência o conjunto de pontos será entregue ao modelo, ele precisa ser capaz de retornar igualmente o mesmo resultado. Logo, deve ser invariável a permutações na ordem dos pontos.

A segunda propriedade evidencia a distância como característica de relacionamento entre os pontos. Ou seja, pontos próximos uns dos outros formam sub conjuntos que possuem significado. Portanto, o modelo deve ser capaz de identificar estruturas locais dado a proximidade entre pontos.

Por fim, a terceira propriedade afirma sobre a invariância de um conjunto de pontos a determinados tipos de transformação como rotação e translação. Logo, o modelo deve ser capaz de manter o mesmo resultado de classificação mesmo que uma dessas operações ocorra com a nuvem de pontos.

No entanto, o modelo *PointNet* não é capaz de capturar ou identificar estruturas locais induzidas pelo espaço entre os pontos. Desse modo, existe uma limitação para o reconhecimento de padrões e generalização (QI, 2017). Para resolução dessa característica Qi (2017) propõe um segundo modelo, *PointNet++*, em seu trabalho "*PointNet++: Deep Hierarchical Feature Learning on*".

A estratégia utilizada por ele para capturar a relação de distância entre os pontos foi converter a nuvem de pontos em um grafo. Um elemento matemático definido por vértices e arestas. Nesse contexto, vértices também podem ser chamados de nós. A partir disso foi possível aplicar a *GNN* aliado a todas as capacidades pré-existentes do modelo *PointNet* inicial.

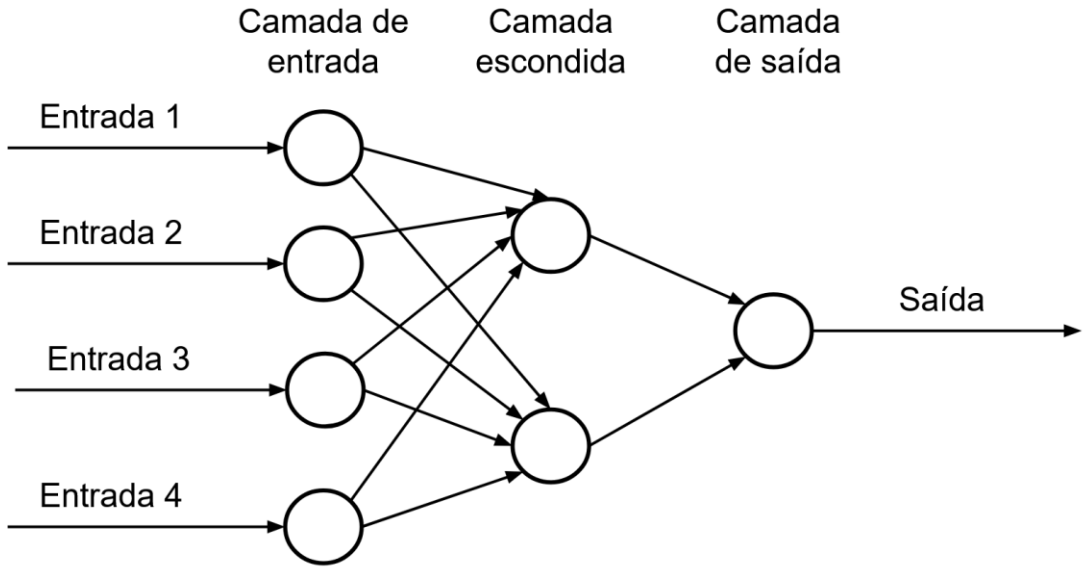
Redes neurais

O modelo *PointNet* é pioneiro no que se refere a introdução de uma arquitetura de treinamento capaz de lidar diretamente com nuvens de pontos. Em seguida, a rede *PointNet++* evolui esse processo transformando nuvens de pontos em grafos antes de treinar o modelo utilizando uma rede neural para grafos *GNN*. Nesses próximos tópicos, o objetivo é introduzir os conceitos básicos de redes neurais *NN* que precedem a rede neural para grafos.

De forma geral, redes neurais imitam o comportamento de neurônios biológicos.
Em suma, os neurônios são células eletricamente ativas. Elas possuem um limiar de ativação. Caso a soma dos sinais de entrada de um neurônio ultrapassem determinado limiar, o neurônio em questão passa a também transmitir um sinal. As redes neurais aplicam essa mesma lógica.

Elas são compostas por camadas de nós (neurônios artificiais), uma camada de entrada (*input layer*), camadas ocultas (*hidden layers*) e uma camada de saída (*output layer*). Cada nó possui sua formulação matemática com contribuições de outros nós e seu próprio limiar de ativação. Assim como o neurônio biológico, caso as entradas de um nó atinjam ou ultrapassem o limiar de ativação, o nó passará a enviar sinais para a próxima camada (O'Shea, 2015; Stryker, 2025a). A figura 2 ilustra como os nós se conectam.

Figura 2 - Estrutura de rede neural básica e suas camadas



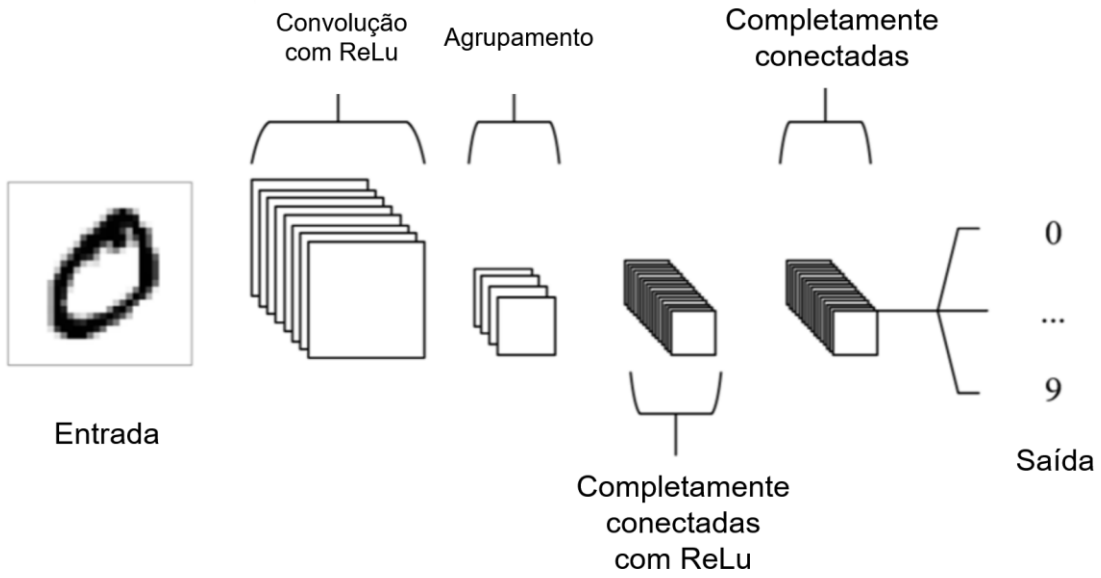
Fonte: adaptado de O'Shea, 2015.

Uma aplicação muito comum de redes neurais é o reconhecimento de imagens. Portanto, um exemplo de entrada para a essa rede são imagens de números escritos manualmente por humanos. Dito isso, a saída do algoritmo deve ser a identificação de qual número está escrito. Em síntese, a *input layer* irá receber as entradas, as *hidden layers* identificam e guardam padrões das imagens e a *output layer* retorna a saída com a identificação do número na imagem.

Redes neurais convolucionais

Redes neurais convolucionais CNN possuem 3 tipos de camadas. As camadas convolucionais (*convolution layers*), camadas de agrupamento (*pooling layers*) e camadas totalmente conectadas (*fully-connected layers*). A figura 3 exemplifica como essas camadas são organizadas.

Figura 3 - Estrutura de uma rede neural convolucional que prediz números manuscritos



Fonte: adaptado de O'Shea, 2015.

A figura 3 organiza as camadas de forma análoga ao exemplo mencionado para redes neurais, ou seja, o reconhecimento de imagens. Dito isso, as camadas convolucionais buscam identificar os padrões contidos nas imagens. Isso ocorre a partir da operação de convolução entre filtros (*kernels*) e a matriz que representa a imagem. Filtros são matrizes de tamanho 3x3 ou 5x5 por exemplo. Cada filtro é capaz de identificar um determinado tipo de padrão na imagem como bordas, curvas e texturas (O'Shea, 2015; Stryker, 2025b).

Para cada convolução entre filtros e imagem o resultado é um *feature map* (mapa de características). Trata-se de uma matriz que indica se existe a característica avaliada naquele filtro e quão intenso é a manifestação dessa característica. A operação de convolução entre imagem e filtros ocorre da seguinte forma: o filtro sobrepõe aos pixels da imagem em determinada região e o produto escalar entre as duas matrizes ocorre. O resultado é armazenado na matriz de característica, o filtro se desloca e o processo recomeça.

Após a convolução, aplica-se uma função de ativação, nesse caso a *ReLU* (*Rectified Linear Unit* ou Unidade Linear Retificada), que possui o objetivo de acrescentar não linearidade ao modelo. Essa função zera valores negativos e mantem valores positivos. A equação 1 representa esse comportamento.

$$ReLU(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \quad (1)$$

Em resumo, aplicar a função *ReLU* permite que a *CNN* resolva problemas de maior complexidade. Caso contrário, a rede neural poderia ser reduzida a uma transformação linear. Em sequência está a camada de agrupamento ou *pooling.* Ela é responsável por simplificar os mapas de características e resumir os resultados da convolução. O principal objetivo é a redução da dimensão dos mapas enquanto evidencia as principais características capturadas pelos filtros na camada de convolução.

Esse efeito é obtido aplicando um filtro que percorre a imagem selecionando os maiores valores de intensidade de cada região da imagem. Desse modo, os valores obtidos irão compor a nova matriz que será entregue a próxima camada. Trata-se da camada completamente conectada ou *fully conected layers*. Ela é responsável por tomar a decisão final na classificação baseado no resultado das matrizes da camada anterior. Primeiro, ocorre a operação de *flatten* ou achatamento em que a matriz da camada anterior é convertida em um vetor unidimensional.

Em seguida, este vetor unidimensional é conectado a uma ou mais camadas completamente conectadas. Isso significa que todos os nós dessas camadas estarão conectados. Esta etapa se assemelha ao que ocorre normalmente nas redes neurais comuns. Por fim, a função *softmax* é responsável pela transformação dos valores finais obtidos das camadas completamente conectadas em percentuais que somados chegam 100%. Se o algoritmo irá identificar a existência ou não de uma flor na imagem, um exemplo da resposta final poder ser “existe” (com 98%) e “não existe” (com 2%).

Redes neurais para grafos

A *NN* e *CNN* precisam de dados tabulares estruturados para realizar seus procedimentos. Esta é a principal diferença entre essas redes e a *GNN*. Grafos, assim como nuvens de pontos, possuem formato não linear. A *GNN* é capaz de utilizar dessa estrutura para diversos propósitos. É possível dividir as aplicações da *GNN* em 3 categorias: *graph-based, node-based, edge-based* (Stryker, 2025c; Awan, 2024; Sanchez, 2021).

Graph-based são as aplicações com capacidade de aprender sobre o grafo em si e a partir disso classificá-los. Exatamente como ocorre na classificação de imagens. *Node-based* são as aplicações em que a predição de interesse é sobre os nós do grafo. Por fim, *edge-based* são os problemas de classificação e predição que buscam compreender as relações entre entidades (nós).

A *GNN* possui arquitetura *graph-in graph-out* que significa receber como entrada um grafo transformá-lo e dar como resposta um grafo na saída. Na arquitetura mais básica de uma

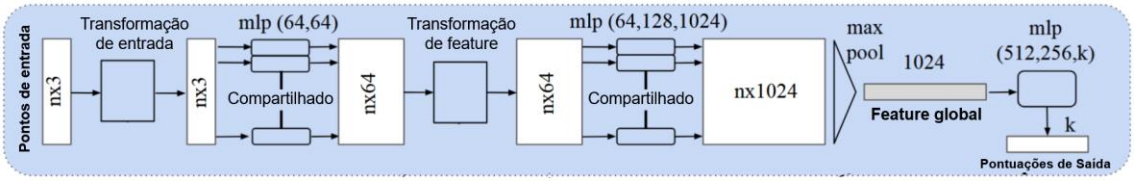
GNN cada elemento do grafo (nós, arestas e contexto global) é processado por um *MLP* (*Multilayer perceptron*) que atua como especialista daquele elemento. Ao fim das *MLP's* o resultado é um grafo com exata mesma estrutura que o grafo inicial, contudo, os valores em seus nós e arestas foram atualizados. Essas mudanças são armazenadas nos vetores de característica (*embeddings*).

Um modelo mais sofisticado pode executar a troca de informação e transformações dentro de uma camada *GNN* aplicando a técnica *message passing*(passagem de mensagem). Primeiro, cada nó reúne informações de seus vizinhos mais próximos, em seguida combina todas as mensagens recebidas em uma etapa de agregação (*pooling*), por fim atualiza sua própria representação baseado nessas informações. O processo da *GNN* com *message passing* é análogo ao que ocorre em camadas de convolução da *CNN* para imagens.

Arquitetura PointNet++

Diversos conceitos das redes neurais convolucionais se estendem a construção da arquitetura *PointNet++*. Iniciando pelo modelo de classificação *PointNet* a figura 4 mostra sua arquitetura completa incluindo filtros, camadas, funções de agregação e entre outros.

Figura 4 - Arquitetura do modelo de classificação de nuvem de pontos *PointNe*t

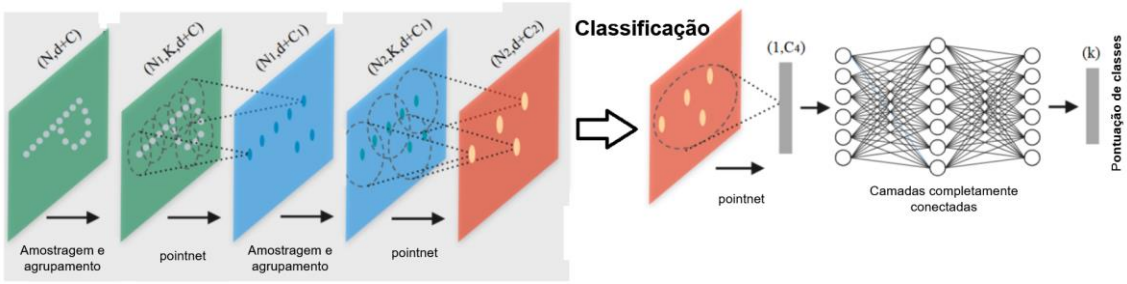


Fonte: adaptado de Qi, 2017.

Os pontos de entrada da nuvem passam por algumas transformações até chegarem à camada *mlp(64,64)* compartilhado no qual 64 é o número de filtros dessa camada. *MLP* ou *Multi Layer Perceptron* é uma rede neural assim como descrito no tópico anterior pela figura 2. Além da camada de entrada e saída, especificamente essa *MLP* possui duas camadas ocultas com 64 nós, também chamados de filtros. No contexto do *PointNet* esses filtros detectam padrões na estrutura da nuvem de pontos. Após mais uma camada de transformação e outra *MLP* os dados são entregues a camada de *pooling* como um vetor *nx1024*.

Por fim, uma última camada *MLP* conduz uma redução de dimensionalidade até chegar ao resultado final. Essa etapa é análoga a camada completamente conectada das redes neurais convolucionais. A figura 5 ilustra a evolução da arquitetura para o modelo *PointNet++*.

Figura 5 - Arquitetura do modelo classificação de nuvem de pontos *PointNet++*



Fonte: adaptado de Qi, 2017.

A primeira etapa é a transformação da nuvem em um grafo. Ela se baseia na decisão de quais pontos estarão conectados. Isso ocorre a partir da busca do vizinho mais próximo. Após encontrar os *K* vizinhos mais próximos de cada ponto, eles são conectados com arestas formando o grafo.

Em seguida, o grafo passa pela fase de agregação executada pela técnica de *message passing* em uma camada *GNN*. Essa fase possui a mesma lógica da camada de *pooling* das redes neurais em que o objetivo é reduzir a dimensão e explicitar as principais características capturadas da imagem. É nessa etapa que o contexto local é capturado pela *GNN* e a *PointNet++* supera o modelo anterior.

Por fim, a segunda fase trata-se da exata mesma lógica encontrada na *PointNet* com redes neurais *MLP* identificando padrões, transformações de matrizes para manutenção da não linearidade, camadas de *pooling* simplificando dimensões e camadas completamente conectadas para as decisões finais.

Arquitetura PPFNet

Desenvolvido por Deng (2018) e baseada na arquitetura *PointNet* o modelo *PPFNet* possui maior tolerância a operação de rotação em nuvens de pontos. A partir da criação de descritores (vetores *4D*) que descrevem a superfície de um par de pontos orientados essa rede é capaz de obter mais robustez e invariância. Ela possui como entrada nuvens de pontos sem tratamento, mas precisa do cálculo dos vetores normais a superfície da nuvem. Por fim, também se destaca ao alcançar mais agilidades nos cálculos de treinamento.

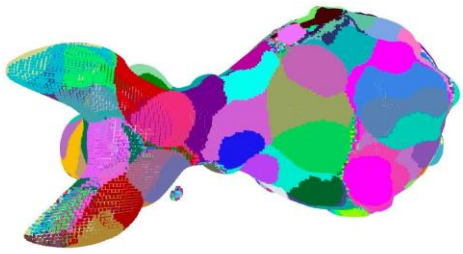
3.3 Conceitos Fundamentais De Compressão Com Quantização Vetorial

Historicamente, a quantização vetorial (*QV*) foi aplicada para compressão ou reconstrução de imagem e áudio. Recentemente, o uso da técnica nas áreas de *machine learning* e *AI* gerou um crescente interesse no tema. Podemos defini-la como uma técnica capaz de representar certo conjunto de dados a partir de um conjunto menor, geralmente, denominado como dicionário ou *codebook*.

Dado que uma imagem é precisamente representada por um *array* de 256 vetores, um exemplo de dicionário seriam 10 vetores capazes de reconstruir a mesma imagem. Entretanto, a imagem reconstruída pode não ser exatamente como antes, dito isso, a *QV* é classificada como técnica de compressão com perdas. A sua principal vantagem é possibilitar taxas de compressão significativamente mais altas do que outras técnicas sem perdas como a codificação por entropia.

Os vetores contidos em um dicionário são chamados de centroides enquanto os agrupamentos que geram os centroides são nomeados como *clusters*. Em suma, os vetores de um dicionário, ou centroides, representam os grupos de vetores, *clusters,* do objeto comprimido. Esse objeto no contexto do sistema elaborado são as nuvens de pontos, mais especificamente as informações de cor e posição das nuvens. A figura 6 mostra o exemplo de uma possível divisão de uma nuvem em grupos(*clusters*).

Figura 6 - Efeito da clusterização de uma nuvem de pontos sinalizado por cores



Fonte: elaborado pelo próprio autor

Esse processo de agrupamento por semelhança é chamado de clusterização e no caso da figura 6 os pontos da nuvem são agrupados utilizando o algoritmo *K-means*. Já a quantização vetorial está presente em diversas aplicações dentro do contexto de transmissão de mídia como introduzido em Nasrabadi (1988). Além disso, a ascensão de tecnologias imersivas com

necessidade de transmissão e armazenamento de grandes blocos de informação torna ainda mais acentuado o uso da quantização vetorial como método de compressão.

A lógica de escolha e construção dos agrupamentos, clusterização, e o algoritmo que executa a busca dos principais representantes de cada agrupamento(dicionário) é discutido em Fonseca (2018) no algoritmo *LBG*. Em síntese, o objetivo é gerar redundância a partir da quantização vetorial, na qual os representantes de cada agrupamento substituem os elementos de seu respectivo agrupamento, reduzindo a variação nos dados (Ferrari, 2004).

Ademais, é possível delimitar 3 elementos essenciais na lógica de compressão da *QV*. O dicionário, os agrupamentos e o vetor índice. O dicionário contém os representantes dos agrupamentos(centroides). Os agrupamentos(*clusters*) são conjuntos de pontos da nuvem agrupados por semelhança, nesse caso, pela regra do vizinho mais próximo. Já o vetor índice é responsável por mapear qual centroide está relacionado a cada *cluster*, consequentemente, a cada ponto da nuvem. O centroide de uma nuvem possui coordenadas (*x,y,z*), o vetor índice possui literalmente os índices da posição dos centroides no dicionário que substituirá o i-ésimo ponto.

Dito isso, supondo que dois dispositivos vão trocar informações nesse contexto. O dispositivo **A** irá enviar nuvens de pontos comprimidas pela *QV*. Enquanto o dispositivo **B** recebe as nuvens de pontos e precisa reconstruí-las. Ambos, precisam possuir o dicionário utilizado para compressão das nuvens que serão enviadas. Uma vez que **A** e **B** possuem o dicionário, a única informação que **B** precisa para reconstruir qualquer nuvem é o vetor índice daquela nuvem.

Dessa forma, em vez de armazenar ou transmitir nuvens de pontos inteiras, pode-se transmitir (ou armazenar) apenas os índices dos vetores do dicionário. Isso significa, geralmente, uma compressão em milhares de vezes. A redundância devido a substituição dos vetores pode então ser explorada por um sistema de codificação como o código de Huffman, por exemplo.

Também é possível utilizar transformadas para se obter coeficientes de frequência, como a transformada discreta do cosseno ou a transformada de Fourier sobre grafos. Devido a suavidade decorrente da redundância introduzida pela *QV*, a transformada apresentará a maior parte de sua energia concentrada nas baixas frequências. Por fim, existe a possibilidade da aplicação de um filtro passa-baixas para reduzir ainda mais a quantidade de componentes com a finalidade de compressão.

Algoritmo *K-Means*

K-Means é um algoritmo de aprendizagem não supervisionado. A sua nomenclatura refere-se à formação de *K* agrupamentos que passarão por uma média (*Means*). O resultado visual obtido pela formação de grupos, clusterização, está ilustrado na figura 6. Entretanto, o principal objetivo é a busca dos representantes de cada grupo, os centróides. Pois, o conjunto dos centróides será o dicionário da quantização vetorial.

Esse algoritmo é comumente subdividido em 4 etapas: inicialização, particionamento, teste de parada e atualização. Em síntese, o seu objetivo é encontrar os *K* vetores que melhor representam determinada distribuição de dados. Consequentemente, ele também busca os melhores agrupamentos dessa distribuição. Isso significa criar grupos que possuam alta similaridade entre seus elementos e baixa similaridade entre os elementos dos demais grupos.

A inicialização refere-se a escolher *K* representantes iniciais. Essa é uma etapa sensível pois, os resultados obtidos pelo *K-means* possuem grande dependência dessa fase. É possível escolher *K* pontos aleatoriamente no espaço, contudo, uma abordagem que simplifica essa etapa é a escolha aleatória de pontos da nuvem. Nesse contexto, os pontos da nuvem também são chamados de vetores de treino enquanto a matriz de pontos é chamada de conjunto de treino.

O particionamento trata-se da busca dos vizinhos mais próximos de cada centro. Consequentemente, essa etapa define quais são os agrupamentos. Ou seja, se um ponto está mais próximo do centro 1 ele estará no grupo 1 e assim por diante. Em seguida, a distância euclidiana quadrática entre cada ponto e centro é calculada. Ao fim dessa etapa a informação do centro mais próximo de cada ponto da nuvem é armazenada e os grupos estão formados.

Assim como *K* define o número de vetores no dicionário o tamanho dos agrupamentos é definido pela variável *tamG*. Por exemplo, *K*=3 e *tamG* = 16 significa que o dicionário terá 3 vetores com 16 intensidades cada. A quantidade de grupos é dada pelo arredondamento da divisão do número de pontos da nuvem pelo tamanho de grupo.

A terceira etapa, teste de parada, delimita quando o algoritmo deve encerrar o processo de busca dos *K* melhores representantes(centros). Nela, verificam-se dois critérios, a similaridade do dicionário em relação aos pontos reais da nuvem e a evolução dessa similaridade entre iterações. A similaridade é calculada a partir da distorção geral que é a soma do módulo das diferenças entre os pontos da nuvem e os centróides. A fórmula da distorção geral é definida na equação 2.

$$DG = \sum_{i=0}^{n} |P_i - C_i| \quad (2)$$

Por fim, cada iteração possui uma distorção e quando a diferença entre as últimas duas distorções for menor que um valor δ pré-estabelecido o algoritmo se encerra.

Caso o teste de parada não seja satisfeito a etapa de atualização se inicia. Nela, novos centros são calculados utilizando da média aritmética dos pontos de cada grupo. A partir dessa etapa em diante, os representantes podem ser propriamente denominados como centróides. Ou seja, a média aritmética dos agrupamentos formados na última iteração. Possuindo novos centróides o algoritmo segue novamente para a etapa de particionamento (Xie, 2020; Kumar, 2020; Fonseca, 2018).

Esse *pipeline* continua em execução até que o critério de parada seja cumprido. Nesse contexto, duas variáveis são essenciais: *K* (o número de centros) e *tamG* (o tamanho dos agrupamentos. O quadro 1 ilustra o pseudocódigo que sintetiza funcionamento do algoritmo.

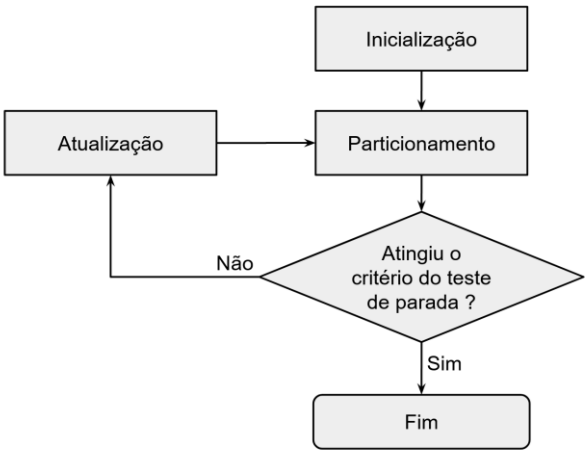
Quadro 1 - Lógica de funcionamento do Algoritmo K-means

Algoritmo *K-means* Pseudocódigo
geração de N centróides (representantes dos agrupamentos) **ENQUANTO** verdadeiro **FAÇA:** para cada vetor de treino, encontrar o representante mais próximo cálculo da distorção geral **SE** condição de parada for satisfeita **ENTÃO:** fim do algoritmo **FIM DO SE** atualização dos representantes **FIM DO ENQUANTO.**

Fonte: adaptado de Fonseca, 2018.

Nele, estruturas essenciais da lógica de programação são substituídas por palavras. Elas expressam a lógica de estruturas condicionais, *loops* e entre outros. Por sua vez, a figura 7 ilustra o diagrama de etapas referentes ao algoritmo *K-Means* e também simplifica a estrutura discutida.

Figura 7 - Fluxograma de etapas do algoritmo *K-Means*



Fonte: elaborado pelo próprio autor

A partir do diagrama é possível analisar o fluxo dos dados e principalmente a lógica de um algoritmo de aprendizado não supervisionado. Ou seja, um algoritmo que não precisa receber dados de treinamento previamente rotulados e que aprende enquanto executa a tarefa.

Codificação de Huffman e quantização vetorial

A codificação de Huffman utiliza da probabilidade de ocorrência dos símbolos na informação a ser codificada para comprimir os dados. Isso ocorre a partir da definição de códigos únicos de tamanho variável. Caracteres que aparecem com frequência possuem menor tamanho em bits que aqueles que não aparecem. De forma simplificada, dado que um símbolo $\boldsymbol{X}$ possui a maior probabilidade de ocorrência, a sua representação em formato binário será a menor do conjunto codificado.

Logo, redundância é o fator essencial para o processo de compactação de Huffman. Dito isso, a quantização vetorial possui como uma de suas características a criação de redundância. Por exemplo, uma nuvem é representada por 1000 pontos. Cada ponto possui 3 intensidades de cor totalizando 1000 vetores de cor (R,G,B). Ao fim da quantização, um dicionário com 10 vetores é capaz de reconstruir as cores da nuvem de pontos. Logo as especificidades dos 1000 pontos iniciais se perdem e o que existe no fim é uma reconstrução com as médias das cores iniciais.

Para detalhar esse fato o quadro 2 mostra a compressão da informação de cor em uma nuvem de pontos.

Quadro 2 - Efeito da quantização vetorial em uma nuvem de pontos

Original	**Comprimida**
	

Fonte: elaborado pelo próprio autor

A redundância gerada pela quantização pode ser explicitada observando os tons da camisa do homem representado. O tom azul acinzentado se torna mais frequente após a compressão. Logo, a codificação de Huffman pode ser utilizada juntamente a quantização vetorial para obter taxas de compressão ainda maiores. Além disso, a codificação de huffman não possui perdas, portanto a similaridade obtida pela quantização vetorial é mantida.

O resultado da quantização vetorial, além do dicionário, é o vetor índice. Logo, a técnica de Huffman será aplicada sobre o vetor índice visto que essa é a informação que será constantemente enviada entre sistemas. De forma prática, dado um nível de quantização $N = 4$, isso significa que teremos os grupos identificados com os índices 0, 1, 2, 3 e o vetor-índice será constituído por uma sequência contendo apenas esses quatro números em formato binário. Portanto, dado que a cor azul acinzentada seja representada pelo índice 2. Então, assim como na imagem ele irá aparecer com mais frequência que antes.

4 MATERIAIS E MÉTODOS

Nesta seção, cada uma das abordagens práticas durante o desenvolvimento e teste dos algoritmos serão detalhadas. O banco de dados, a construção dos códigos e as ferramentas utilizadas são alguns dos temas tratados. Todos os códigos foram desenvolvidos a partir da linguagem de programação *Python* a partir da *IDE* (Integrated development environment) do VSCode ou do *Google Colab*. Além disso, a principal biblioteca utilizada no desenvolvimento

do modelo de classificação foi a *Pytorch Geometric*. Por fim, o dispositivo notebook utilizado para desenvolvimento do código de quantização vetorial possui as seguintes configurações:

- Processador Intel(R) Core(TM) i3-8130U CPU @ 2.20GHz 2.21 GHz
- SSD 256 GB
- RAM instalada de 4,00 GB
- Sistema operacional 64 bits
- Windows 11

4.1 Banco De Dados

Foram utilizados 3 *datasets* de nuvens de pontos. O banco de dados público da *Sketchfab*, o banco disponibilizado pela Universidade de São Paulo (2017) do site da *EmergIMG* e o banco da *SydneyUrbanObjects.* Os dois primeiros foram utilizados para elaborar o código de quantização vetorial a partir de nuvens de pontos com maior densidade. Já o banco de dados da *SydneyUrbanObjects* permitiu validar tanto a quantização vetorial como o modelo de classificação.

EmergIMG USP (UNIVERSIDADE DE SÃO PAULO)

A Universidade de São Paulo possui o banco de dados armazenado na página do projeto de pesquisa *EmergIMG* que se dedica a área de compressão e qualidade de imagens buscando estabelecer um *framework* padrão. O quadro 3 resume as opções de nuvens de pontos na página de *downloads.*

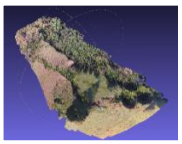
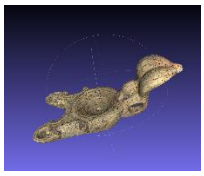
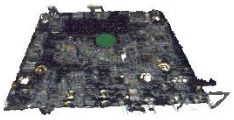
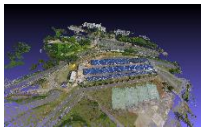
Quadro 3 - Lista de nuvens de pontos disponíveis no banco de dados da USP EmergIMG e suas características

(continua)

Nome	Tamanho	Número de Pontos	Visualização
Bumbameuboi	5 MB	150.388	
CITIUSP	88 MB	5.929.878	

Quadro 3 - Lista de nuvens de pontos disponíveis no banco de dados da USP EmergIMG e suas características

(conclusão)

Nome	Tamanho	Número de Pontos	Visualização
DouradoSite	392 MB	26.137.113	
ErasmusMill	66 MB	66.191.900	
Ipanema	130 MB	8.128.921	
ItanguaCanyon	81 MB	5.456.996	
RamosDeAzevedo	962 MB	64.153.694	
RomanOilLight	42 MB	1.286.052	
Labrador	2 MB	55.158	
VillaLobosPark	109 MB	7.288.489	

Fonte: adaptado de USP, 2017.

São 10 nuvens que variam de milhares de pontos a milhões. Levando em consideração poder computacional e a estrutura da nuvem, a nuvem de pontos Bumbameuboi foi a única

utilizada deste banco de dados. Com 150 mil pontos e seu formato simples, essa nuvem é ideal para testes iniciais.

Sketchfab

O site *Sketchfab* (About, 2025) é um repositório público em que artistas *3D* podem compartilhar seus trabalhos. A aplicação se dedica a modelos *3D* de forma geral incluindo as nuvens de pontos. A figura 8 ilustra a nuvem de pontos *Guy* baixada dessa aplicação.

Figura 8 - Nuvem de pontos *Guy* do repositório público *Sketchfab*

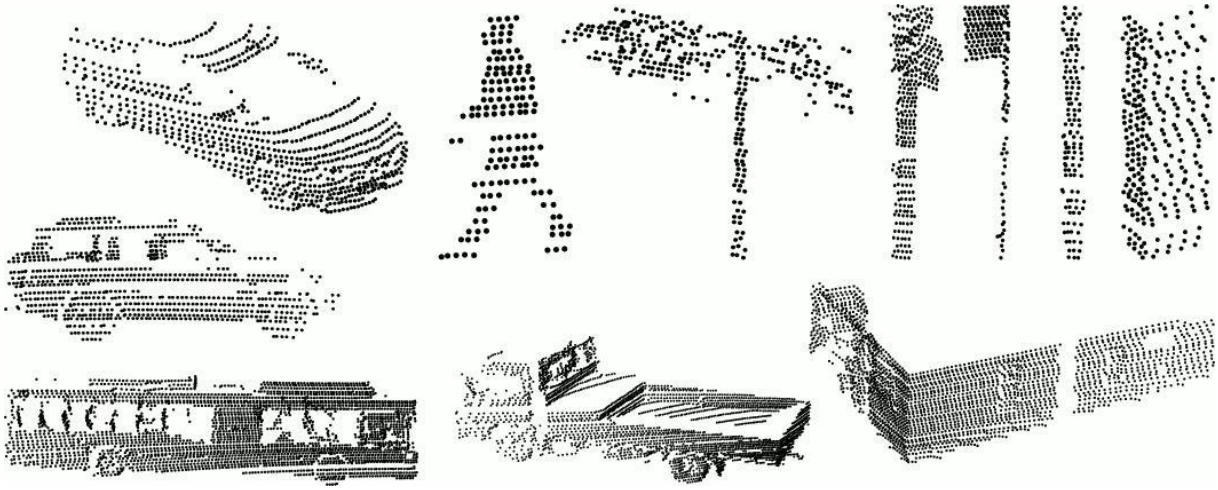
Fonte: About, 2022.

Assim como a nuvem de pontos Bumbameuboi, a nuvem *Guy* possui simplicidade e poucos pontos (66.860), o que é ideal para testes iniciais.

Sydney urban objects

O banco de dados da *SydneyUrbanObjects* possui nuvens de pontos adquiridas em contexto urbano pelo sensor *Velodyne HDL-64E LIDAR*. Elas foram coletadas em *Sydney CBD (central business district)*, na Austrália. Tratam-se de 631 nuvens de pontos com 26 diferentes classes incluindo arvores, veículos, pedestres e estruturas. Esse *dataset* foi elaborado com objetivo de testar algoritmos de classificação e correspondência. A figura 9 ilustra os diferentes elementos desse banco de dados.

Figura 9 - Nuvens de pontos do banco de dados da *Sydney Urban Objects*



Fonte: De Deuge, 2025.

Portanto, as nuvens foram coletadas em cenários não ideais. Isso significa capturar nuvens enquanto ocorre uma obstrução ou enquanto o ângulo e posicionamento do *scanner* prejudicam a identificação do objeto. Em síntese, esse tipo de nuvem de pontos é o cenário real de operação de veículos com direção autônoma. O quadro 4 descreve as informações(campos) que cada uma das nuvens de pontos possui.

Quadro 4 - Informações contidas nas nuvens de pontos do banco de dados da SydneyUrbanObjects

Campo	**Descrição**	**Formato**
t	Carimbo de data e hora	*Int64*
intensity	Intensidade de retorno do ponto ao laser (0 – 255)	*Uint8*
id	*Laser id* (existem 64 *lasers* no *Velodyne*)	*Uint8*
x,y,z	Pontos	*Float32 x3*
azimuth	Ângulo de captura azimutal horizontal	*Float32*
range	Distância do sensor ao ponto coletado pelo laser em metros	*Float32*
pid	Indice de identificação, scan 360°	*Int32*

Fonte: adaptado de Sydney, 2013.

Por sua vez, o quadro 5 lista todas as classes e a quantidade de nuvens para cada classe no banco.

Quadro 5 - Classes e instâncias do banco de dados da SydneyUrbanObjects

Classes	**Instâncias**
4wd	21
bench	2
bicycle	8
biker	4
building	20
bus	16
car	88
cyclist	3
excavator	1
pedestrian	152
pillar	20
pole	21
post	8
scooter	1
ticket_machine	3
traffic_lights	47
traffic_sign	51
trailer	1
trash	5
tree	34
truck	12
trunk	55
umbrella	5
ute	16
van	35
vegetation	2

Fonte: adaptado de Sydney, 2013.

No quadro 5 observa-se que cada classe possui diferentes números de instâncias. O algoritmo de classificação precisa ser capaz de ultrapassar esse cenário não linear tanto da estrutura do banco como também no que se refere a captura das nuvens.

4.2 Quantização Vetorial

O algoritmo completo de quantização possui muitas partes. Além das 4 etapas do algoritmo *K-Means,* outras capacidades também são necessárias ao longo do código. O tratamento das nuvens, plotar nuvens específicas para verificar resultados e testes de modificação de vetores são alguns exemplos de sub-etapas. Esta seção apresenta a progressão dos códigos, a metodologia de desenvolvimento, problemas enfrentados e métricas de avaliação.

Métricas avaliativas para quantização vetorial

A quantização vetorial é um processo com perdas que prioriza a capacidade de compressão. Dito isso, para avaliar a semelhança entre as nuvens comprimidas e as nuvens originais algumas métricas são introduzidas. São elas: *PSNR, SSIM* e *RMSE.*

Elas serão utilizadas após a compressão e reconstrução das nuvens utilizando os vetores do dicionário produzido pelo algoritmo *K-means*. Existem duas informações que são alvos da quantização vetorial nesse estudo. A informação das cores e a localização dos pontos.

O *PSNR* é definido pela equação 3,

$$PSNR = 20log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right), \tag{3}$$

em que MAX_I é o valor máximo do tipo de informação quantizada. Ou seja, quando a compressão ocorre sobre a informação de cor, o valor máximo é 255 na escala *RGB*. Devido a tratamentos necessários no código a informação de cor nessa etapa do algoritmo tornou-se um percentual, ou seja, seu valor máximo é 1.

No caso da informação de posição dos pontos MAX_I é a maior distância existente entre os pontos da nuvem. Por fim, MSE é o erro médio quadrático entre cada cor da nuvem quantizada e a respectivo cor original. De forma análoga também ocorre para a informação de posição dos pontos.

Já o *SSIM* está explicitado na equação 4,

$$SSIM(x,y) = \frac{(2u_x u_y + c_1)(2\sigma_{xy} + c_2)}{(u_x^2 + u_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}, \quad (4)$$

em que u_x é a média das coordenadas dos pontos da nuvem original (nuvem x), u_y é a média das coordenadas dos pontos da nuvem comprimida (nuvem y). Em seguida, σ_x e σ_y são as variâncias da nuvem original e da nuvem comprimida respectivamente. Já σ_{xy} é a covariância entre a nuvem original e a comprimida. Por fim, c_1 e c_2 são constantes pré-estabelecidas. Analogamente ocorre para as informações de cor de cada ponto da nuvem.

O *RMSE*, raiz quadrática média dos erros, é definido pela equação 5,

$$RMSE = \sqrt{\sum_{i=1}^{n} (y'_i - y_i)^2 / n} \quad (5)$$

na qual y'_i é um ponto do conjunto de dados original, y_i é o um ponto do conjunto de dados após a quantização e *n* é o número de pontos. De forma análoga se calcula o *RMSE* para quantização de cores.

Taxas e fatores de compressão

Com objetivo de mensurar a compressão algumas taxas e fatores foram definidas. As taxas são valores percentuais que identificam quanto do volume de dados inicial foi comprimido. Por exemplo, se a taxa de compressão for de 50%, significa que o tamanho em bits da nuvem de pontos foi reduzido pela metade. Uma taxa genérica é exemplificada na equação 6.

$$Tx = 1 - \frac{bits\ após\ compressão}{bits\ antes\ da\ compressão} \quad (6)$$

Ou seja, a taxa avalia a redução percentual do tamanho da nuvem. Já os fatores de compressão avaliam em quantas vezes o tamanho inicial da nuvem foi reduzido. A equação 7 ilustra a relação.

$$Fx = \frac{bits\ antes\ da\ compressão}{bits\ após\ compressão} \quad (7)$$

Logo, podemos sintetizar a relação entre taxas e fatores a partir da equação 6 e 7 que entrega a equação 8.

$$Tx = 1 - \frac{1}{Fx} \quad (8)$$

Taxas compressão são essenciais para a fácil leitura do comportamento de técnicas de compressão como a quantização vetorial. Entretanto, conforme os fatores de compressão se tornam muito grandes a magnitude da compressão é perdida ao longo das casas decimais do percentual. Por exemplo, dado duas taxas de compressão 99.53% e 99.93%. Apesar da proximidade entre os valores elas guardam fatores de compressão muito distintos. A primeira taxa de 99.53% exige um fator de compressão de 212.77 enquanto a segunda possui fator de compressão 1428.57.

Ademais, com a definição do dicionário, que é muito menor que o conjunto das cores originais, será necessário apenas a relação dos índices que indicam a qual ponto cada vetor do dicionário está relacionado. Ou seja, precisamos apenas do vetor índice. O mesmo, em condições reais de troca de dados utilizando da técnica de quantização vetorial seria o único dado a ser enviado para transportar toda a informação de cores da nuvem de pontos. Já o dicionário, precisa ser enviado uma única vez, pois, deve ser armazenado com intuito de reconstruir a informação de cor ao receber o vetor-índice.

Há 3 taxas de compressão calculadas: TQ (Taxa da Quantização), TH (Taxa de Huffman) e TQH (Taxa da Quantização e Huffman). Analogamente, também há 3 fatores de compressão: FQ (Fator da Quantização), FH (Fator de Huffman), FQH (Fator da Quantização e Huffman).

Versionamento

O versionamento de bibliotecas *Python* foi um detalhe importante para ambas as etapas (compressão e classificação). No caso da quantização vetorial utiliza-se a versão 3.11.2 do *Python.* As bibliotecas *Open3D, pandas, numpy, matplotlib, scikit-learn* e *scikit-image* só puderam ser utilizadas em conjunto nessa versão do *Python*. A maior limitação ocorreu devido

a biblioteca *Open3D* em que algumas funções foram descontinuadas. A lista de requerimentos estará no Github (seção 4.4).

Clusterização

Após a correta configuração do ambiente de execução, o primeiro código desenvolvido foi o de clusterização. Um vetor marcador é um vetor de tamanho igual ao número de pontos da nuvem que marca a participação ou não de um ponto dentro de um agrupamento. O vetor marcador pode ser compreendido como uma lista de tarefas em que a tarefa é o agrupamento daquele ponto. Iremos verificar a lista de tarefas sempre que necessário para saber se aquela tarefa já foi feita ou não.

Isso ajuda no sentido de não repetir uma tarefa, que nesse caso significaria agrupar um mesmo vetor em grupos diferentes. Além disso, ***i*** é uma variável de controle que pertence aos inteiros e auxilia na leitura de todos os pontos da nuvem. Para esse pseudocódigo definiu-se o tamanho de agrupamento como 16 (*TamG* = 16). Ou seja, a nuvem de pontos será dividida em grupos de 16 pontos. O quadro 6 ilustra o pseudocódigo discutido.

Quadro 6 - Lógica do funcionamento do algoritmo K-means para PC

Algoritmo *K-means* para nuvens de pontos 3D (pseudocódigo)
Inicialização do vetor marcador **ENQUANTO** todos os pontos não forem agrupados **FAÇA:** Cálculo dos 16 pontos mais próximos do centro[i] Agrupamento dos pontos mais próximos do centro[i] **SE** marcador[i] já possui grupo **ENTÃO:** Nada é feito **FIM DO SE** **SE** marcador[i] não possui grupo **ENTÃO:** Ponto[i] faz parte do grupo do centro[i] **FIM DO SE** **FIM DO ENQUANTO.**

Fonte: elaborado pelo próprio autor

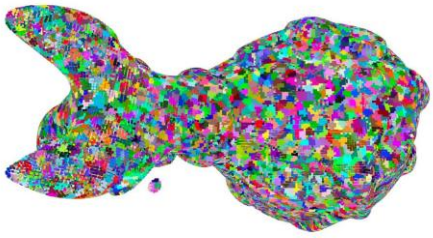
A etapa de particionamento incluindo cálculo das distâncias e verificação do vetor marcador é aquela que demanda maior poder computacional. Primeiramente, foi feito a

inicialização dos centros como pontos aleatórios da nuvem. A nuvem de pontos Bumbameuboi possui 150.388 pontos. Como a nuvem será dividida em grupos de 16 isso significa que irá existir 9399 centros. Para cada centro foram calculadas 150.388 distâncias (todos os pontos da nuvem), para descobrir os 16 pontos mais próximos de cada centro.

Essa abordagem resulta em 1 bilhão de cálculos de distâncias. Entretanto, caso essa etapa fosse executada via força bruta seriam 150 mil cálculos de distâncias para cada ponto da nuvem. Isso equivale a mais de 22 bilhões de cálculos. Logo, existe uma significativa redução no tempo de cálculo utilizando da primeira abordagem.

A adição dos vizinhos mais próximos a cada centro ocorre em um centro por vez. Observe abaixo na figura 10 os agrupamentos gerados por este método, em que cada agrupamento foi pintado com uma cor diferente e as condições são exatamente as descritas acima com relação a número de pontos e tamanho de grupo.

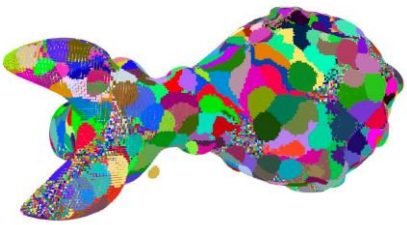
Figura 10 - Processo de clusterização utilizando tamanho de grupo igual a 16



Fonte: elaborado pelo próprio autor

Conforme o valor do tamanho de grupo aumenta os agrupamentos ficam mais visíveis como na figura 11.

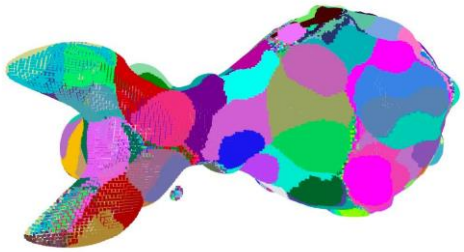
Figura 11 - Processo de clusterização utilizando tamanho de grupo igual a 256



Fonte: elaborado pelo próprio autor

Por fim, ao modificar o tamanho de grupo para 1024 os clusters e seus formatos ficam visualmente maiores. A modificação do tamanho de grupo implica na compressão de um conjunto maior de vetores. O resultado está ilustrado na figura 12.

Figura 12 - Processo de clusterização utilizando tamanho de grupo igual a 1024



Fonte: elaborado pelo próprio autor

Quando o objetivo é a compressão da posição dos pontos, após a clusterização os centróides são calculados normalmente conforme descrito no algoritmo *K-Means*. Já quando o objetivo é a compressão das cores existe uma fase intermediária. Os agrupamentos que estão em função da posição tridimensional foram convertidos em agrupamentos de cores. Essa tratativa ocorre a partir da suposição de que cores semelhantes estarão em regiões próximas.

Metodologia de desenvolvimento de código

A estratégia central é a construção de blocos de código por tarefa ou etapa. Primeiro lidamos apenas com duas nuvens de pontos. A Bumbameuboi e a *Guy*. O primeiro código desenvolvido foi a etapa de agrupamento baseado na regra do vizinho mais próximo. Esse código tem como saída os agrupamentos das 2 nuvens de pontos mencionadas, ambas com tamanho de grupo fixo em 16. Os arquivos de extensão *npy* salvando os agrupamentos feitos nessa etapa foram então utilizados para o processo de compressão das cores. Primeiro é feito a correlação entre agrupamentos de pontos para agrupamento das intensidades de cores de cada camada.

Em seguida o código segue calculando os centróides até que o teste de parada seja cumprido. Obtido o principal resultado que é o dicionário aliado ao vetor índice seria então necessário reconstruir a nuvem de pontos a partir desses dois elementos. Após a reconstrução, o *RMSE* deveria ser calculado e printado. Todas as funcionalidades básicas foram corretamente

implementadas como: a visualização, compressão e reconstrução das nuvens de pontos, cálculo *RMSE* e codificação de Huffman.

Essa estrutura funcionava isoladamente. Para estender os testes e validar o método de compressão era preciso organizar estruturas de *loop* condicionais para automatizar a variação de parâmetros e acesso de mais de uma nuvem de pontos por vez. Assim foi feito inicialmente para cores com nuvens de pontos na extensão *ply*. Em seguida esse processo foi estendido para nuvens de pontos no formato *bin*. Por fim, de modo análogo, a compressão foi reformulada para operar na informação de coordenadas (*x,y,z*) dos pontos.

4.3 Classificação De Nuvens De Pontos

Para desenvolver o modelo baseado no *PointNet++* a linguagem *Python* foi utilizada. A biblioteca *PyG* (*PyTorch Geometric*) reuni uma série de ferramentas para treinar *GNN's*. Já a biblioteca *Open3D* foi utilizada especificamente para processar nuvens com extensão *ply* (Zhou,2018; PyG, 2025). Além delas as bibliotecas *pandas, numpy, matplotlib* e *scikit-learn* foram indispensáveis.

Todo o código de treinamento foi elaborado e executado na plataforma *Google Colab* que permite executar scripts *Python* diretamente na nuvem. Em síntese, os códigos estarão em execução nas máquinas da *Google*. Além de usar a capacidade computacional da nuvem a plataforma *Google Colab* é um modo eficiente para compartilhar códigos. Armazenando os códigos dessa forma enviar o código assemelha-se ao compartilhamento do *link* de uma página web.

Versionamento

Primeiro problema enfrentado no desenvolvimento do algoritmo foi o conflito de versões de diferentes bibliotecas. Apesar da *PyG* estar em constante atualização, outras bibliotecas auxiliares para esse contexto não eram compatíveis. O maior conflito ocorre entre as duas principais bibliotecas *PyG* e *Open3D*. O *Google Colab* por sua vez possui versionamento automático e nesse contexto o problema foi automaticamente resolvido.

Transformação de nuvens de pontos em grafos

Para que uma nuvem de pontos seja transformada em grafo é necessário elaborar algum critério para conectar os pontos(nós). Isso foi feito a partir da regra do vizinho mais próximo. Ou seja, os *N* vizinhos mais próximos de um ponto estarão conectados a ele. Analogamente é

feito para todos os pontos da nuvem. A partir de tentativa e erro o valor de *N* foi fixado em 5 avaliando a acurácia final dos treinamentos. Ou seja, todos os pontos de cada nuvem terão conexão com os 5 pontos mais próximos.

Em seguida os grafos são armazenados num *array* chamado de *dataset* que nesse caso é o banco de dados geral com todas as nuvens. Cada nuvem é estruturada no formato da classe Data da *PyG*. Essa classe descreve os grafos com 4 informações: nós, arestas, índice da classe, nome da classe. Por exemplo, se as classes são carro, pessoa e casa os índices serão 0, 1 e 2. Cada grafo precisa mais adiante entregar ao algoritmo de classificação qual a correta predição sobre sua classe.

Divisão do banco de dados

O banco possui 631 nuvens de pontos e foi dividido na proporção 70 para 30. Ou seja, 70% das nuvens irá para o banco de treino *train_dataset* enquanto 30% delas irá para o banco de teste *test_dataset.* De modo prático, essa divisão ocorreu de duas formas, com ou sem embaralhar o *dataset*. Como o banco possui uma variação muito grande na quantidade de instâncias por classe, embaralhar o banco antes de dividi-lo é essencial. Caso contrário o que ocorre é uma distribuição tendenciosa do banco priorizando as primeiras classes existentes no *dataset*.

Métricas e variáveis de treinamento

Ao longo do treinamento de modelos alguns parâmetros são calculados para compreender o comportamento e o aprendizado deles. O quadro 7 traz as principais métricas e seus significados.

Quadro 7 – Métricas utilizadas para avaliar o desempenho do treinamento do modelo de classificação de nuvens de pontos

Métrica	**Significado**
Accuracy	Percentual de acertos que o algoritmo possui ao classificar o conjunto de treino ou teste.
Recall	Percentual de acerto do algoritmo considerando predições feitas especificamente sobre uma das classes.
F1-score	Média harmônica entre acurácia e recall.
Loss	Valor numérica que indica quanto o algoritmo não aprendeu.
Epoch	Número de ciclos em que o modelo se atualizou e tentou aprender.

Fonte: elaborado pelo próprio autor

A *accuracy* (acurácia) é constantemente utilizada para comparação entre modelos. O objetivo central durante o treinamento é buscar a acurácia mais próxima possível de 100%. Para compreender o *Recall* o seguinte exemplo é destacado. Se um algoritmo irá classificar entre as classes: pessoa, carro, casa e moto. A *Recall* da classe carro refere-se ao percentual de acerto considerando apenas as predições feitas quando as nuvens de pontos eram realmente carros.

Em síntese, considerando apenas as nuvens com classe “carro” quantas dessas nuvens o algoritmo classificou como carro, esse é o *recall* da classe carro. Por fim, a *Loss* é um valor importante verificado ao longo de todo treinamento. Há várias funções capazes de calcular a *loss* do algoritmo como o *MSE(Mean Squared Error)* e *CEL(Cross Entropy Loss)*. A fórmula de cada uma das métricas está organizada no quadro 8.

Quadro 8 - Fórmula das principais métricas de treinamento do modelo de classificação de nuvens de pontos

Métrica	**Fórmula**	
Acurácia	$A = \frac{vn + vp}{fp + fn + vp + vn}$	(9)
Recall	$r = \frac{vp}{vp + fn}$	(10)
F1-Score	$f1 = \frac{2 * p * r}{p + r}$	(11)
Cross Entropy Loss	$L = -\sum P(x) * \log(Q(x))$	(12)

Fonte: elaborado pelo próprio autor

Legendas:

- vn – verdadeiro negativo
- vp – verdadeiro positivo
- fn – falso negativo
- fp – falso positivo
- $P(x)$ – probabilidade real da classe x
- $P(x)$ – probabilidade predita da classe x

Definição da arquitetura e configuração de treinamento

A principal etapa da classificação é a construção da arquitetura do modelo. Nessa etapa o objetivo é encontrar a melhor combinação entre número de camadas, tipos de camadas e quantidade de filtros utilizados. Além disso os parâmetros de treinamento precisam ser ajustados para obter maior exatidão possível. O quadro 9 exibe uma explicação simples sobre cada parâmetro das configurações de treinamento.

Quadro 9 - Hiperparâmetros do treinamento do modelo de classificação de nuvens de pontos

Nome do Parâmetro	**Significado**	**Tipo**	**Valor/Opção Final**
Batch_size	Número de predições realizadas antes de atualizar os pesos da rede neural.	Valor numérico	8
Learning_rate	Magnitude do passo que o modelo dá a cada iteração tentando ajustar os pesos do modelo.	Valor numérico	0,0005
Class_weights	Valores que torna o aprendizado das classes com menos instâncias mais importante.	Técnica	Não utilizado
N	Número de vizinhos mais próximos aos quais cada nó será conectado.	Valor numérico	5
Patience	Número de ciclos que o algoritmo aceita ficar sem melhorar acurácia.	Valor numérico	25 - 150

Fonte: elaborado pelo próprio autor

Inicialmente a arquitetura utilizada foi o padrão disponível na biblioteca *PyG*. Esse código está ilustrado na figura 13.

Figura 13 - Código do modelo de classificação de nuvens de pontos *PointNet*

```python
class PointNetLayer(MessagePassing):
    def __init__(self, in_channels: int, out_channels: int):
        super().__init__(aggr='max')

        self.mlp = Sequential(
            Linear(in_channels + 3, out_channels),
            ReLU(),
            Linear(out_channels, out_channels),
        )

    def forward(self,
        h: Tensor,
        pos: Tensor,
        edge_index: Tensor,
    ) -> Tensor:
        return self.propagate(edge_index, h=h, pos=pos)

    def message(self,
        h_j: Tensor,
        pos_j: Tensor,
        pos_i: Tensor,
    ) -> Tensor:
        edge_feat = torch.cat([h_j, pos_j - pos_i], dim=-1)
        return self.mlp(edge_feat)
```

```python
class PointNet(torch.nn.Module):
    def __init__(self):
        super().__init__()

        self.conv1 = PointNetLayer(3, 32)
        self.conv2 = PointNetLayer(32, 32)
        self.classifier = Linear(32, len(unique))

    def forward(self,
        pos: Tensor,
        edge_index: Tensor,
        batch: Tensor,
    ) -> Tensor:

        h = self.conv1(h=pos, pos=pos, edge_index=edge_index)
        h = h.relu()
        h = self.conv2(h=h, pos=pos, edge_index=edge_index)
        h = h.relu()
        h = global_max_pool(h, batch)

        return self.classifier(h)
```

Fonte: elaborado pelo próprio autor

Mantendo a arquitetura fixa, as variáveis do quadro 9 foram modificadas até encontrar experimentalmente os melhores valores ou opções. No caso da técnica *class_weights* não existiu vantagem significativa em utilizá-la. O valor de *patience* muda de acordo com a arquitetura do modelo, pois, há arquiteturas que precisam de mais iterações para aprender do que outras.

Em seguida, fixado os valores do quadro 9, a busca ocorre pela melhor organização de camadas, filtros e técnicas de treinamento. Existem 5 tipos de camadas nesse contexto: camadas de convolução, camadas de *ReLu*, camadas de *pooling*, camadas de *dropout* e camadas de classificação. Por exemplo, na figura 13 a classe *PointNetLayer* é utilizada para criar uma camada de convolução no padrão *PointNet++*. Essa classe recebe os parâmetros de entrada e o número de filtros da camada.

Ainda na figura 13 a classe *PointNet* é o modelo de classificação em si. Nele existem 2 camadas de convolução criadas pela classe *PointNetLayer* ambas com 32 filtros em cada camada. Há também 2 camadas de *ReLu* para acrescentar não linearidade ao treinamento. Elas são posicionadas sempre após as camadas de convolução seguindo o padrão do estado da arte. Por último ocorre uma camada de *pooling* e uma camada de classificação.

Ademais, camadas de *dropout* buscam evitar o efeito de *overfitting*. Esse problema ocorre quando o modelo aprende e se especializa no conjunto de treino. A partir disso o algoritmo tem baixo desempenho quando faz predições de nuvens de pontos que nunca viu. Esse efeito é identificado por uma alta acurácia no conjunto de treino e uma baixa exatidão no

conjunto de teste. Para tratar esse problema, camadas de *overfiting* buscam desconectar algumas arestas do grafo. Isso obriga o algoritmo a aprender relações de forma genérica.

Cerca de 300 testes foram executados buscando a melhor arquitetura. Em cada um deles, modificou-se parte do código de treinamento e isso inclui mudanças nos hiperparâmetros e nas variáveis da própria arquitetura como número de camadas, funções e entre outros. Desses testes, 61 combinações foram registradas na planilha de testes que poderá ser visualizada na página do *GitHub* (seção 4.4). Nela, se destaca a existência de 5 arquiteturas: *PointNetPPF, PointNetLayer, PointNetLayer + DPT0, PointNetLayer + DPTS, PointNetLayer + DPTA*. A principal diferença entre elas é a função utilizada para camadas de convolução e a forma de execução do *dropout*.

O quadro 10 exemplifica algumas combinações existentes na planilha de testes e ilustra como a análise comparativa foi possível. Nele *cc* indica o número de camadas de convolução, *drop* refere-se ao uso de camadas de *dropout, shuf* refere-se ao uso do embaralhamento do banco de dados antes da divisão 70/30, *class* é o número de classes que o algoritmo irá predizer, filtros são a quantidade de filtros em cada uma das camadas de convolução (*cc*) e por fim as funções de convolução utilizadas para aquele modelo.

Quadro 10 - Exemplo das características dos modelos avaliados para classificação de nuvens de pontos

Modelo	**Filtros de convolução**	***cc***	***Drop***	***shuf***	***class***	**Funções de convolução**
A	32,32,32	3	Não	Não	6	*PointNetConv*
B	32,32,32	3	Não	sim	6	*PointNetPPF*
C	32,64,32	3	Não	Sim	26	*PPFConv*
D	32,64,128,64,32	5	Sim	Sim	26	*PointNetLayer*
E	32,64,128,256	4	não	sim	26	*PointNetConv*

Fonte: elaborado pelo próprio autor

Uma análise comparativa é feita a partir da coluna de acurácia que foi omitida nesse quadro. As funções *PointNetConv* e *PPFConv* são ambas fornecidas pela biblioteca *PyG*. *PointNetConv* está embasada no trabalho do modelo *PointNet* de Qi (2017) enquanto *PPFConv* é uma função baseada no estudo de Deng (2018) e sua rede *PPFNet*. Em resumo, ambas formam

camadas de convolução. A partir de tentativa e erro verificou-se que a combinação delas é efetiva.

Dropout

A estratégia utilizada para o *dropout* foi aplica-lo em vários cenários: entre as camadas, antes da camada de *pooling*, aplicar uma camada única e aplicar camadas independentes. Conforme será documentado na seção de resultados, as melhores camadas de *dropout* ocorrem quando aplicadas entre as camadas de convolução e com baixo percentual de *dropout*. Esse percentual indica a chance de um nó ser desconectado da rede e na maior parte dos testes foi configurado em cerca de 10%. Esse valor é padrão na literatura que sugere entre 10% a 30% de taxa de *dropout*. Existem 3 modelos de aplicação de *dropout*.

São eles: *DMLP, FMLP, DFMLP, Esp DFMLP*. *DMLP* é a técnica de *dropout* aplicada dentro da *MLP* (*Multi Layer Perceptron*) que está dentro da camada de convolução. *FMLP* é um *dropout* criado externamente a *MLP* e aplicada entre as camadas de convolução. *DFMLP* é uma combinação das últimas duas. Por fim *Esp DFMLP* utiliza a *DFMLP* com taxa de *dropout* interno ao *MLP* igual a 10%, taxas de *dropout* entre camadas de 7% e 10%. A definição dessas taxas ocorre de forma experimental com tentativa e erro. As demais combinações utilizam 10% de *dropout* em todas as posições. O quadro 11 ilustra a diferença de aplicação dentro da convolução e entre as convoluções.

Quadro 11 - Diferença entre o dropout aplicado dentro ou fora do MLP

Dentro da MLP	**Fora da MLP**

`self.conv1 = PPFConv(`
`  local_nn=MLP([entryChannel + 4, h1, h1], dropout=dropout_rate),`
`  global_nn=MLP([h1, h1],dropout=dropout_rate)`
`)`
	`x = self.conv1(x=pos, pos=pos, normal=normal, edge_index=edge_index)`
	`x = F.relu(x)`
	
	`x = self.dropout1(x)`
	
	`x = self.conv2(x=x, pos=pos, edge_index=edge_index)`
	`x = F.relu(x)`

Fonte: elaborado pelo próprio autor

Data augmentation

A implementação de algumas técnicas de expansão do banco de dados (*data augmentation*) foi essencial para o alcançar os resultados obtidos. Essas técnicas tem como princípio, modificar as nuvens de pontos do banco de dados de tal forma que elas possam ser consideradas novas nuvens para o banco. Algumas das possibilidades é rotacionar, transladar,

acrescentar ruídos e acrescentar ou remover pontos da nuvem. A combinação entre essas técnicas também é possível.

Dois novos bancos de dados foram criados a partir do original utilizando *data augmentation*. O primeiro banco foi criado a partir de 2 técnicas: a inserção de ruído branco aos pontos e a exclusão aleatória de 30% dos pontos. Cada técnica gera 631 novas nuvens para o *dataset* totalizando 1893 nuvens no banco **A**. Já o banco **B** possui 4 técnicas: adição de ruído, exclusão de pontos, adição de ruído mais exclusão de pontos e adição de pontos.

Portanto, além de combinar as duas técnicas do primeiro banco numa terceira, o banco **B** também possui nuvens com mais densidade de pontos. Dado um ponto da nuvem **P1**, a quarta técnica cria um ponto entre **P1** e seus vizinhos mais próximos. Por fim o banco **B** possui 3155 nuvens de pontos.

Redução de classes

O *dataset* original possui 26 classes que descrevem objetos urbanos. São bancos, paredes, postes, carros, pessoas, ciclistas, motos, caminhões, caminhonetes e entre outros elementos. Num contexto de direção autônoma as decisões que são tomadas não dependem da identificação de todos os tipos de elementos presentes nas ruas. Entretanto, é muito importante identificar pessoas, carros, motos, sinais de trânsito e estruturas no geral.

Visando obter simplicidade, as classes foram reduzidas de 26 para 6. São elas: pessoa, veiculolongo, veiculo2rd, veiculo4rd, transito, outro. Se a classificação possui este foco, diversos fatores contribuem para o crescimento da exatidão do modelo. Primeiro, o modelo terá menos opções e estatisticamente fica mais fácil acertar. Segundo, haverá mais exemplos no banco para cada tipo de classe logo tornando o banco mais homogêneo. Por fim a redução de classes aglomera todas as nuvens que são muito parecidas em algum nível.

Por exemplo, sinais de trânsito e semáforos são classes aglomerados na classe “transito”. Veículos como caminhões, carretas, ônibus são aglomerados na classe “veiculolongo”. Estruturas e elementos gerais urbanos como sombrinha, bancos, paredes, postes e latas de lixo são agrupados na classe “outros”. Apesar da classe “outros” ser formada de diferentes elementos, que diferem em formato, eles surgem como perfeito exemplo do que não seria um carro para o modelo. Logo, também beneficia o processo de aprendizagem.

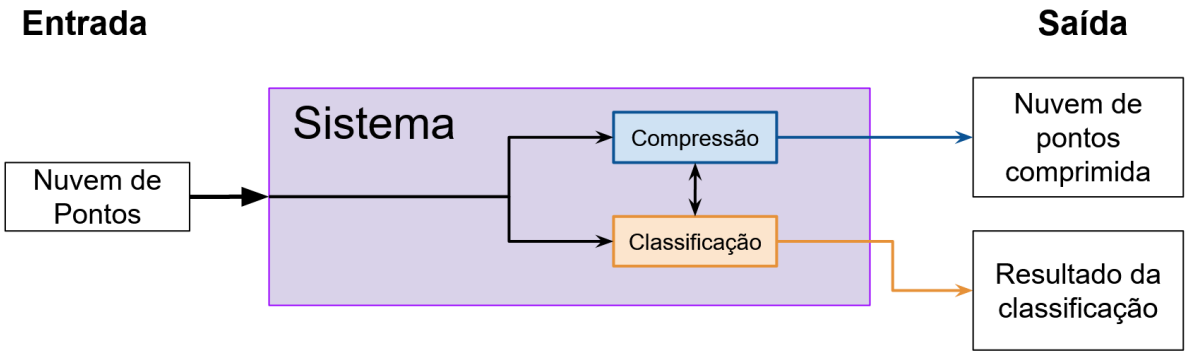
Por fim, é importante destacar que as arquiteturas dos modelos foram desenvolvidas em diferentes cenários. Inclusive mudando o número de classes de 26 para 6. Ou seja, os modelos

obtidos como resultado desse trabalho possuem tanto a capacidade de classificar 26 classes como de classificar 6 classes.

4.4 Sistema de Classificação e Compressão

O sistema trata-se de um conjunto de algoritmos capazes de trabalhar com compatibilidade para classificar e comprimir nuvens de pontos em qualquer ordem. Ou seja, deve ser possível comprimir e classificar, assim como deve ser possível classificar e comprimir. Ele também precisa ser capaz de executar essas funções isoladamente se necessário. Dito isso os algoritmos integram um sistema capaz de desempenhar essas duas funções em diferentes contextos. Em uma visão macro do ambiente de aplicação desses códigos, a figura 14 ilustra como o sistema pode operar.

Figura 14 - Diagrama do sistema de classificação e compressão



Fonte: elaborado pelo autor

Inicialmente, a capacidade de compressão é essencial para o uso de nuvens de pontos em cenários gerais diversos. Pois, isso implica numa maior facilidade de transmissão, armazenamento e processamento. O sistema desenvolvido possui um algoritmo de compressão que cumpre com essa característica generalista.

Já a classificação de objetos foi elaborada com o banco de dados da *Sydney Urban Objects* que possui foco em elementos urbanos em um contexto real com sensores LIDAR. Isso traz uma ênfase para o contexto de direção autônoma, contudo, não limita o uso dos modelos em qualquer outro cenário de classificação.

4.5 Github

A tecnologia *git*, plataforma *Github*, fornece um modo organizado e estruturado para versionamento de código. Sua principal capacidade é possibilitar que diferentes pessoas trabalhem sobre o mesmo projeto sem que ocorram conflitos ao fim das contribuições individuais. Além disso, ele opera como repositório público. Foi possível divulgar os resultados obtidos na pesquisa com a criação de uma página na plataforma. A partir da formatação de um arquivo *readme.md* tabelas, imagens e textos desse documento foram organizados. Assim que o trabalho for registrado a página do *Github* se tornará pública contendo:

- tabelas de resultados de compressão;
- tabelas de resultados de treinamento de modelos;
- versão final do presente trabalho;
- códigos de quantização vetorial;
- códigos de classificação de nuvens de pontos;

O link para acessar a página é: https://github.com/DaviMelo558/LIDAR-Point-Cloud-Compression-and-Classification

5 RESULTADOS E DISCUSSÃO

Os resultados estão divididos em 4 seções: resultados da compressão de nuvens de pontos, resultados da classificação de nuvens de pontos sem compressão, resultados da classificação de nuvens de pontos com compressão e por fim uma breve avalição entre sistemas. Primeiro será abordada a compressão isoladamente, em seguida a classificação também de forma isolada. Em seguida, o impacto da compressão na informação de cor e de posição serão mensurados na terceira seção.

5.1 Quantização Vetorial

Os experimentos elaborados se baseiam em dois tipos. Primeiro, a avaliação numérica utilizando das métricas de qualidade, taxas e fatores de compressão. Em segundo a avaliação visual a partir da reconstrução das nuvens de pontos.

Compressão das cores

O primeiro experimento sintetizado no apêndice A mostra a reconstrução das cores da nuvem de pontos Bumbameuboi utilizando diferentes valores de K (nível da quantização).

Nesse experimento o tamanho de grupo está fixo em 16 (*tamG*=16). Variando *K* em potências de 2, desde *K*=2 até *K*=1024 é possível observar a qualidade das cores da nuvem melhorar significativamente.

K o nível de quantização é o número de vetores no dicionário elaborado pelo *K-Means*. Portanto, quanto maior *K*, mais vetores descrevem as cores da nuvem, consequentemente mais precisa é a reconstrução. Em níveis de quantização superiores a 128 a mudança nas cores é imperceptível. Portanto, com essa distância e ângulo de visão da nuvem a qualidade é satisfatória com *K*=128. O objetivo intrínseco desses testes será sempre encontrar o menor *K* possível que atinja a maior qualidade.

De mesma forma esse experimento foi repetido para a nuvem de pontos *Guy*. O apêndice B sintetiza os resultados da variação de *K* e reconstrução da nuvem de pontos. Ademais, *K*=128 também obteve o mesmo resultado visual. Ou seja, para valores de *K* maiores que 128 a melhora nas cores é imperceptível. O segundo experimento trata-se da análise numérica das métricas de qualidade, taxas e fatores. Esses dados podem ser visualizados na tabela 1 para a nuvem de pontos Bumbameuboi.

Tabela 1 - Métricas Avaliativas Bumbameuboi

K	RMSE	PSNR	SSIM	FH	FQ	FQH
2	32.11	17.9	0.59	2.3	289.9	2319.22
4	26.15	19.7	0.69	2.3	289.9	1323.68
16	21.99	21.21	0.76	2.3	207.91	586.37
32	20.6	21.78	0.79	2.3	173.34	466.7
64	19.27	22.36	0.81	2.3	154.29	393.37
128	17.94	22.98	0.82	2.3	136.63	339.27
256	16.84	23.53	0.84	2.3	113.17	281.65
512	15.54	24.24	0.86	2.3	104.19	253.95
1024	14.04	25.12	0.88	2.3	99.48	236.14

Fonte: elaborado pelo próprio autor

Uma análise comparativa pode ser traçada com a tabela 2 que ilustra os mesmos parâmetros para a nuvem de pontos *Guy.*

Tabela 2 - Métricas Avaliativas Guy

K	RMSE	PSNR	SSIM	FH	FQ	FQH
2	33.93	17.51	0.51	2.3	293.82	2350.55
4	27.87	19.22	0.56	2.3	293.82	1175.27
16	24.57	20.32	0.62	2.3	204.09	555.93
32	23.54	20.69	0.64	2.3	181.62	480.7
64	22.21	21.2	0.66	2.3	159.25	404.71
128	21.01	21.68	0.68	2.3	138.61	340.44
256	19.04	22.37	0.71	2.3	113.46	285.78
512	17.71	23.16	0.74	2.3	106.54	260.16
1024	15.05	24.58	0.79	2.3	101.28	240.35

Fonte: elaborado pelo próprio autor

Para evidenciar a relação entre compressão e qualidade da reconstrução da nuvem de pontos o apêndice C reuni os resultados da tabela 1 e 2 em forma de gráficos.

Note que, para cada valor de *PSNR* ou *SSIM* existe um *K* equivalente. Quanto maior *K* maior é a similaridade da reconstrução e menor são as taxas e fatores de compressão. A partir dos resultados desses experimentos é necessário avaliar alguns cenários.

A primeira possibilidade ocorre no caso de um sistema que permite perdas e visa trabalhar com nuvens de pontos compactadas na maior taxa possível enquanto mantem o significado das cores da nuvem. O algoritmo de quantização vetorial desse estudo pode oferecer nesse caso o nível de quantização *K*=4 com uma taxa de 99,9% de compressão. Apesar da opacidade obtida no resultado do apêndice A e B, esse nível de quantização permite a reconstrução básica das cores.

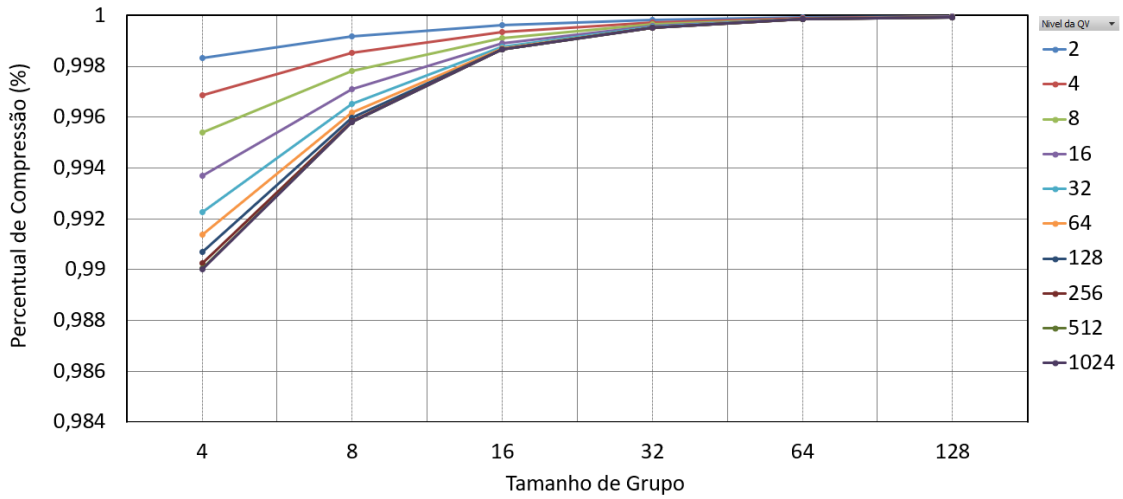
Uma segunda opção é a de favorecer a qualidade enquanto se mantem a maior compressão possível. Nesse caso *K*=128 entrega uma taxa de compressão de 99,7% enquanto permite uma similaridade de 82%. Ademais, é necessário explicitar a contribuição da codificação de Huffman aliada a quantização vetorial. Considerando apenas a nuvem de pontos *Guy*, o maior valor de fator de compressão obtido pela quantização vetorial isoladamente com *K*=2 é 293.82. O menor valor de compressão obtido pela quantização aliada a técnica de Huffman é 240.35. Em termos de taxa de compressão seria TQ=99,66% e TQH=99,56%.

No primeiro caso a similaridade é 51% enquanto no outro é 79%, uma diferença percentual de 28%. Portanto, a quantização vetorial aliada a técnica de Huffman possui

capacidades significativamente melhores. Ela é capaz de alcançar maiores taxas de compressão enquanto mantem a qualidade e similaridade alcançada pelo método. Os testes seguintes visam confirmar essas capacidades aplicando a quantização para uma maior quantidade de nuvem de pontos.

O algoritmo foi executado para todas as 631 nuvens de pontos do banco de dados da *SydneyUrbanObjects* variando *K* e o tamanho de grupo. Os níveis de quantização *K* variam de 2 até 1024 sempre em potências de 2. Já os tamanhos de grupo (*tamG*) foram especificamente 4, 8, 16, 32, 64 e 128. Para cada nuvem, *K* e *tamG* uma taxa (TQH) foi calculada totalizando 37.860 taxas de compressão. A figura 15 ilustra a média dessas taxas de compressão equivalente para cada *K* e tamanho de grupo.

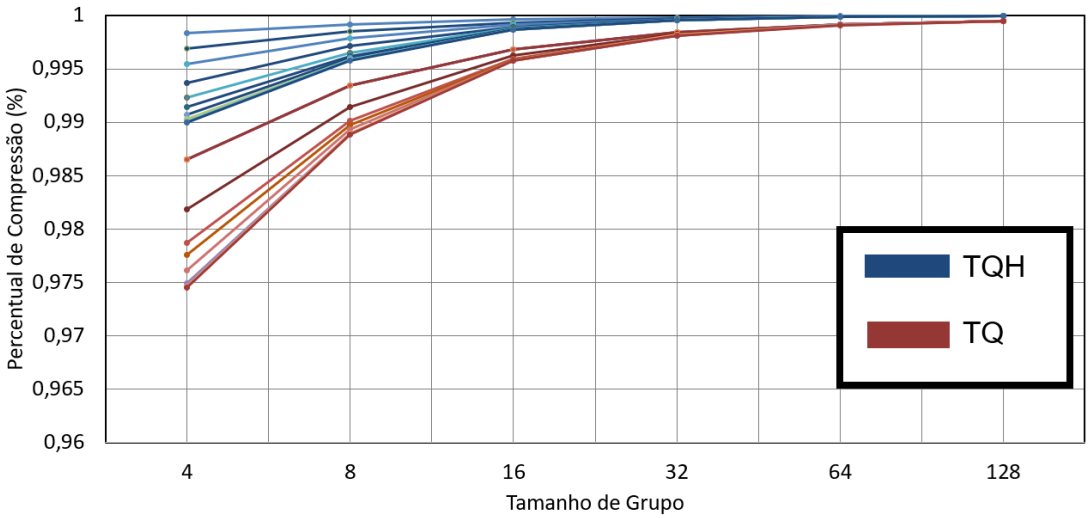
Figura 15 - Média do TQH (taxa de compressão QV + Huffman) para cada K e tamanho de grupo na compressão de cores das nuvens de pontos



Fonte: elaborado pelo próprio autor

Já a figura 16 traz uma análise comparativa entre TQ e TQH na qual fica claro o impacto da codificação de Huffman nesse processo.

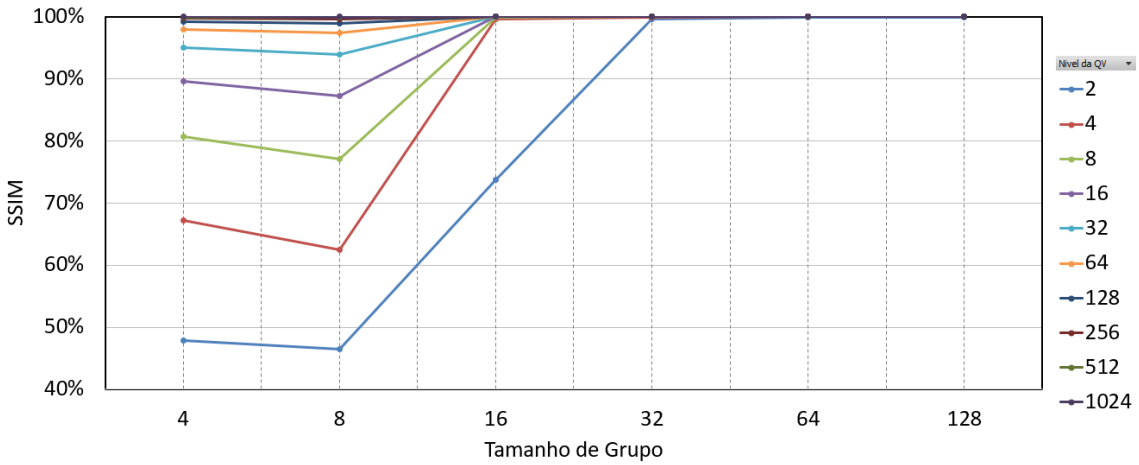
Figura 16 - Análise comparativa entre as taxas TQH e TQ variando K e tamanho de grupo na compressão de cores das nuvens de pontos



Fonte: elaborado pelo próprio autor

A qualidade de reconstrução das nuvens também foi avaliada no mesmo contexto entre *K* e *tamG*. A figura 17 ilustra a média de *SSIM* por *K* e *tamG*.

Figura 17 - Média do SSIM (qualidade da reconstrução) para cada K e tamanho de grupo na compressão de cores das nuvens de pontos



Fonte: elaborado pelo próprio autor

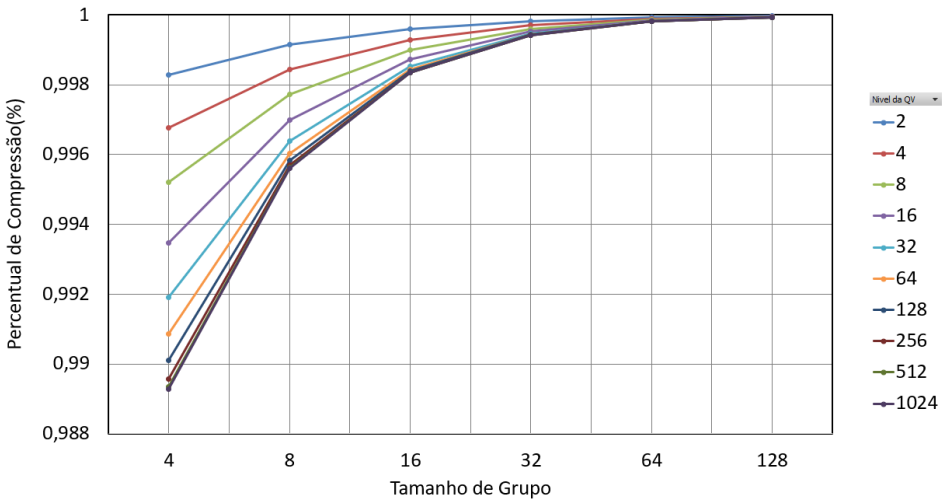
Esses resultados coincidem com os primeiros experimentos elaborados para duas nuvens de pontos isoladamente. A partir desses dados é possível afirmar que o algoritmo de compressão elaborado possui consistência e permite altas taxas de compressão enquanto

mantem à similaridade acima de 90%. A planilha com todos os cálculos e resultados estará disponível no github apresentado na seção 4.4.

Compressão das posições

Os experimentos da compressão de coordenadas foram executados diretamente sobre o banco de dados da *SydneyUrbanObjects*. Observa-se no primeiro deles a principal taxa de compressão TQH em função de *K* e *tamG* figura 18.

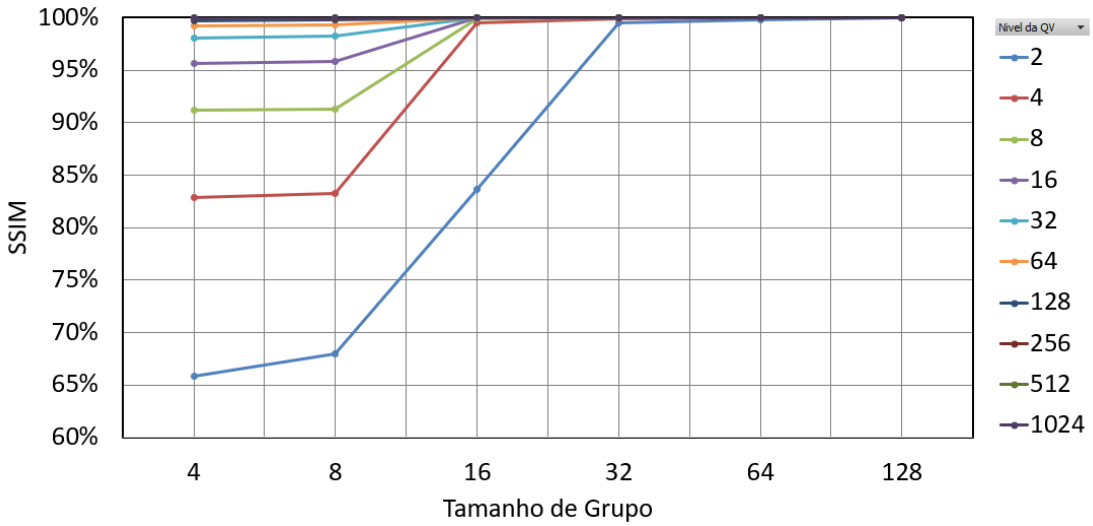
Figura 18 – Média do TQH (taxa de compressão QV + Huffman) para cada K e tamanho de grupo na compressão de coordenadas dos pontos das nuvens



Fonte: elaborado pelo próprio autor

Em comparação com a quantização de cores não existe diferença significativa entre as taxas de compressão. Em seguida, assim como executado para cores foi calculado o *SSIM* médio para cada *K* e *tamG* figura 19.

Figura 19 – Média do SSIM (qualidade da reconstrução) para cada K e tamanho de grupo na compressão de coordenadas dos pontos das nuvens



Fonte: elaborado pelo próprio autor

Analogamente a compressão de cores, o *SSIM* com poucos vetores no dicionário tem sérios problemas de similaridade. Contudo, rapidamente médias de *SSIM* iguais a 100% são alcançadas conforme *K* ou *tamG* aumenta. O apêndice D ilustra a diferença visual de qualidade na reconstrução da nuvem de pontos conforme *tamG* muda e *K* é fixo em 32. Nessa perspectiva é possível verificar como a posição é um dado muito sensível.

Mesmo que a média de *SSIM* para *K*=32 esteja sempre acima de 95% conforme mostra o gráfico da figura 19, essa nuvem reconstruída possui aspecto significativamente diferente da original. Conforme o tamanho de grupo aumenta a nuvem perde aspecto cúbico e passa a descrever as especificidades do carro capturado pelo sensor *LIDAR*.

5.2 Classificação De Nuvens De Pontos

A tabela 3 lista em ordem decrescente de acurácia os modelos treinados e suas características. A partir deles é possível compreender quais técnicas foram melhores e os parâmetros que mais impactam durante o treinamento.

Tabela 3 - 10 Melhores modelos de classificação

Nome do Arquivo	**Filtros de Convolução**	***cc***	***Acc***	***Drop***	***Aug***	***Shuf***	***Class***
Grouped	32,32,32	3	97,38%	Não	Não	Não	3
Grouped_Aug2	32,32,32	3	89,77%	Não	Sim	Sim	6
Grouped	32,64,32	3	88,02%	Sim	Não	Sim	6
Grouped_Aug1	32,64,128,64,32	5	87,04%	Sim	Sim	Sim	6
Grouped	32,64,32	3	86,46%	Sim	Não	Não	6
Grouped	32,64,32	3	84,9%	Sim	Não	Sim	6
Grouped	32,32,32	3	84,38%	Sim	Não	Sim	6
Aug1	32,64,32	3	84,23%	Sim	Sim	Sim	26
Grouped	32,64,32	3	83,33%	Não	Não	Não	6
Grouped	32,64,128,256	4	82,29%	Não	Não	Não	6

Fonte: elaborado pelo próprio autor

Alguns padrões destacam-se na tabela 3. Nove dos dez modelos utilizam a redução de classe e o melhor modelo possui o menor número de classes para classificar. Seis utilizam camadas de *dropout*. Oito modelos utilizam apenas 3 camadas de convolução (coluna *cc*) variando entre (32,32,32) e (32,64,32) para os valores de filtros. O único modelo capaz de classificar 26 classes diferentes utiliza camada de *dropout*, *data augmentation* e *shuf*. Ademais, o modelo com 3 classes (coluna *class*) classifica se a nuvem de pontos são “veiculo”, “pessoa” ou “outro”.O arquivo Classification_PC.py exemplifica o treinamento e estará disponível na página do Github (seção 4.4).

Camadas de convolução

Durante os testes a maior diferença entre os modelos inicialmente estava em quais funções de convolução eram utilizados. Após a sequência de testes experimentais duas funções de convolução específicas se destacam: *PointNetConv* e *PPFConv*. Quando utilizadas em conjunto dão resultado a base da arquitetura desenvolvida nesse trabalho chamada *PointNetPPF*. Todos os modelos da tabela 3 utilizam a combinação dessas duas funções.

A estrutura básica da rede *PointNetPPF*, sem *dropout*, é ilustrado na figura 20.

Figura 20 - Código do modelo *PointNetPPF*

```
class PointNetPPF(torch.nn.Module):
    def __init__(self, num_classes=len(unique), entryChannel = 3, h1=32, h2=32, h3=32):
        super().__init__()

        self.conv1 = PPFConv(
            local_nn=MLP([entryChannel + 4, h1, h1] ),
            global_nn=MLP([h1, h1])
        )
        self.conv2 = PointNetConv(
            local_nn=MLP([h1 + 3, h2, h2]),
            global_nn=MLP([h2, h2])
        )
        self.conv3 = PointNetConv(
            local_nn=MLP([h2 + 3, h3, h3] ),
            global_nn=MLP([h3, h3])
        )

        self.classifier = torch.nn.Linear(h3, num_classes)


    def forward(self, pos: Tensor, edge_index: Tensor, batch: Tensor, normal: Tensor = None) -> Tensor:
        if normal is None:
            normal = torch.ones_like(pos) # Default fallback

        x = self.conv1(x=pos, pos=pos, normal=normal, edge_index=edge_index)
        x = F.relu(x)

        x = self.conv2(x=x, pos=pos, edge_index=edge_index)
        x = F.relu(x)

        x = self.conv3(x=x, pos=pos, edge_index=edge_index)
        x = F.relu(x)

        x = global_max_pool(x, batch)
        return self.classifier(x)
```

Fonte: elaborado pelo próprio autor

São 3 camadas de convolução, a primeira é uma camada *PPFNet*, a segunda e a última camada de convolução são *PointNet*. A *PPFConv* precisa informações de vetores normais a superfície. O cálculo de vetores normais não foi desenvolvido durante o estudo, mas a *PPFConv* continua sendo efetiva entregando apenas a posição dos pontos *x,y,z*. Após cada camada de convolução uma camada de *ReLu* é aplicada. Por fim, uma camada de *pooling* e uma camada de classificação finalizam a sequência do modelo.

Número de camadas e filtros

A arquitetura de um modelo é definida, em grande parte, pelo número de camadas e filtros. Conforme pôde ser verificado na tabela 3 os melhores modelos utilizam 3 camadas com filtro de tamanho único 32,32,32 ou com leve variação no filtro interno 32,64,32. Entretanto, uma maior quantidade filtros geralmente fornece capacidade ao modelo para identificar padrões complexos das nuvens de pontos.

No contexto da pesquisa, não foi possível testar modelos com quantidades maiores de filtros devido a limites de poder computacional. Dado essa limitação que existe tanto na

plataforma *Google Colab* como na máquina local, a maioria das combinações de camadas elaboradas possuem poucos filtros. Especificamente o *Google Colab* permite treinamentos de no máximo 12h, contudo, qualquer instabilidade na rede quase sempre finaliza o processo.

Por fim, a atividade de classificação no contexto urbano pôde ser bem desenvolvida com 3 camadas e poucos filtros variando entre 32 e 64. Isso provavelmente ocorre devido a quantidade de pontos que as nuvens desse banco de dados possuem. Nele existem desde nuvens com 23 pontos até cerca de 13 mil pontos. Valores relativamente baixos comparados as grandes nuvens de pontos do banco de dados da USP.

Redução de classes

A redução de classes foi verificada como uma possibilidade para simplificar o cenário de classificação e aumentar a acurácia. Conforme a tabela 3 mostra, quanto menor o número de classes (coluna *class*) maior a acurácia. Quanto mais classes, mais complexo a tarefa e mais difícil se torna o treinamento.

Data augmentation e Shuf

As técnicas de expansão do banco de dados foram efetivas em todos os testes. Conforme a tabela 3 exibe, entre os algoritmos de classificação que possuem como saída 6 classes, o melhor modelo possui *data augmentation* (coluna *Aug*). Além disso, o melhor modelo elaborado para a classificação com 26 classes também utiliza *data augmentation*. Logo, elas são técnicas indispensáveis para o treinamento de modelos de classificação.

Ademais, o embaralhamento (coluna *shuf*) é um processo essencial principalmente quando há uma expansão do banco de dados. Baseado nos 300 testes experimentais e na planilha com os 61 principais testes é possível afirmar que: com 631 nuvens no banco de dados e certas classes possuindo apenas 1 instância o embaralhamento antes da divisão 70/30 torna-se irrelevante. Contudo, quando a técnica de *data augmentation* é aplicada e o banco possui 1893 ou 3155 nuvens de pontos, o embaralhamento possui impacto de 2 a 4 pontos percentuais na acurácia dos modelos.

Por fim, novamente, os melhores modelos de classificação obtidos para 26 e 6 classes fazem uso do *data augmentation* em conjunto com o embaralhamento.

Dropout

As camadas de *dropout* podem variar muito. É possível aplicar *dropout* sobre as conexões do grafo que representa a nuvem, seja ao nó ou para as arestas. No contexto da pesquisa o *dropout* ocorre desativando momentaneamente alguns nós da rede de treinamento,ou seja, das redes *MLP's*. De acordo com os resultados da tabela 3 observa-se que o *dropout* está sendo utilizado no segundo melhor modelo para 6 classes e também no melhor modelo para 26 classes. Além disso 6 modelos entre os 10 melhores utilizam *dropout*. Isso demonstra a relevância da técnica que permite a criação de algoritmos mais adaptados a novos dados.

Principais Modelos

Existem 3 modelos principais como resultado da pesquisa. O modelo A com 89,77% de acurácia para predição de 6 classes. O modelo B com 88,02% de acurácia na predição de 6 classes. Por fim, o modelo C com 84,23% de acurácia que classifica entre as 26 classes diferentes do banco de dados original. A tabela 4 resume as características dos 3 modelos.

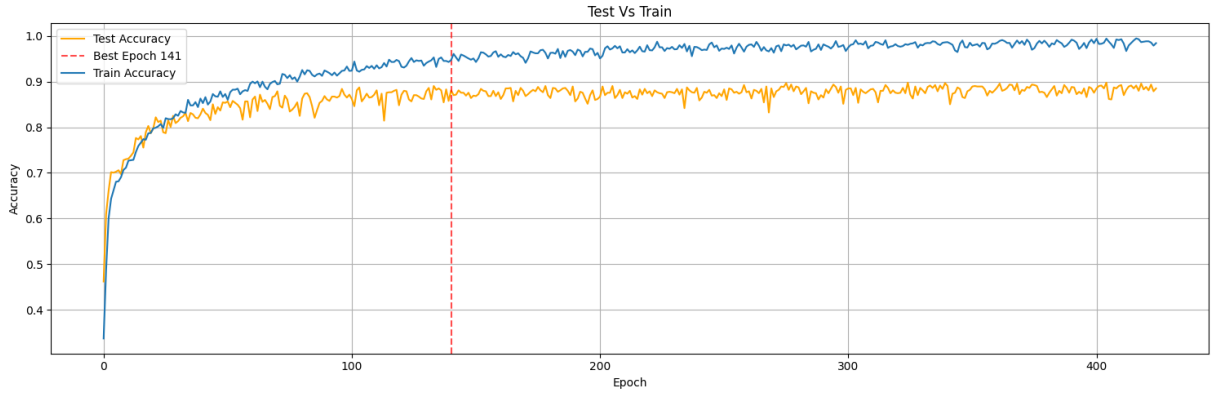
Tabela 4 - Principais modelos e suas características

Modelo	**Filtros de Convolução**	***cc***	***Acc***	***Drop***	***Aug***	***Dataset***	***Shuf***	***Class***
A	32,32,32	3	89,77%	Não	Sim	3155	Sim	6
B	32,64,32	3	88,02%	Sim	Não	631	Sim	6
C	32,64,32	3	84,23%	Sim	Sim	1893	Sim	26

Fonte: elaborado pelo próprio autor

Todos utilizam como base a arquitetura *PointNetPPF* da figura 20. O modelo A, com a maior acurácia, utiliza a exata mesma estrutura da figura 20. São 3 camadas de convolução, todas com 32 filtros. Uma camada de *ReLu* após cada camada de convolução. A primeira convolução ocorre com a *PPFConv* e as últimas duas com a *PointNetConv*. Por fim, ocorre uma camada de *pooling* e uma de classificação. Para o modelo A não foi utilizado nenhuma camada de *dropout*. A figura 21 ilustra a evolução da acurácia no treinamento do modelo.

Figura 21 - Comparação entre acurácia de teste e acurácia de treino (modelo A)

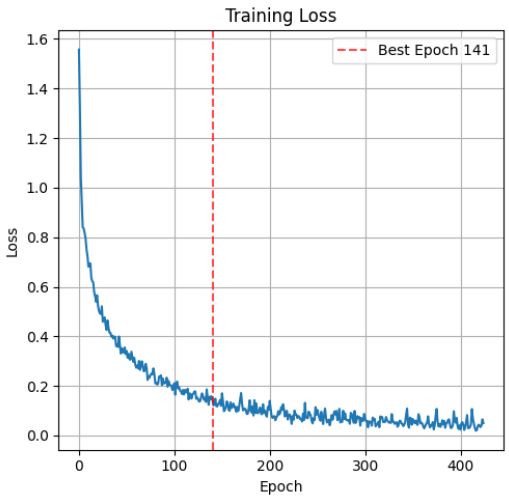


Fonte: elaborado pelo próprio autor

Nesse contexto é importante destacar a diferença entre acurácia(*accuracy*) de treinamento e acurácia de teste. As classes das nuvens no banco de treino são conhecidas pelo algoritmo, as iterações e correções existentes no modelo são desenvolvidas a partir disso. Já no conjunto de teste o algoritmo não sabe a classe da nuvem em momento algum. Obter exatidão alta no banco de teste significa que o algoritmo é capaz de lidar com nuvens que nunca viu, ou seja, que não obteve informação prévia.

Apesar da acurácia no banco de treino continuar evoluindo ao longo das épocas, a acurácia no banco de teste fica saturada próximo à época 150. O gráfico de *loss* na figura 22 exemplifica o significado da métrica.

Figura 22 - *Loss* do treinamento do modelo A



Fonte: elaborado pelo próprio autor

Conforme o algoritmo obtém maiores precisões a *loss* diminui até saturar próximo de zero. Ademais, a coluna *Support* possui a quantidade de nuvens de pontos no banco de teste para cada classe e em português seria o "apoio" que aquela classe possui no conjunto de teste. A tabela 5 traz algumas métricas essenciais para análise de acurácia por classe.

Tabela 5 - Análise por classe do (modelo A)

Classe	***Recall***	***Accuracy***	***F1-Score***	***Support***
veiculo4rd	95%	96%	95%	240
veiculo2rd	60%	71%	65%	20
veiculolongo	69%	83%	75%	42
pessoa	97%	94%	95%	233
transito	81%	77%	79%	147
outros	85%	86%	85%	266

Fonte: elaborado pelo próprio autor

Existem algumas mudanças relevantes entre as arquiteturas do modelo A e B. A figura 23 mostra o código.

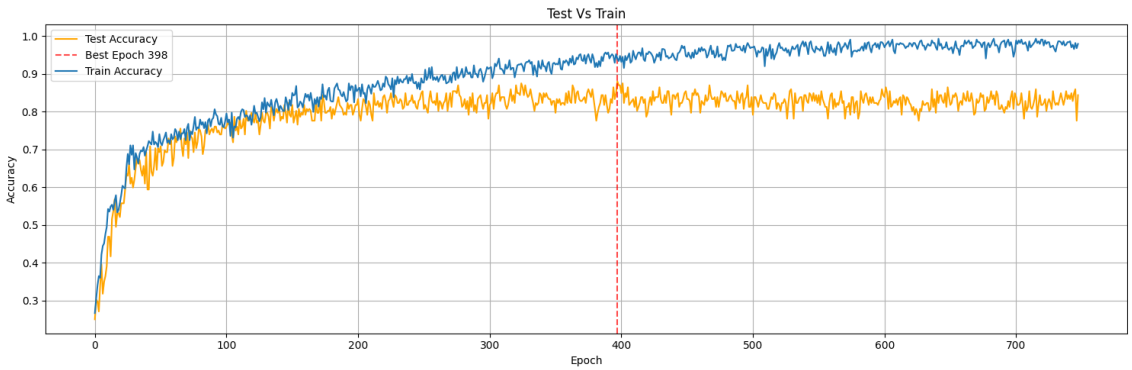
Figura 23 - Código do modelo *PointNetPPF* (modelo B)

```
    def forward(self, pos: Tensor, edge_index: Tensor, batch: Tensor, normal: Tensor = None) -> Tensor:
        if normal is None:
            normal = torch.ones_like(pos) # Default fallback

        x = self.conv1(x=pos, pos=pos, normal=normal, edge_index=edge_index)
        x = F.relu(x)

        if self.training:
            x = self.dropout1(x)

        x = self.conv2(x=x, pos=pos, edge_index=edge_index)
        x = F.relu(x)

        if self.training:
            x = self.dropout2(x)

        x = self.conv3(x=x, pos=pos, edge_index=edge_index)
        x = F.relu(x)

        if self.training:
            x = self.dropout3(x)

        x = global_max_pool(x, batch)
        return self.classifier(x)
class PointNetPPFby6(torch.nn.Module):
    def __init__(self, num_classes=len(unique), entryChannel = 3, h1=32, h2=64, h3=32, dpr=0.1, dpr1=0.07, dpr2=0.1, dpr3=0.07):
        super().__init__()

        self.conv1 = PPFConv(
            local_nn=MLP([entryChannel + 4, h1, h1], dropout=dpr),
            global_nn=MLP([h1, h1],dropout=dpr)
        )
        self.conv2 = PointNetConv(
            local_nn=MLP([h1 + 3, h2, h2],dropout=dpr),
            global_nn=MLP([h2, h2], dropout=dpr)
        )
        self.conv3 = PointNetConv(
            local_nn=MLP([h2 + 3, h3, h3], dropout=dpr),
            global_nn=MLP([h3, h3], dropout=dpr)
        )

        self.dropout1 = nn.Dropout(p=dpr1)
        self.dropout2 = nn.Dropout(p=dpr2)
        self.dropout3 = nn.Dropout(p=dpr3)

        self.classifier = torch.nn.Linear(h3, num_classes)
```

Fonte: elaborado pelo próprio autor

Ele executa predição de 6 classes e possui 88,02% de acurácia. Esse modelo implementa *dropout* dentro e fora das redes neurais *MLP's* com taxa padrão de 10% dentro das redes. Além disso, aplica diferentes taxas entre as camadas de convolução. Inicia com uma taxa de *dropout* de 7%, em seguida aumenta para 10% e por fim volta para 7%. Ademais, o padrão *PointNetPPF* ocorre exatamente como antes, camadas de *ReLu* após cada convolução e finaliza com uma camada de *pooling* seguida da classificação. A figura 24 mostra a evolução da acurácia do treinamento desse modelo.

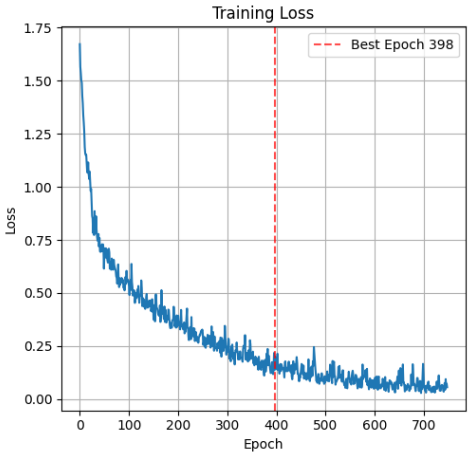
Figura 24 - Comparação entre acurácia de teste e acurácia de treino (modelo B)



Fonte: elaborado pelo próprio autor

Já figura 25 exibe o comportamento da *loss* ao longo das épocas de treinamento do modelo B.

Figura 25 - *Loss* do treinamento do modelo B



Fonte: elaborado pelo autor

Novamente, as métricas de treinamento por classe podem ser verificadas na tabela 6. Nesse caso, o modelo B possui 100% de recall na classe pessoa. Essa característica é essencial no contexto urbano. Significa que sempre que a nuvem de pontos é uma pessoa na rua o modelo identifica corretamente.

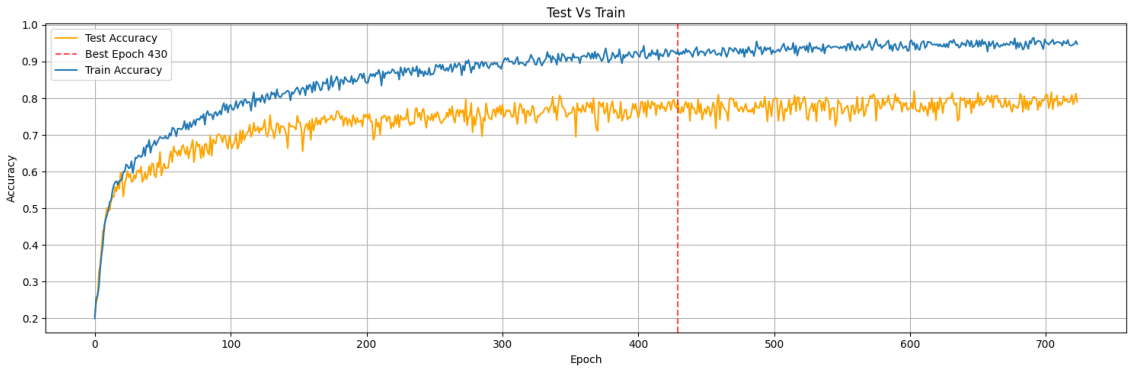
Tabela 6 - Análise por classe (modelo B)

Classe	Recall	Acurácia	F1-Score	Support
veiculo4rd	89,58%	91%	91%	48
veiculo2rd	75%	75%	75%	4
veiculolongo	44%	57%	50%	9
pessoa	100%	96%	98%	47
transito	60%	86%	71%	30
outros	87%	73%	80%	54

Fonte: elaborado pelo próprio autor

Por sua vez o modelo C possui a exata mesma estrutura que aparece na figura 23. Entretanto, sua taxa de *dropout* é 10% para todos os contextos. O resultado do treinamento desse modelo está sintetizado na figura 26.

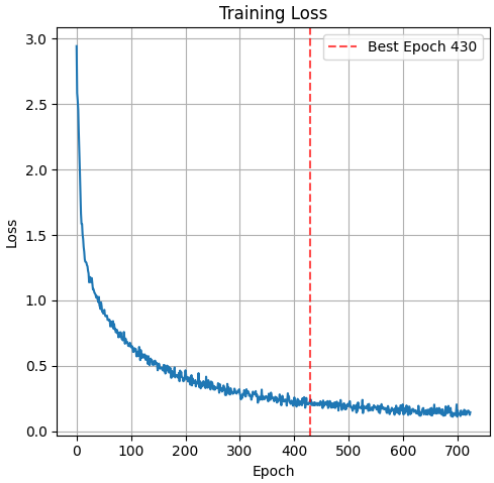
Figura 26 - Comparação entre acurácia de teste e acurácia de treino (modelo C)



Fonte: elaborado pelo próprio autor

Em destaque a saturação do modelo C ocorre em cerca de 80% de acurácia enquanto nos modelos A e B isso ocorre acima de 80%. Ademais, a *loss* do treinamento desse modelo pode ser observada na figura 27.

Figura 27 - Loss do treinamento do modelo C



Fonte: elaborado pelo autor

O treinamento da maioria dos modelos com alta acurácia precisam de muitas épocas de aprendizado. Esses treinamentos longos foram executados de 150 em 150 épocas. Sempre que a paciência igual a 150 finalizava o algoritmo o *Google Colab* mantinha os dados armazenados. O código então era executado novamente iniciando da acurácia que havia parado. Isso evita que o código passe mais de 12h em execução ultrapassando o limite de tempo da plataforma *Google Colab*.

Por fim, os dados com acurácia, *recall* e *f1-score* para cada classe é explicitado na tabela 7.

Tabela 7 - Análise por classe (modelo C)

Classes	***Acurácia***	***Recall***	***F1-Score***	***Support (test_dataset)***
4wd	50%	31,58%	39%	19
bench	33%	100%	50%	2
bicycle	67%	50%	57%	8
biker	100%	100%	100%	4
building	88%	77,78%	82%	18
bus	82%	60%	69%	15
car	77%	100%	87%	80
cyclist	0%	0%	0%	3
excavator	50%	100%	67%	1
pedestrian	90%	97,81%	94%	137
pillar	68%	83,33%	75%	18
pole	64%	73,68%	68%	19
post	89%	100%	94%	8
scooter	100%	100%	100%	1
ticket_machine	67%	66,67%	67%	3
traffic_lights	70%	86,05%	77%	43
traffic_sign	89%	71,74%	80%	46
trailer	100%	100%	100%	1
trash	100%	40%	57%	5
tree	100%	83,87%	91%	31
truck	100%	27,27%	43%	11
trunk	70%	56%	62%	50
umbrella	83%	100%	91%	5
ute	64%	60%	62%	15
van	65%	53,12%	59%	32
vegetation	100%	50%	67%	2

Fonte: elaborado pelo próprio autor

Apesar da acurácia ser baixa em algumas classes como *cyclist* e *bench*, o impacto dessas instâncias na acurácia final é menor dado que o *support* dessas classes é 5 no total.

5.3 Classificação Pós Compressão De Nuvem De Pontos

Até este trecho do estudo, todas as precisões referem-se à classificação do banco de dados original ou expandido. Nessa seção o impacto da compressão para a classificação de nuvens de pontos será mensurado.

Classificação pós compressão de cores

No contexto da arquitetura *PointNetPPF* a compressão de cores não impacta a classificação de nuvem de pontos. Isso ocorre por 2 razões. Primeiro, as cores não são entregues ao algoritmo como informação de treinamento. Logo, para o algoritmo, mesmo que as cores fossem removidas ou aleatoriamente modificadas, não haveria impacto algum. O segundo motivo é que o processo de compressão de cores não modifica a posição dos pontos.

Apesar de ser intrínseco que a compressão de cores influenciará apenas as cores, nesse caso não é trivial. A informação dos pontos da nuvem é utilizada durante o processo de quantização vetorial em várias etapas, mas nenhuma delas modifica as coordenadas dos pontos originais. Portanto, para o caso de compressão de cores, os modelos possuem a exata mesma acurácia mencionada anteriormente.

Classificação pós compressão de pontos

A quantização vetorial dos pontos modifica as coordenadas ao reconstruir a nuvem. Ou seja, o formato da nuvem muda e isso terá impacto direto nos algoritmos de classificação. Sete modelos diferentes foram exportados para o formato *pth* após o treinamento. Nesse experimento, o objetivo é avaliar a acurácia desses modelos em nuvens que passaram pelo processo de compressão.

Primeiro a nuvem é comprimida com o algoritmo de quantização vetorial para um determinado K e $tamG$. Em seguida, com o dicionário as nuvens são reconstruídas e salvas formando um novo banco de dados. Nele, só existem nuvens que passaram pela compressão. Em seguida esse conjunto de nuvens é entregue aos algoritmos de classificação. Os valores de K e $tamG$ vão de, por exemplo K=64 e $tamG$=128, na qual a média de *SSIM* é cerca de 100% até K = 8 e $tamG$ = 8 que possui média acima de 90%.

A tabela 8 ilustra a característica dos 7 modelos que irão avaliar a compressão das nuvens.

Tabela 8 - Sete melhores modelos salvos e suas características

Modelo	Filtros de Convolução	Acurácia do Modelo	Número de classes	Banco de Treinamento
A	32,32,32	90%	6	3155
B	32,32,32	89%	6	3155
C	32,64,32	88%	6	631
D	32,64,32	86%	6	3155
E	32,64,128,64,32	84%	6	1893
F	32,64,32	81%	26	1893
G	32,32,32	80%	26	3155

Fonte: elaborado pelo próprio autor

Há uma diferença entre os 10 principais modelos mencionados na tabela 3 e os modelos da tabela 8. Os 10 principais modelos são obtidos na fase de experimentação na qual apenas os dados dos resultados de treinamento foram registrados. Já nesta etapa atual os modelos foram baixados e armazenados após o treinamento. Os melhores modelos da tabela 3 foram utilizados para gerar esses modelos da tabela 8.

Ademais, existem variações no percentual de acurácia por serem resultados de treinamentos diferentes. Por exemplo, o mesmo modelo pode gerar resultados ligeiramente diferentes como é o caso do modelo A e B. Dado o contexto da acurácia dos modelos, a tabela 9 sintetiza experimento desta seção.

Tabela 9 - Impacto da compressão das coordenadas dos pontos na classificação das nuvens

K	tamG	Média SSIM	TQH	A 90%	B 89%	C 88%	D 86%	E 84%	F 81%	G 80%
64	128	100%	99,994%	98%	96%	95%	91%	90%	93%	92%
32	32	99,998%	99,95%	96%	95%	94%	90%	90%	90%	89%
8	32	98,26%	99,64%	95%	93%	92%	89%	88%	86%	86%
8	8	91,24%	99,77%	77%	79%	79%	79%	74%	58%	61%
4	8	83,21%	99,84%	68%	67%	69%	65%	66%	45%	51%

Fonte: elaborado pelo próprio autor

Nele foi mensurado o impacto da compressão das coordenadas dos pontos na etapa de classificação das nuvens. O banco de dados original é o *dataset* da *Sydney Urban Objects*. Cada linha da tabela 9 representa um banco de nuvens reconstruídas após a quantização vetorial. Na primeira linha está a melhor das condições possíveis, *K* e *tamG* são valores altos que fornecem uma média de *SSIM* igual a 100%. Ou seja, na média, as nuvens são perfeitamente reconstruídas. Todos os 7 modelos possuem precisões altas, acima de 90% neste caso.

As duas linhas seguintes possuem média de *SSIM* igual a 99.998% e 98.26% respectivamente. A acurácia dos modelos em média cai 1.5% da primeira para a segunda linha e 2% da segunda para a terceira. Entretanto, continuam mantendo valores próximos a 90% de acurácia. Escolhendo *K* e *tamG* iguais a 8 o cenário muda significativamente. Com a média de *SSIM* igual a 91.24% todos os modelos caem abaixo de 80% de exatidão. Para *K* igual a 4 e *tamG* igual a 8 a média do *SSIM* é ainda mais baixo com 83.21%.

Devido a isso, todos os 7 modelos possuem neste caso acurácia abaixo de 70%. A utilização de valores mais baixos de *K* e *tamG* se justifica ao buscar um processamento mais rápido ou taxas de compressão mais altas. Contudo, se o objetivo é classificar as nuvens após a reconstrução essa é uma escolha mais sensível. Com a média de *SSIM* igual a 83.21% a grande maior parte das nuvens será idêntica a nuvem original. Apesar disso, o ruído e perdas geradas pela quantização vetorial é suficiente para impactar a acurácia dos modelos de classificação.

5.4 Análise Comparativa Entre Sistemas

Vários trabalhos recentes abordam a compressão ou classificação de nuvem de pontos. Contudo, estudos que avaliam as duas funções em conjunto não são tão comuns. Logo, existe uma dificuldade em comparar os resultados obtidos no trabalho com o estado da arte. Está seção busca, de forma simplificada, analisar alguns dos trabalhos mais relevantes em comparação com o presente estudo desenvolvido.

Remoção de ruído

Em Lu (2025) um algoritmo de aprendizado profundo para remoção de ruído de nuvens de pontos é introduzido. O DEN4 utiliza um módulo de separação de ruído multi nível que é capaz de distinguir entre sinal e ruído. A média do *SSIM* obtido após a aplicação do DEN4 é 83.33%. Esse e demais resultados do algoritmo confirmam que ele supera os principais métodos tradicionais de remoção de ruído. Dito isso, apesar desse trabalho tratar um cenário diferente, ele sintetiza valores competitivos do *SSIM* no estado da arte vigente.

Da figura 17 e 19 é possível concluir que: tomando $K \geq 16$ é possível obter sempre uma média de *SSIM* superior a 83.33%. Isso significa uma significativa liberdade na escolha do tamanho de grupo o que implica obter maiores níveis de compressão enquanto se mantém a qualidade da reconstrução.

Direção autônoma

O trabalho de Abbasi (2023) revisa vários estudos sobre compressão, processamento e aprendizado de máquina para nuvens de pontos no contexto de direção autônoma. As acurácias de vários modelos são comparadas assim como ocorre em Sarker (2024). O *PointNet++* se mantem entre os melhores modelos *pointwise MLPMethods*. O banco de dados que possui mais resultados de diferentes modelos é o *ModelNet40*. Ele possui 40 classes de nuvens de pontos criadas digitalmente por modelos *CAD 3D*. A média de acurácia dos modelos catalogados em Abbasi (2023) é 90.63% entre os métodos ponto a ponto *MLP*.

No geral, modelos que possuem acurácia próxima ou superior a 90% são considerados relevantes no estado da arte. Apesar dos testes com os 7 diferentes modelos serem executados em um banco de dados diferente, todos eles alcançam acurácia acima de 90%. Além disso, o banco de dados da *SydneyUrbanObjects* possui como objetivo fornecer nuvens de capturas reais, enquanto o *ModelNet40* oferece nuvens ideias criadas digitalmente. Logo, os resultados obtidos com os modelos possuem desempenho adequado.

Fresagem robótica

Direção autônoma, e controle robótico são dois possíveis cenários em que o presente estudo possui aplicação direta. Contudo, direção autônoma possui um contexto crítico de risco e nele os resultados necessariamente precisam ter acurácia significativamente alta e baixo tempo de processamento. Em contrapartida, o controle de usinagem robótica possui um cenário mais flexível.

Gao (2021) aborda o controle de usinagem a partir de um braço robótico que utiliza nuvem de pontos. Nesse contexto a acurácia é o parâmetro principal enquanto o tempo de processamento e execução é flexível. A estrutura desenvolvida por Gao (2021) executa usinagem de peças de grande porte. Além disso, as peças também precisam passar pela geração de sua versão digital como nuvem de pontos.

Logo, as nuvens obtidas nesse processo possuem um grande volume. Por sua vez, o algoritmo de compressão desenvolvido poderia ser aplicado sem problemas ao sistema em questão. Já a classificação é uma etapa que não necessariamente faria parte do processo.

Previsão de poses humanas

O estudo de previsão de poses humanas elaborado por Zhou (2022) primeiro transforma imagens de profundidade em nuvens de pontos. Em seguida um algoritmo de segmentação é aplicado separar o corpo humano das demais partes da nuvem. Uma fase de amostragem ocorre para simplificar e remover ruídos e por fim um modelo de regressão é capaz de identificar as articulações e partes do corpo.

Mais uma vez, a compressão de nuvens de pontos é perfeitamente aplicável nesse contexto. Já a classificação precisaria de algumas adaptações para receber partes isoladas do corpo humano e só então classificar qual parte do corpo a nuvem representa.

6 CONCLUSÃO

Nuvens de pontos capturam valiosas informações sobre objetos e cenas. Elas são utilizadas em múltiplas áreas do conhecimento como engenharia, saúde e educação. Entretanto, o formato não linear e o alto volume de dados são problemas intrínsecos as nuvens de pontos. Esses problemas interferem na aplicabilidade e impacto desse tipo de mídia. Além disso, em cenários reais de captura de nuvens de pontos, como em um contexto urbano, as nuvens acabam adquirindo ruídos e falhas.

Dito isso, o presente trabalho buscou elaborar e avaliar um sistema de compressão e classificação de nuvens de pontos obtidas por sensores *LIDAR*. As duas etapas foram avaliadas isoladamente e em conjunto. O algoritmo de compressão foi desenvolvido utilizando quantização vetorial. Já a classificação das nuvens ocorre a partir da criação de um novo modelo com camadas de convolução baseadas nos modelos *PointNet++* e *PPFNet*.

A quantização vetorial aliada a codificação huffman demonstrou superioridade absoluta ao ser comparada com a quantização vetorial aplicada isoladamente. A união das duas técnicas implica numa maior taxa de compressão enquanto se mantem a qualidade da reconstrução. O impacto da quantização na classificação de nuvens de pontos foi medido. Para isso sete diferentes modelos foram criados. Todos os 7 modelos possuem acurácia acima de 90%.

Para manter essa exatidão após comprimir as nuvens, o *SSIM* do banco de dados comprimido precisa estar acima de 98%. Com isso, o sistema é capaz de manter acurácias acima de 90% enquanto possui taxas de compressão em torno de 99.64%. Por fim, uma análise crítica foi construída sobre o sistema. Ele é capaz de se adaptar a diferentes cenários, principalmente quanto a necessidade de compactação de nuvens de pontos. A classificação alcança percentuais de acurácia competitivos com o estado da arte e também possui capacidade relevante de adaptação.

REFERÊNCIAS

AWAN, Ali Abid. **A comprehensive introduction to graph neural networks (GNNs)**. [*S, l.*], 2024. Site. Disponível em: https://www.datacamp.com/pt/tutorial/comprehensive-introduction-graph-neural-networks-gnns-tutorial. Acesso em: 07 abr. 2025.

AGAPAKI, Eva; BRILAKIS, Ioannis. CLOI-NET: Class segmentation of industrial facilities' point cloud datasets. **Advanced Engineering Informatics.** Cambridge, v. 45, 2020. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S1474034620300902. Acesso em: 20 de jun 2025.

ABOUT Sketchfab**.** New York, 2025. Site. Disponível em: https://sketchfab.com/about. Acesso em: 02 mar. 2025.

ABBASI, Rashid; BASHIR, Ali Kashif; ALYAMANI, Hasan; AMIN, Farhan; DOH, Jaehyeok; CHEN, Jianwen. Lidar Point Cloud compression, processing and learning for Autonomous Driving. **IEEE Transactions on Intelligent Transportation Systems**, v. 24, n. 1, p. 962-979, Jan. 2023. DOI: 10.1109/TITS.2022.3167957. Disponível em: https://e-space.mmu.ac.uk/631072/1/Lidar%20point%20Cloud%20Compression.pdf. Acesso em: 19 jun. 2025.

BRUDER, G.; STEINICKE, F.; NÜCHTER, A. Poster: Immersive point Cloud Virtual Environments. *In*: 2014 IEEE Symposium on 3D User Interfaces (3DUI) (IEEE), p. 161–162, 2014, Minneapolis. **Anais**[...] Minneapolis, Minnesota. DOI:10.1109/3DUI.2014.6798870. Disponível em: https://ieeexplore.ieee.org/document/6798870. Acesso em: 16 jul. 2025.

BRACCI, F.; DRAUSCHKE, M.; KÜHNE, S.; MÁRTON, Z. Challenges in fusion of heterogeneous point clouds. **The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences.** v. 42, n. 2, p. 155–162, Jun. 2018. Disponível em: https://isprs-archives.copernicus.org/articles/XLII-2/155/2018/isprs-archives-XLII-2-155-2018.pdf . Acesso em: 15 jul. 2025.

CAI, W.; LIU, D.; NING, X.; WANG, C.; XIE, G. Voxel-based three- view hybrid parallel network for 3d object classification. **Displays**. v. 69, p. 102076, Set. 2021. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0141938221000834. Acesso em: 27 jul. 2025.

COMBA, Lorenzo; BIGLIA, Alessandro; AIMONINO, Davide Ricauda; GAY, Paolo. Unsupervised detection of vineyards by 3D point-cloud UAV photogrammetry for precision agriculture. **Computers and Electronics in Agriculture**. v. 155, p. 84-95, Dez. 2018. Disponível em: https://iris.unito.it/retrieve/e27ce42d-810c-2581-e053-d805fe0acbaa/IRIS.pdf . Acesso em: 06 jun. 2025.

CATALA-ROMAN, P.; NAVARRO, E. A.; SEGURA-GARCIA, J.; GARCIA-PINEDA, M. Harnessing Digital Twins for Agriculture 5.0: A Comparative Analysis of 3D Point Cloud Tools. **Applied Sciences**. v. 14, n. 5, p. 1709, Fev. 2024. Disponível em: https://www.mdpi.com/2076-3417/14/5/1709/pdf . Acesso em: 09 mai. 2025.

CHANG, A. X. *et al*. ShapeNet: An Information-Rich 3D Model Repository. **Arxiv**. v. abs/1512.03012, Dez. 2015. Disponível em: https://arxiv.org/pdf/1512.03012. Acesso em: 17 jun. 2025.

CHEN, S.; LIU, B.; FENG, C.; VALLESPI-GONZALEZ, C.; WELLINGTON, C. 3D point cloud processing and learning for autonomous driving. **Arxiv**. v. abs/2003.00601, Mar. 2020 .Disponível em: https://arxiv.org/pdf/2003.00601. Acesso em: 19 jun. 2025.

CUI, Y.; CHEN, R.; CHU, W.; CHEN, L.; TIAN, D.; LI, Y.; CAO, D. Deep learning for image and point cloud fusion in autonomous driving: A review. **IEEE Transactions on Intelligent Transportation Systems**. V.23, n. 2, p. 722-739, Fev. 2021. Disponível em: https://ieeexplore.ieee.org/abstract/document/9380166/. Acesso em: 8 jul. 2025.

CHEN, S.; GARCIA, R.; SCHMID, C.; LAPTEV, I. Polarnet: 3d point clouds for language-guided robotic manipulation. **Arxiv**. v. abs/2309.15596, Set. 2023. Disponível em: https://arxiv.org/pdf/2309.15596. Acesso em: 14 jun. 2025.

DIAB, Ahmed; KASHEF, Rasha; SHAKER, Ahmed. Deep Learning for LiDAR Point Cloud Classification in Remote Sensing. **Sensors**. v. 22, n. 20, p. 7868, Out. 2022. Disponível em: https://www.mdpi.com/1424-8220/22/20/7868/pdf. Acesso em: 09 ago. 2025.

DE DEUGE, M.; QUADROS, A.; HUNG, C.; DOUILLARD, B. Unsuper-vised feature learning for classification of outdoor 3d scans. *In*: Australasian Conference on Robitics and Automation, 2-4; Australian Robotics & Automation Association, 2013, Sydeny. **Anais**[...]. Sydney: University of New South Wales. Disponível em: https://www.araa.asn.au/acra/acra2013/papers/pap133s1-file1.pdf. Acesso em: 24 jun. 2025. Perth

DUAN, H.; WANG, P.; HUANG, Y.; XU, G.; WEI, W.; SHEN, X. Robotics dexterous grasping: The methods based on point cloud and deep learning**. Frontiers in Neurorobotics**. v.15, p.658280, Jun 2021. Disponível em: https://public-pages-files-2025.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2021.658280/pdf . Acesso em: 6 jul. 2025.

DENG, Haowen; BIRDAL, Tolga; ILIC, Slobodan. PPFNet: Global Context Aware Local Features for Robust 3D Point Matching. **IEEE/CVF Conference on Computer Vision and Pattern Recognition.** v. 1, p. 195-205, Mar. 2018. Disponível em: https://arxiv.org/pdf/1802.02669. Acesso em: 16 mai. 2025.

FONSECA, C. S.; FERREIRA, F. A. B. S.; MADEIRO, F. Vector quantization codebook design based on fish school search algorithm. **Applied soft computing**. v. 73, p. 958-968, Dez. 2018. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S1568494618305428. Acesso em: 19 abr. 2025.

FERRARI, S; FROSIO, I.; PIURI, V.; BORGHESE, N. A. Enhanced vector quantization for data reduction and filtering. *In:* Proceedings. 2nd International Symposium on 3D Data Processing; Visualization and Transmission**,** v. 3DPVT, p. 470-477, 2004, Thessaloniki. **Anais**[...]. DOI: 10.1109/TDPVT.2004.1335275. Disponível em: https://ieeexplore.ieee.org/document/1335275. Acesso em: 16 jul. 2025.

FUKUDA, T.; ZHU, Y.; YABUKI, N. (2018). Point Cloud Stream on Spatial Mixed Reality – toward Telepresence in Architectural Field. *In*: Computing for a Better Tomorrow; Proceedings of the 36th eCAADe Conference, v. 2, p. 727–734, 2018; **Anais**[...]**.** Lodz: Lodz University of Technology. Disponível em: https://papers.cumincad.org/data/works/att/ecaade2018_145.pdf . Acesso em: 17 jul. 2025.

FLORES, T.; MEDEIROS, M; SILVA, M.; COSTA, D. G.; SILVA, I. Enhanced Vector Quantization for Embedded Machine Learning: A Post-Training Approach With Incremental Clustering. **IEEE Access.** v. 13, p. 17440-17456, Jan. 2025. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10849357. Acesso em: 16 jul. 2025.

GAO, W; FAN, S.; LI, G.; LIN, W. A thorough benchmark and a new model for light field saliency detection. **IEEE Transactions on Pattern Analysis and Machine Intelligence**. v. 45, n. 7, Jan. 2023. Disponível em: https://ieeexplore.ieee.org/document/10012539/ . Acesso em: 27 fev. 2025.

GAO, Wei; GAO, Wenxu; UM, Xingming; PENG, Changhao; LI, Ge. Overview and Comparison of AVS Point Cloud Compression Standard. **APSIPA Transactions on Signal and Information Processing**. v. 14, n. 2, e103, Abr. 2025. Disponível em: https://www.nowpublishers.com/article/OpenAccessDownload/SIP-20240066 . Acesso em: 16 jul. 2025.

GAO, Yongzhuo; GAO, Haibo; BAI, Kunpeng; LI, Mingyang; DONG, Wei. 2021. A Robotic Milling System Based on 3D Point Cloud. **Machines**. v. 9, n. 12, p. 355, Dez. 2021. Disponível em: https://www.mdpi.com/2075-1702/9/12/355/pdf. Acesso em: 18 jul. 2025.

INTRODUCING Zivid 3. **ZIVID**, Norway, [202-?]. Site. Disponível em: https://www.zivid.com . Acesso em: 02 jul. 2025.

KUMAR, S.; SUHAIB, M.; ASJAD, M. Narrowing the barriers to Industry 4.0 practices through PCA-FuzzyAHP-K means. **Journal of Advances in Management Research.** v. 18, n. 2, p. 200-226, Ago. 2020. New Delhi, Índia. Disponível em: https://www.emerald.com/jamr/article-abstract/18/2/200/435189/Narrowing-the-barriers-to-Industry-4-0-practices?redirectedFrom=PDF. Acesso em: 30 jul. 2025.

LIU W; LAI, B.; WANG, C.; BIAN, X.; YANG, W.; XIA, Y.; LIN, X.; LAI, S.; WENG, D.; LI, J. Learning to match 2d images and 3d lidar point clouds for outdoor augmented reality. *In*: 2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), p. 654–5, Mar. 2020. **Anais**[...]. Disponível em: https://www.cs.nthu.edu.tw/~lai/pdf/publications/2020/Learning_to_Match_2D_Images_and_3D_LiDAR_Point_Clouds_for_Outdoor_Augmented_Reality.pdf. Acesso em: 26 jul. 2025.

LI, Y.; MA, L.; ZHONG, Z.; LIU, F.; CHAPMAN, M. A.; CAO, D. Deep Learning for LiDAR Point Clouds in Autonomous Driving: A Review. *In*: IEEE Transactions on Neural Networks and Learning Systems, v. 32, n. 8, p. 3412-3432, Ago. 2021. DOI: 10.1109/TNNLS.2020.3015992. **Anais**[...]. Disponível em: https://ieeexplore.ieee.org/abstract/document/9173706. Acesso em: 14 jul. 2025.

LIN, T. Y.; JUANG, J. G. Application of 3D point cloud map and image identification to mobile robot navigation. **Measurement and Control**. v. 56, n. 5-6, p. 911-927, Jun. 2023. Disponível em: https://journals.sagepub.com/doi/epub/10.1177/00202940221136242 . Acesso em: 19 jun. 2025.

LI, R.; LI, X.; HENG, P. A.; FU, C. W. (2020). Pointaugment: an auto-augmentation framework for point cloud classification. *In*: Proceedings of the IEEE/CVF conference; Conference on Computer Vision and Pattern Recognition (CVPR), p. 6378-6387, 2020. **Anais**[...]. Seattle, 2020. Disponível em: http://openaccess.thecvf.com/content_CVPR_2020/papers/Li_PointAugment_An_Auto-Augmentation_Framework_for_Point_Cloud_Classification_CVPR_2020_paper.pdf. Acesso em: 17 jun. 2025.

LIU, H.; JIA, J.; GONG, N. Z. Pointguard: Provably robust 3d point cloud classification**.** *In*: Proceedings of the IEEE/CVF conference; Conference on Computer Vision and Pattern Recognition (CVPR), p. 6186-6195, 2021. **Anais**[...].Disponível em: http://openaccess.thecvf.com/content/CVPR2021/papers/Liu_PointGuard_Provably_Robust_3D_Point_Cloud_Classification_CVPR_2021_paper.pdf. Acesso em: 5 jun. 2025.

LU, X.; YE, Z.; FU, L.; WANG, H.; WANG, K.; DOU, Y.; XIE, D.; ZHAO, X. LiDAR point cloud denoising for individual tree extraction based on the Noise4Denoise. **Frontiers in Plant Science**. v. 15, Jan. 2025. DOI: 10.3389/fpls.2024.1490660. Disponível em: https://public-pages-files-2025.frontiersin.org/journals/plant-science/articles/10.3389/fpls.2024.1490660/pdf . Acesso em: 14 jul. 2025.

O'SHEA, K.; NASH, R. An Introduction to Convolutional Neural Networks. **Arxiv**. v. abs/1511.08458, Dez. 2015. Disponível em: https://arxiv.org/pdf/1511.08458. Acesso em: 08 mai. 2025.

QUACH, M.; PANG, J.; TIAN, D.; VALENZISE, G.; DUFAUX, F. Survey on Deep Learning-Based Point Cloud Compression. **Frontiers in Signal Processing.** v. 2, Fev. 2022. DOI:10.3389/frsip.2022.846972. Disponível em: https://public-pages-files-2025.frontiersin.org/journals/signal-processing/articles/10.3389/frsip.2022.846972/pdf . Acesso em: 19 jul. 2025.

QI, C. R.; YI, L.; SU, H.; GUIBAS, L. J. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. **Arxiv**. v. abs/1706.02413, Jun. 2017. Disponível em: https://arxiv.org/pdf/1706.02413. Acesso em: 19 ago. 2025

QI, C. R.; SU, H.; MO, K.; GUIBAS, L. J. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. p. 77-85, Abr. 2017. Disponível em: https://arxiv.org/pdf/1612.00593. Acesso em: 25 mai. 2025.

REN, J.; PAN, L.; LIU, Z. (2022, June). Benchmarking and analyzing point cloud classification under corruptions. *In*: International Conference on Machine Learning, p. 18559-18575, e39, 2022. **Anais**[...]. Disponível em: https://proceedings.mlr.press/v162/ren22c/ren22c.pdf. Acesso em: 4 jun. 2025.

STRYKER, C.; LEE, F; BERGMANN, D.; SCAPICCHIO, M. **O que é uma rede neural?** [*S, l.*], 2025a. Site. Disponível em: https://www.ibm.com/br-pt/think/topics/neural-networks . Acesso em: 02 mar. 2025.

STRYKER, C.; LEE, F; BERGMANN, D.; SCAPICCHIO, M. **O que são redes neurais convolucionais?** [*S, l.*], 2025b. Site. Disponível em: https://www.ibm.com/br-pt/think/topics/convolutional-neural-networks. Acesso em: 07 abr. 2025.

STRYKER, C.; LEE, F; BERGMANN, D.; SCAPICCHIO, M. **What is a GNN (Graph Neural Network)?** [*S, l.*], 2025c. Site. Disponível em: https://www.ibm.com/think/topics/graph-neural-network. Acesso em: 03 abr. 2025.

SARKER, S.; SARKER, P.; STONE, G.; GORMAN, R.; TAVAKKOLI, A.; BEBIS, G.; SATTARVAND, J. A comprehensive overview of deep learning techniques for 3D point cloud classification and semantic segmentation. **Machine Vision and Applications.** v. 35, n. 67, Mai. 2024. Disponível em: https://arxiv.org/pdf/2405.11903 . Acesso em: 16 jul. 2025.

SANCHEZ-LENGELING, *et al*. **A Gentle Introduction to Graph Neural Networks**. Distill, Set. 2021. Disponível em: https://distill.pub/2021/gnn-intro. Acesso em: 09 abr. 2025.

Sydney University. **Sydney Urban Objects Dataset, 2013.** Sydney: SU. Disponível em: https://www.acfr.usyd.edu.au/papers/SydneyUrbanObjectsDataset.shtml. Acesso em: 02 jul. 2025.

SANDRI, G.; DE QUEIROZ, R.; CHOU, P. A. Compression Of Plenoptic Point Clouds Using The Region-Adaptive Hierarchical Transform. **IEEE Trans Image Process**. p. 1153-1157, Set. 2018. DOI: 10.1109/TIP.2018.2877486. Disponível em: https://packet.media/wp-content/uploads/2018/06/icip2018_sandri_v2.pdf. Acesso em: 15 jul. 2025.

TOMMASI, C.; ACHILLE, C.; FASSI, F. From Point Cloud to Bim: A Modelling Challenge in the Cultural Heritage Field. *In*: ISPRS – International Archives of the Photogrammetry; Remote Sensing and Spatial Information Sciences (IEEE), v. XLI-B5, p. 429-436, 2016. DOI:10.5194/ isprsarchives-XLI-B5-429-2016. **Anais**[...]. Disponível em: https://isprs-archives.copernicus.org/articles/XLI-B5/429/2016/isprs-archives-XLI-B5-429-2016.pdf. Acesso em: 14 jul. 2025.

UNIVERSIDADE DE SÃO PAULO, **USP**: Emerging image modalities representation and compression**.** São Paulo: USP**,** 2017**.** Site. Disponível em: http://uspaulopc.di.ubi.pt. Acesso em: 10 jun. 2025.

WANG, Q.; KIM, M. K. Applications of 3D point cloud data in the construction industry: A fifteen-year review from 2004 to 2018. **Advanced Engineering Informatics**. v. 39, p. 306-319, Jan. 2019. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S1474034618304683?via%3Dihub. Acesso em: 10 jun. 2025.

WEBER, M.; WILD, D.; KLEESIEK, J.; EGGER, J; GSAXNER, C. Deep Learning-based Point Cloud Registration for Augmented Reality-guided Surgery. *In*: 2024 IEEE International Symposium on Biomedical Imaging (ISBI). p. 1-5, 2024, Athens. **Anais**[...]. Disponível em: https://arxiv.org/pdf/2405.03314. Acesso em: 07 jul. 2025.

WU, Z.; SONG, S.; KHOSLA, A.; YU, F.; ZHANG, L.; TANG X.; XIAO. J. 3D ShapeNets: A Deep Representation for Volumetric Shapes. *In*: Proceedings of 28th IEEE Conference on Computer Vision and Pattern Recognition. v. 1, 2015, Boston. **Anais**[...]. Disponível em: http://3dvision.princeton.edu/projects/2014/3DShapeNets/paper.pdf. Acesso em: 16 jun. 2025.

WHAT is LIDAR? [*S, l.*], 16 jun. 2024. Site. Disponível em: https://oceanservice.noaa.gov/facts/lidar.html. Acesso em: 12 jul. 2025.

WANG, G.; LI, W.; JIANG, C.; ZHU, D.; LI, Z.; XU, W.; ZHAO, H.; DING, H. Trajectory planning and optimization for robotic machining based on measured point cloud. **IEEE transactions on robotics**. v. 38, n. 3, p. 1621-1637, Dez. 2021. Disponível em: https://ieeexplore.ieee.org/abstract/document/9642395/. Acesso em: 16 jun. 2025.

XIE, Y.; TIAN, J.; ZHU, X. X. Linking Points With Labels in 3D: A Review of Point Cloud Semantic Segmentation. **IEEE Geoscience and Remote Sensing Magazine**. v. 8, p. 38-59, Mar. 2020. Disponível em : https://ieeexplore.ieee.org/document/9028090. Acesso em: 25 jun. 2025.

XU, Y.; ZHANG, K.; HE, L.; JIANG, Z.; ZHU, W. Introduction to point cloud compression. **ZTE Communications**. v. 16, n. 3, 2018. Disponível em: http://kns.cnki.net/kcms/detail/34.1294.TN.20180824.1103.002.html . Acesso em: 21 jul. 2025.

YUE, X.; WU, B.; SESHIA, S. A.; KEUTZER, K.; SANGIOVANNI-VINCENTELLI, A. L. A LiDAR Point Cloud Generator: from a Virtual World to Autonomous Driving. *In*: Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval, p. 458–464, 2018. DOI:10.1145/3206025.3206080. **Anais**[...]. New York, NY: Association for Computing Machinery. Disponível em: https://arxiv.org/pdf/1804.00103. Acesso em: 18 jul. 2025.

YANG, D.; GAO, W. PointCHD: A Point Cloud Benchmark for Con-genital Heart Disease Classification and Segmentation. **IEEE Journal of Biomedical and Health Informatics**. v. 29, n. 4, p. 2683-2694, Abr. 2024. Disponível em: https://ieeexplore.ieee.org/document/10748410. Acesso em: 19 jul. 2025.

YANG, Z.; CHEN, L.; SUN, Y.; LI, H. Visual point cloud forecasting enables scalable autonomous driving. *In*: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. p. 14673-14684, 2024. **Anais**[...]. Disponível em: https://arxiv.org/pdf/2312.17655 . Acesso em: 19 jul. 2025.

ZHOU *et al***.** LiDAR-PTQ: Post-Training Quantization for Point Cloud 3D Object Detection. 2024. **ArXiv**. v. abs/2401.15865, Jan. 2024. Disponível em: https://arxiv.org/pdf/2401.15865. Acesso em: 06 ago. 2025.

ZHOU, P.; PENG, R.; XU, M.; WU, V.; NAVARRO-ALARCON, D. Path planning with automatic seam extraction over point cloud models for robotic arc welding. **IEEE robotics and automation letters**. v. 6, n. 3, p. 5002-5009, Abr. 2021. Disponível em: https://ieeexplore.ieee.org/abstract/document/9394722/. Acesso em: 14 jun. 2025.

ZHANG, H.; WANG, C.; TIAN, S.; LU, B.; ZHANG, L.; NING, X.; BAI, X. Deep learning-based 3D point cloud classification: A systematic survey and outlook. **Displays**. v. *79*, p. 102456, Nov. 2023. Disponível em: https://arxiv.org/pdf/2311.02608. Acesso em: 12 jul. 2025.

ZHANG, J.; CHEN, L.; OUYANG, B.; LIU, B.; ZHU, J.; CHEN, Y.; MENG, Y.; WU, D. (2021). Pointcutmix: Regularization strategy for point cloud classification. **Neurocomputing**. v. *505*, p. 58-67, Fev. 2021. Disponível em: https://arxiv.org/pdf/2101.01461. Acesso em: 21 jun. 2025.

ZHANG, M.; YOU, H.; KADAM, P.; LIU, S.; KUO, C. C. J. Pointhop: An explainable machine learning method for point cloud classification. **IEEE Transactions on Multimedia**, v. 22, n. 7, p. 1744-1755, Dez. 2019. Disponível em: https://arxiv.org/pdf/1907.12766. Acesso em: 23 jun. 2025.

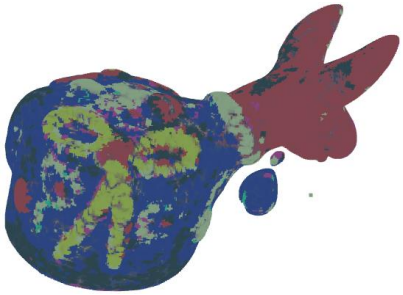
ZHOU, Y.; DONG, H.; EL SADDIK, A. Learning to Estimate 3D Human Pose From Point Cloud. **IEEE Sensors Journal.** v. 20, p. 12334-12342, Dez. 2020. Disponível em: https://arxiv.org/pdf/2212.12910. Acesso em: 16 ago. 2025.

APÊNDICE A – RESULTADOS DA QUANTIZAÇÃO VETORIAL PARA A NUVEM DE PONTOS BUMBAMEUBOI

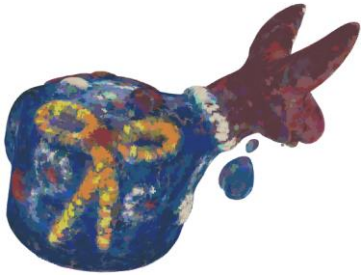
***K* (Nível da Quantização)**

Original

$K = 2$



$K = 4$



$K = 16$

$K = 32$

$K = 64$

$K = 128$

$K = 256$

$K = 512$

$K = 1024$

APÊNDICE B - RESULTADOS DA QUANTIZAÇÃO VETORIAL PARA A NUVEM DE PONTOS *GUY*

Original

$K = 2$

$K = 4$

$K = 16$

$K = 32$

$K = 64$

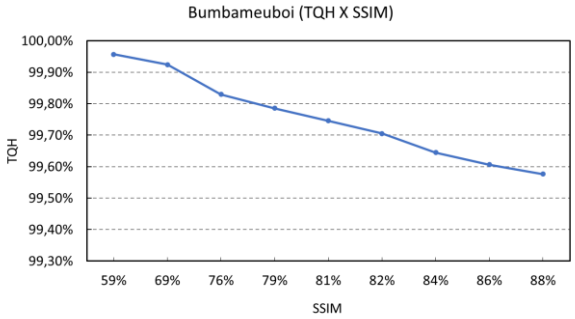
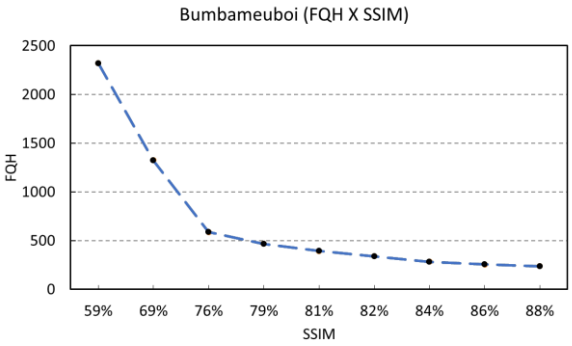
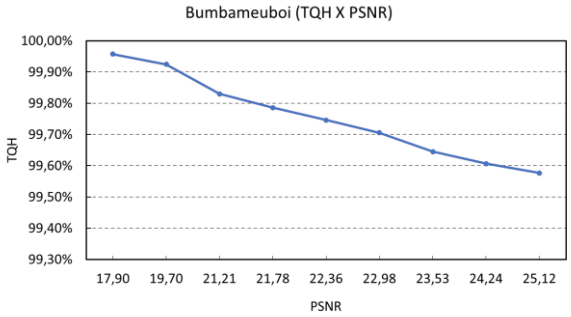
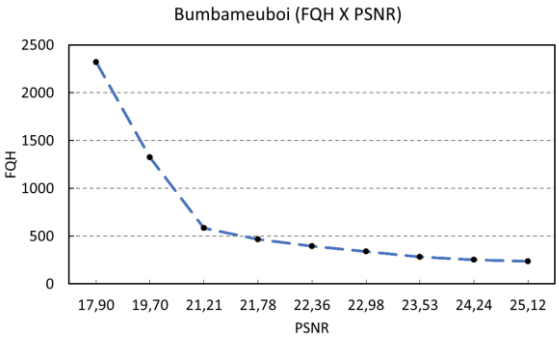
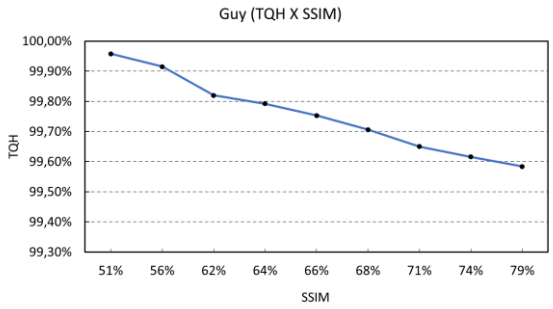
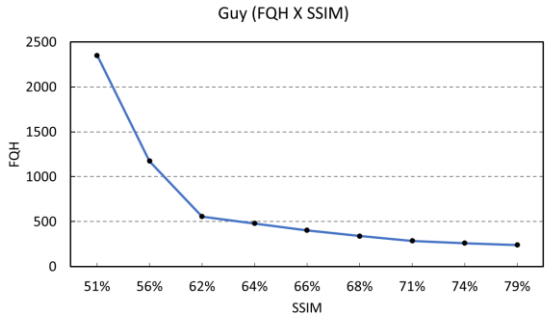
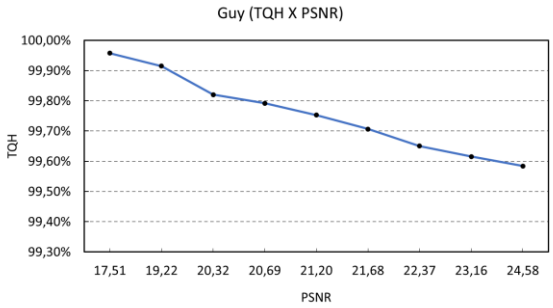
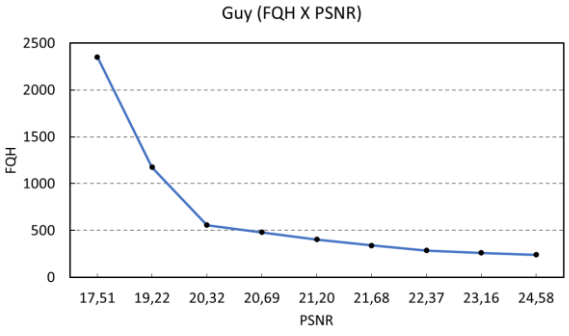
$K = 128$

$K = 256$

$K = 512$

$K = 1024$

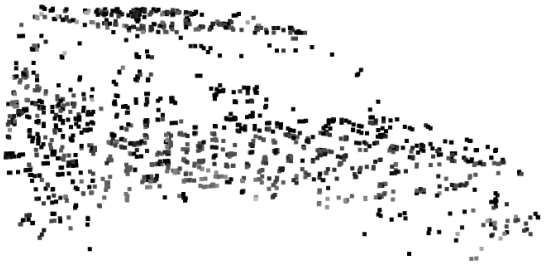
APÊNDICE C – RELAÇÃO ENTRE COMPRESSÃO E QUALIDADE

Taxa de compressão	Fator de compressão
	
	
	
	

APÊNDICE D – EFEITO DA COMPRESSÃO DE POSIÇÃO

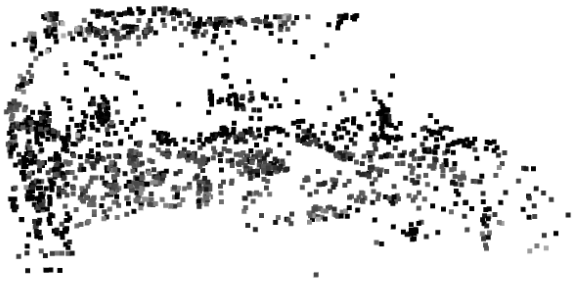
TamG(Tamanho de Grupo) *K*=32

TamG=4

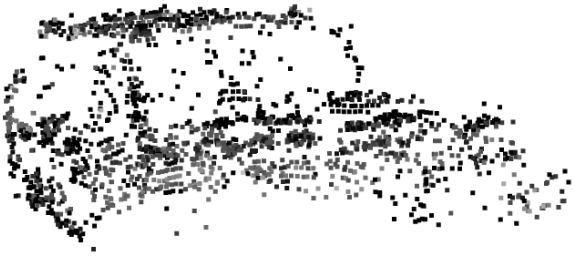


TamG=8

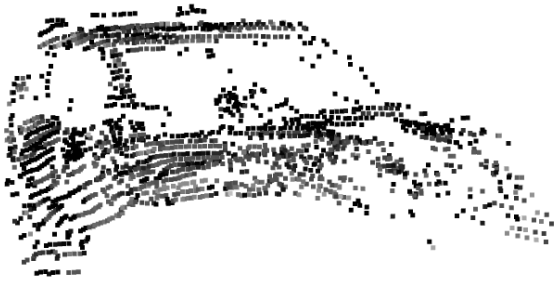
TamG=16



TamG=32



TamG=64



TamG=128