UNIVERSIDADE FEDERAL DE GOIÁS / INSTITUTO DE INFORMÁTICA

# Transferência Sim2Real aplicada à Visão Computacional

## Estudo sobre a Segmentação 3D de Point Clouds para Aplicações Autônomas

Pedro Martins Bittencourt

UFG

UNIVERSIDADE FEDERAL DE GOIÁS

UNIVERSIDADE FEDERAL DE GOIÁS (UFG)
INSTITUTO DE INFORMÁTICA (INF)

PEDRO MARTINS BITTENCOURT

# Transferência Sim2Real aplicada à Visão Computacional

Estudo sobre a Segmentação 3D de Point Clouds para Aplicações Autônomas

Goiânia
2025

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

# TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO NO REPOSITÓRIO INSTITUCIONAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio do Repositório Institucional (RI/UFG), regulamentado pela Resolução CEPEC no 1240/2014, sem ressarcimento dos direitos autorais, de acordo com a Lei no 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo dos Trabalhos de Conclusão dos Cursos de Graduação disponibilizado no RI/UFG é de responsabilidade exclusiva dos autores. Ao encaminhar(em) o produto final, o(s) autor(a)(es)(as) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

**1. Identificação do Trabalho de Conclusão de Curso de Graduação (TCCG)**

Nome(s) completo(s) do(a)(s) autor(a)(es)(as): PEDRO MARTINS BITTENCOURT

Título do trabalho: Transferência Sim2Real aplicada à Visão Computacional

Estudo sobre a Segmentação 3D de Point Clouds para Aplicações Autônomas

**2. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador) Concorda com a liberação total do documento** [X] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante: a) consulta ao(à)(s) autor(a)(es)(as) e ao(à) orientador(a); b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo do TCCG. O documento não será disponibilizado durante o período de embargo.

**Casos de embargo:**

- Solicitação de registro de patente;

- Submissão de artigo em revista científica;

- Publicação como capítulo de livro.

**Obs.: Este termo deve ser assinado no SEI pelo orientador e pelo autor.**

Documento assinado eletronicamente por **Pedro Martins Bittencourt**, **Discente**, em 18/01/2025, às 16:34, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Fernando Marques Federson**, **Professor do Magistério Superior**, em 20/01/2025, às 19:49, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5089801** e o código CRC **6CF65F3A**.

**Referência:** Processo nº 23070.001597/2025-48 SEI nº 5089801

PEDRO MARTINS BITTENCOURT

# Transferência Sim2Real aplicada à Visão Computacional

Estudo sobre a Segmentação 3D de Point Clouds para Aplicações Autônomas

Relatório final de Trabalho de Conclusão de Curso, apresentado à Universidade Federal de Goiás, como parte das exigências para a obtenção do título de Bacharel em Inteligência Artificial.
Orientador: Prof. Dr. Fernando Marques Federson

Goiânia
2025

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

BITTENCOURT, PEDRO MARTINS
Transferência Sim2Real aplicada à Visão Computacional [manuscrito] : Estudo sobre a Segmentação 3D de Point Clouds para Aplicações Autônomas / PEDRO MARTINS BITTENCOURT. - 2025.
101 f.

Orientador: Prof. Dr. Fernando Marques Federson.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Goiás, Instituto de Informática (INF), Inteligência Artificial, Goiânia, 2025.

1. inteligência artificial. 2. visão computacional. 3. veículos autônomos. I. Federson, Fernando Marques , orient. II. Título.

CDU 004

PEDRO MARTINS BITTENCOURT

**Transferência Sim2Real aplicada à Visão Computacional**

Estudo sobre a Segmentação 3D de Point Clouds para Aplicações Autônomas

Relatório final de Trabalho de Conclusão de Curso, apresentado à Universidade Federal de Goiás, como parte das exigências para a obtenção do título de Bacharel em Inteligência Artificial.

Data da Aprovação: 17 de dezembro de 2024.

Prof. Dr. Fernando Marques Federson
Orientador (INF-UFG)

Prof. Dr. Aldo André Díaz Salazar
Coordenador de TCC do BIA (INF-UFG)

Prof. Dr. Anderson da Silva Soares
Coordenador do BIA (INF-UFG)

Prof. Me. Lucas Araújo Pereira
(INF-UFG)

PEDRO MARTINS BITTENCOURT

# Transferência Sim2Real aplicada à Visão Computacional

Estudo sobre a Segmentação 3D de Point Clouds para Aplicações Autônomas


## RESUMO

Este Relatório de Conclusão de Curso tem como objetivo reunir os resultados da minha jornada para me tornar um especialista em **Visão Computacional (Veículos Autônomos)**. Uma ilustração e sua narrativa descrevem os períodos de trabalho. Os Apêndices contêm os Termos de Aceite de Entrega e os resultados obtidos durante cada período de trabalho.


Palavras-chave: inteligência artificial, modelos grandes de linguagem, geração automática de datasets.


## ABSTRACT

This Course Completion Report aims to bring together the results of my journey to become an expert in **Computer Vision (Autonomous Vehicles)**. An illustration and its narrative describe the work periods. The Appendices contain the Delivery Acceptance Terms and the results obtained during each work period.


Keywords: artificial intelligence, large language models, automatic dataset generation.

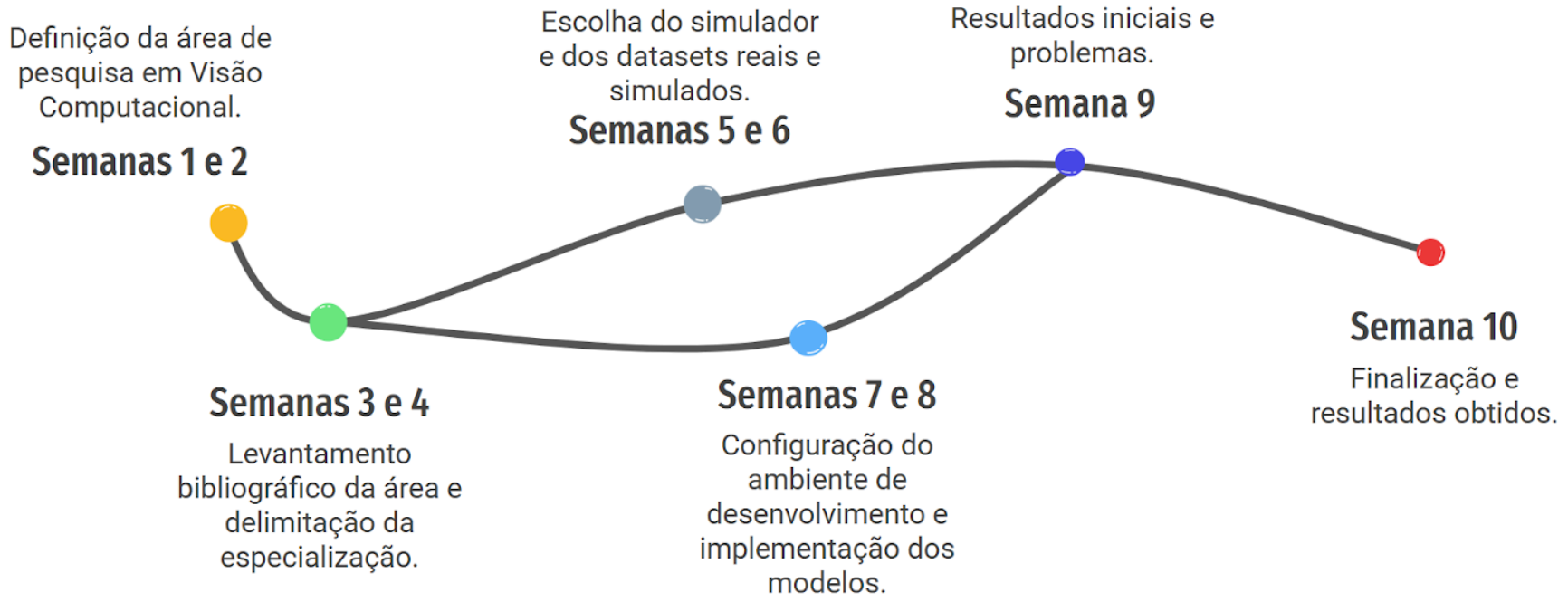
Goiânia

2025

# Minha Jornada

Pedro Martins Bittencourt

Especialista em: Visão Computacional (Veículos Autônomos)



# MINHA JORNADA

**Nome:** **Pedro Martins Bittencourt**

**Especialidade:** **Visão Computacional (Veículos Autônomos)**

## Objetivo deste documento

Durante o processo da disciplina Residência em IA[1], foram gerados diversos resultados na construção da minha especialização. A cada semana, um conjunto de resultados foi formalizado por um Termo de Aceite de Entrega e avaliado por uma banca, considerando o planejado e o realizado para o período. Este documento tem como objetivo descrever esses resultados obtidos, fazendo referência aos Termos de Aceite de Entrega e seus documentos associados.

## Minha Jornada

Minha jornada na Residência em Inteligência Artificial teve início nas **Semanas 1 e 2**, quando me dediquei a explorar e definir a macroárea de interesse para minha especialização. Durante esse período, realizei um estudo inicial de artigos, livros e outros materiais que me permitiram compreender a história e os avanços recentes da área. Dessa análise, emergiu meu interesse por Visão Computacional, uma escolha que seria a base para os trabalhos futuros. Este levantamento também me ajudou a compilar uma lista de aplicações e cenários reais em que essa tecnologia desempenha um papel central. Na segunda Semana, me aprofundei nos conceitos de visão computacional aplicada a veículos autônomos. Esse foco me levou a estudar marcos históricos e desafios atuais dessa tecnologia. Trabalhos acadêmicos como Self-Driving Cars: A Survey e Reimagining an Autonomous Vehicle foram fundamentais para expandir meu entendimento sobre sensores, algoritmos de visão computacional e arquiteturas baseadas em redes neurais profundas. Assim, compreendi como essas tecnologias suportam a percepção, o planejamento e a

[1] Dez semanas, entre setembro de 2024 e dezembro de 2024.

tomada de decisão em tempo real. Os materiais relacionados a estas duas Semanas podem ser encontrados no **Apêndice 1**.

Nas **Semanas 3 e 4**, iniciei o levantamento detalhado de tarefas específicas em veículos autônomos, como detecção e classificação de objetos, rastreamento e SLAM (Simultaneous Localization and Mapping). Esses estudos destacaram a necessidade de recursos computacionais robustos e da geração de dados sintéticos em ambientes simulados. A partir disso, passei a estudar como o gap entre simulação e realidade (Sim2Real) impacta diretamente o desempenho de modelos treinados e a importância de gerar dados sintéticos realistas e de alta qualidade em ambientes simulados. Além disso, explorei técnicas de detecção e classificação de objetos, incluindo abordagens baseadas em Deep Learning. Durante essa etapa, decidi que minha linha de pesquisa se concentraria na geração de datasets em ambientes simulados. Paralelamente, investiguei simuladores como CARLA, Gazebo e NVIDIA DRIVE Sim, avaliando suas vantagens e limitações. Essa investigação me levou a selecionar o CARLA para conduzir meus experimentos devido à sua capacidade de gerar datasets ricos e variados. Os materiais relacionados a estas Semanas podem ser encontrados no **Apêndice 2**.

Nas **Semanas 5 e 6**, me dediquei à configuração do simulador CARLA e à coleta inicial de dados. Durante o processo, enfrentei diversos desafios técnicos, como problemas de compatibilidade de drivers e dependências de bibliotecas. Superados esses obstáculos, consegui gerar um pequeno dataset inicial para validação e posteriormente ampliei os cenários de coleta de dados no simulador, incluindo variações de clima, trânsito e períodos do dia. Durante essas Semanas, percebi que a coleta de dados é um trabalho complicado e optei por utilizar o dataset KITTI-CARLA, nos mesmos moldes do SemanticKitti. Este dataset, descrito no trabalho de Deschaud (2021), combina a estrutura bem conhecida do SemanticKitti com os cenários variados e controlados gerados pelo simulador CARLA. Complementarmente, iniciei a integração de dados reais e simulados, identificando os datasets NuScenes e SemanticKitti como possíveis fontes para benchmarking e comparação. Os materiais relacionados a estas duas Semanas podem ser encontrados no **Apêndice 3**.

Nas **Semanas 7 e 8**, inicialmente fiz o levantamento dos modelos para detecção e segmentação. Entre eles, o PillarNeXt, um modelo baseado em convoluções para converter nuvens de pontos 3D em pseudo-imagens 2D e categorizar objetos. Para segmentação, analisei o MvLiDARNet, que utiliza projeções esféricas de nuvens de pontos e camadas de convolução 2D para segmentação pixel a pixel. Posteriormente, optei por seguir apenas com o MvLiDARNet, principalmente devido à complexidade do dataset NuScenes, que exigiria um processamento mais elaborado. Com essa escolha, configurei o ambiente de treinamento do modelo utilizando tanto dados simulados quanto reais, harmonizando os datasets para garantir consistência nas classes e formatos. Os materiais relacionados a estas duas Semanas podem ser encontrados no **Apêndice 4**.

Na **Semana 9**, me concentrei no treinamento e avaliação de modelos com dados simulados e reais, realizando treinamentos extensivos em cada dataset separadamente para comparações diretas. Confiante no modelo escolhido, foquei nesse processo contínuo, mas os resultados obtidos ficaram abaixo do esperado, indicando limitações no desempenho quando os dados eram utilizados isoladamente. Os materiais relacionados a esta Semana podem ser encontrados no **Apêndice 5**.

Finalmente, na **Semana 10**, conduzi uma série de experimentos combinando dados reais e simulados, explorando diferentes proporções para identificar o equilíbrio ideal. Foi nesse momento que obtive os melhores resultados. Os experimentos demonstraram que, embora dados reais apresentem maior eficácia, a inclusão gradual de dados simulados aumenta a robustez e melhora a generalização dos modelos. A combinação estratégica dos cenários resultou em melhorias consideráveis na acurácia geral. Os materiais relacionados a esta Semana podem ser encontrados no **Apêndice 6**.

Essa jornada foi marcada por desafios e aprendizados intensos. Cada etapa contribuiu para meu amadurecimento técnico e acadêmico, reforçando minha paixão pela área de visão computacional aplicada a veículos autônomos. As lições obtidas durante o processo

me capacitaram a enfrentar desafios futuros com confiança e preparação, consolidando a base para minha especialização em Inteligência Artificial.

# APÊNDICE 1

# Termo de Aceite de Entrega

## Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

## Data da Reunião ("gate") de aprovação: 18 de set. de 2024

## Participantes da Entrega [matriculados em Residência em IA]:

| |
|---|
| **Pedro Martins Bittencourt** |

## Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Nessa primeira semana o intuito foi conhecer a minha macroárea de interesse, para isso foi realizada a exploração e o estudo de artigos, livros e outros materiais. O foco principal foi em revisões, o que me permitiu ter uma noção mais ampla da história da área e de como ela se desenvolveu nos últimos anos. Também fui capaz de montar uma lista não exaustiva das aplicações e dos cenários reais abrangidos pela macroárea.

Estudo realizado: Estudo residência 180924

A macroárea de interesse escolhida foi Visão Computacional.

## Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

- Pesquisar para encontrar o estado da arte e o que há de mais recente na literatura, as novas técnicas empregadas e as maiores dificuldades atualmente.
- Fechar um pouco mais o escopo da área de interesse.

**Observação: [caso precise fazer alguma observação, de qualquer "natureza"]**

## ACEITE DA ENTREGA:

**CEDRIC LUIZ DE CARVALHO:** Go!

# Termo de Aceite de Entrega

## Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

## Data da Reunião ("gate") de aprovação: 25 de set. de 2024

## Participantes da Entrega [matriculados em Residência em IA]:

| |
|---|
| **Pedro Martins Bittencourt** |

## Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

O estudo realizado e subproduto dessa entrega está em: Estudo residência 250924

**1 - Estudo de papers e história dos veículos autônomos**
Foram explorados os principais marcos históricos e papers acadêmicos relevantes sobre a evolução dos veículos autônomos, destacando tanto a história quanto os desafios atuais da tecnologia. Alguns trabalhos fundamentais foram revisados:

- **Self-Driving Cars: A Survey** - Uma análise (brasileira) abrangente dos veículos autônomos, cobrindo sensores, algoritmos de visão computacional e desafios técnicos.
- **Reimagining an Autonomous Vehicle** - Discussão sobre as arquiteturas avançadas de inteligência artificial aplicadas em veículos autônomos, incluindo a visão computacional e o uso de redes neurais profundas

Um levantamento detalhado da evolução histórica dos veículos autônomos foi realizado, abordando os primeiros experimentos, os avanços tecnológicos e o estado atual.

**2 - Visão Computacional em Veículos Autônomos**
A visão computacional foi explorada como uma tecnologia central nos veículos autônomos, já que essa é a minha macroárea de pesquisa. Foco nas arquiteturas e técnicas mais avançadas:

- **Arquiteturas de Rede Neural Usadas**: Modelos como Convolutional Neural Networks (CNNs) para reconhecimento de padrões e redes neurais profundas (DNNs) para tomada de decisões em tempo real.
- **Sistemas de Percepção Baseados em Câmeras**: Integração de câmeras com outras tecnologias (Lidar e Radar) e aplicação de redes neurais para detecção de objetos.
- **Técnicas de Aprendizado Profundo**: Melhorias no reconhecimento de objetos em condições adversas, como baixa luminosidade e mau tempo.

## Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

**1. Estudo de Desafios Atuais em Veículos Autônomos**

- **Objetivo:** Identificar e entender os principais desafios técnicos que a indústria de veículos autônomos enfrenta atualmente, buscando alinhar as tarefas mais relevantes com o estado da arte.

**2. Levantamento e Análise de Tasks Específicas em Veículos Autônomos**

- **Objetivo:** Explorar as tasks principais envolvidas no desenvolvimento de veículos autônomos para escolher uma área de atuação específica.

**Observação: [caso precise fazer alguma observação, de qualquer "natureza"]**

## ACEITE DA ENTREGA:

**Leonardo Alves:** Go!

# APÊNDICE 2

# Termo de Aceite de Entrega

## Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

## Data da Reunião ("gate") de aprovação: 3 de out. de 2024

## Participantes da Entrega [matriculados em Residência em IA]:

| |
|---|
| **Pedro Martins Bittencourt** |

## Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

O estudo realizado e subproduto dessa entrega está em: Estudo residência 031024

1. **Levantamento e Análise de Tasks Específicas em Veículos Autônomos**
    - Percepção (Compreensão do Ambiente)
        - Detecção e Classificação de Objetos
        - Rastreamento de Objetos
    - Localização e Mapeamento
        - Localização e Mapeamento Simultâneos (SLAM)
    - **Simulação e Testes Virtuais**
        - **Geração de Dados Sintéticos**
            - Necessidade de poder computacional
2. **Levantamento de Desafios**
    - Desafios de Percepção
    - Planejamento de Trajetória
    - Sim2Real (Aprendizado a partir de Simulação vs. Dados do Mundo Real)

## Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

1. **Levantamento de Técnicas**
    - Detecção e Classificação de Objetos
    - SLAM (Simultaneous Localization and Mapping)
2. **Levantamento de simuladores**
    - **Objetivo**: Identificar as principais plataformas e ferramentas utilizadas para simulação de veículos autônomos.

**Observação: [caso precise fazer alguma observação, de qualquer "natureza"]**

## ACEITE DA ENTREGA:

**CEDRIC LUIZ DE CARVALHO:** Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

# Termo de Aceite de Entrega

## Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

## Data da Reunião ("gate") de aprovação: 9 de out. de 2024

## Participantes da Entrega [matriculados em Residência em IA]:

| |
|---|
| **Pedro Martins Bittencourt** |

## Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

O estudo realizado e subproduto dessa entrega está em: Estudo residência 09102024

1. **Levantamento de Técnicas**
    - Detecção e Classificação de Objetos
        - Detecção baseada em Deep Learning
            - Detecção Baseada em Deep Learning
            - Segmentação Semântica
    - SLAM (Simultaneous Localization and Mapping)
        - Métodos clássicos
            - Kalman Filter
            - Extended Kalman Filter
        - Métodos usando Deep Learning
            - DROID-SLAM
            - Deepvo
2. **Levantamento de simuladores**
    - Gazebo
    - NVIDIA DRIVE Sim
    - CARLA

Após o levantamento de técnicas, decidi que gostaria de trabalhar com geração de datasets em ambientes simulados para detecção e classificação de objetos. Ainda estou investigando os simuladores para definir qual será utilizado.

## Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

- Fazer uma análise mais aprofundada dos simuladores selecionados.
- Estudar o processo de configuração dos simuladores.
- Procurar recursos computacionais.

**Observação: [caso precise fazer alguma observação, de qualquer "natureza"]**

## ACEITE DA ENTREGA:

**CEDRIC LUIZ DE CARVALHO:** Go!

# APÊNDICE 3

# Termo de Aceite de Entrega

## Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

## Data da Reunião ("gate") de aprovação: 16 de out. de 2024

## Participantes da Entrega [matriculados em Residência em IA]:

| |
|---|
| **Pedro Martins Bittencourt** |

## Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

O estudo realizado e subproduto dessa entrega está em: Estudo residência 16102024

1. **Procurar recursos computacionais.**
   - Consegui acesso a uma máquina com GPU.

2. **Fazer uma análise mais aprofundada selecionados e estudar o processo de configuração dos simuladores**
   - **Gazebo**
     - Processo de configuração simples
     - Limitado número de recursos pré-configurados
     - Integração com outras bibliotecas
     - Comunidade ativa e documentação
   - **CARLA**
     - Processo de configuração mais complicado
     - Diversas integrações com outros frameworks
     - Nativamente monta um dataset mais completo
   - **NVIDIA DRIVE Sim**
     - Por ser uma plataforma fechada, a comunidade é pequena
     - Pouca documentação disponível
     - Foco em hardware especializado (AGX, OVX)

## Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

- Fazer o processo de configuração de pelo menos um simulador
- Começar a coletar dados em pelo menos um simulador

**Observação: [caso precise fazer alguma observação, de qualquer "natureza"]**

## ACEITE DA ENTREGA:

**CEDRIC LUIZ DE CARVALHO:** Go!

# Termo de Aceite de Entrega

## Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

## Data da Reunião ("gate") de aprovação: 30 de out. de 2024

## Participantes da Entrega [matriculados em Residência em IA]:

| |
|---|
| **Pedro Martins Bittencourt** |

## Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

O subproduto dessa entrega está em: Documentação residência 30102024

1. **Fazer o processo de configuração de pelo menos um simulador**
   - Realizei o processo de configuração do CARLA (Car Learning to Act)
   - Erros encontrado durante o processo:
     - Simulador não respondia (erro de drivers da NVIDIA)
     - Erro de biblioteca (ausente no Docker)
     - Erro na instalação do módulo python (Ubuntu 24.04 incompatível)
2. **Começar a coletar dados em pelo menos um simulador**
   - Coletei um pequeno dataset para validar o simulador

## Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

- Coletar mais dados em diferentes cenários
- Selecionar os modelos para comparação entre dataset real e simulado

**Observação: [caso precise fazer alguma observação, de qualquer "natureza"]**

## ACEITE DA ENTREGA:

**LEONARDO ALVES:** Em análise!

# APÊNDICE 4

# Termo de Aceite de Entrega

## Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

## Data da Reunião ("gate") de aprovação: 6 de nov. de 2024

## Participantes da Entrega [matriculados em Residência em IA]:

| |
|---|
| **Pedro Martins Bittencourt** |

## Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

O subproduto dessa entrega está em: Documentação residência 06112024

1. **Coletar mais dados em diferentes cenários**
   - Coletei mais alguns dados para testar as diferentes possibilidades do simulador
     - Período do dia
     - Clima
     - Intensidade de trânsito
   - Complementar os dados com outros datasets disponíveis
2. **Modelos para comparação entre dataset real e simulado**
   - Modelos de segmentação
   - Modelos de detecção

## Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

- Afunilamento dos modelos para comparação
- Escolha do dataset real para comparação

**Observação: [caso precise fazer alguma observação, de qualquer "natureza"]**

## ACEITE DA ENTREGA:

**CEDRIC LUIZ DE CARVALHO:** Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

# Termo de Aceite de Entrega

## Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

## Data da Reunião ("gate") de aprovação: 30 de out. de 2024

## Participantes da Entrega [matriculados em Residência em IA]:

| |
|---|
| **Pedro Martins Bittencourt** |

## Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

O subproduto dessa entrega está em: Documentação residência 13112024

- **Escolha do dataset real para comparação**
  - **NuScenes**
    - 15h coletadas em Boston e Singapura em cenários variados
    - 390 mil frames
    - 6 câmeras + 5 radares + LiDaR 32 feixes
    - 23 classes de objeto
  - **SemanticKitti**
    - Coletado em Karlsruhe (Alemanha)
    - 43 mil frames
    - LiDaR 64 feixes
    - 28 classes de segmentação

- **Escolha dos modelos para comparação**

  **Detecção**
  - **PillarNeXt**
    - O modelo converte nuvens de pontos 3D em uma pseudo-imagem 2D.
    - Em seguida, usa uma série de convoluções para processar e compactar os dados.
    - São feitas as previsões finais de objetos, categorizando-os por tipo.

  **Segmentação**
  - **MvLiDARNet**
    - Realiza a projeção esférica da nuvem de pontos
    - Utiliza apenas camadas de convolução 2D para extrair características
    - Decoder recupera a resolução original para gerar a segmentação
    - Segmenta cada pixel

- Já configurei os ambientes para treinamento dos modelos.

- Encontrei mais datasets para treinamento

**Planejamento:** **[descrever o que pretende fazer para realizar a próxima ENTREGA]**

- Iniciar o treinamento dos modelos

**Observação: [caso precise fazer alguma observação, de qualquer "natureza"]**

## ACEITE DA ENTREGA:

**CEDRIC LUIZ DE CARVALHO:** Go!

# APÊNDICE 5

# Termo de Aceite de Entrega

## Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

## Data da Reunião ("gate") de aprovação: 27 de nov. de 2024

## Participantes da Entrega [matriculados em Residência em IA]:

| |
|---|
| **Pedro Martins Bittencourt** |

## Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

O subproduto dessa entrega está em: Documentação residência 27112024

- **Datasets para comparação**
  - **SemanticKitti** - Dataset real
  - **NPM3d-KittiCarla** - Dataset para comparação
- **Modelo para comparação**
  - **MvLiDARNet**
- Foi feita a conversão dos datasets para terem exatamente o mesmo formato e exatamente as mesmas classes.
- O modelo foi treinado no dataset simulado por 500 épocas. O resultado foi abaixo do esperado.

## Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

- Realizar experimentos "menores" investigando o que está acontecendo durante o treinamento e tentando entender o que está acontecendo com a rede.

**Observação: [caso precise fazer alguma observação, de qualquer "natureza"]**

## ACEITE DA ENTREGA:

**LEONARDO ANTÔNIO ALVES:** Em análise!

# APÊNDICE 6

## Termo de Aceite de Entrega

### Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

### Data da Reunião ("gate") de aprovação: **5 de dez. de 2024**

### Participantes da Entrega **[matriculados em Residência em IA]**:

| **Pedro Martins Bittencourt** |
|---|

### Entrega: **[descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]**

O subproduto dessa entrega está em:
Cópia de Exploração-e-treinamentos - Documentação Residência 05122024.ipynb

1. **Treino com Dados Reais** (Cenas 0, 1, 2, 3 e 4)
   - **Acurácia Geral**: 72.22%
   - **Destaques**: *Road* (82.08%), *Vehicles* (85.22%).

2. **Treino com Dados Simulados**
   - **Acurácia Geral**: 26.83%
   - **Destaques**: *Vegetation* (52.11%), *Person* (38.02%).

**Experimentos Mistos**

- **Conjunto 1**

- **Experimento 1**: Cenas reais (0, 1, 2)
  - **Acurácia Geral**: 69.44%
- **Experimento 2**: Cenas reais (0, 1, 2) + Cena simulada (5)
  - **Acurácia Geral**: 74.95%
- **Experimento 3**: Cenas reais (0, 1, 2) + Cenas simuladas (5, 6)
  - **Acurácia Geral**: 69.83%

- **Experimento 4**: Cenas reais (0, 1, 2) + Cenas simuladas (5, 6, 7)
  - **Acurácia Geral**: 76.40%

- **Conjunto 2**

- **Experimento 1**: Treino com cenas Reais (0, 1)
  - **Acurácia Geral**: 31.64%
  - **Destaques**: *Road* (54.23%), *Vehicles* (76.42%).
- **Experimento 2**: Reais (0, 1) + Simulada (5)
  - **Acurácia Geral**: 56.67%
  - **Destaques**: *Road* (81.78%), *Vegetation* (70.19%), *Vehicles* (82.93%).
- **Experimento 3**: Reais (0, 1) + Simuladas (5, 6)
  - **Acurácia Geral**: 61.35%
  - **Destaques**: *Road* (86.06%), *Vegetation* (74.56%), *Vehicles* (81.27%).

## Conclusões

- Dados reais são mais eficazes.
- Apenas dados simulados apresentam resultado extremamente baixo.
- A inclusão de dados simulados aumenta a robustez e melhora a generalização.
- Mistura gradual com dados simulados melhora desempenho geral.

**Planejamento:** **[descrever o que pretende fazer para realizar a próxima ENTREGA]**

**Observação: [caso precise fazer alguma observação, de qualquer "natureza"]**

## ACEITE DA ENTREGA:

**CEDRIC LUIZ DE CARVALHO:** Go!

## Imports

```
import os
import time
from colorcloud.chen2020mvlidarnet import MVLidarNet
from colorcloud.UFGsim2024infufg import UFGSimDataset
from colorcloud.UFGsim2024infufg import ProjectionToTensorTransformSim,
UFGSimDataset, SphericalProjection, ProjectionSimTransform,
ProjectionSimVizTransform, plot_projections
import lightning as L
from torchvision.transforms import v2
import numpy as np
import matplotlib.pyplot as plt
import torch
from torch.utils.data import DataLoader
from tqdm import tqdm
from torch.optim import AdamW
from torchmetrics.classification import Accuracy
from torchmetrics.segmentation import MeanIoU
from torchmetrics.classification import Dice
from torchmetrics.classification import MulticlassF1Score
from sklearn.metrics import accuracy_score

device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)

print(f"Using {device} device")
```

```
Using cuda device
```

## Default

```
proj = SphericalProjection(fov_up_deg=2., fov_down_deg=-24.8, W=2156, H=64)
tfms = v2.Compose([
    ProjectionSimTransform(proj),
    ProjectionToTensorTransformSim(),
])

# Inicializaçao do Dataset de Avaliacao
evaluation_dataset =
UFGSimDataset(data_path='/home/ceia-pedro/residencia/skitti', split='test',
```

```
transform=tfms)
evaluation_loader = DataLoader(
    evaluation_dataset,
    batch_size=256,
    shuffle=False,
    num_workers=16,
    pin_memory=True
)

print("Size of evaluation dataset: ", len(evaluation_dataset))
```

```
Size of evaluation dataset:  7864
```

```
def evaluate_on_large_dataset_by_class():

    class_names = {
        0: "unlabeled",
        1: "building",
        2: "fence",
        3: "person",
        4: "pole",
        5: "road",
        6: "sidewalk",
        7: "vegetation",
        8: "vehicles",
        9: "traffic-sign",
        10: "other-ground",
        11: "on-rails",
        12: "terrain",
    }

    model.eval()

    # Create containers for predictions and labels
    all_preds = []
    all_labels = []

    # Make predictions
    with torch.no_grad():
        for batch in evaluation_loader:
            test_item = {key: value.to(device) for key, value in
batch.items()}
            img = test_item['frame']
            label = test_item['label']
            mask = test_item['mask']
```

```
            label[~mask] = 0

            pred = model(img)
            pred_labels = torch.argmax(pred, dim=1)  # Get class
predictions
            pred_labels[~mask] = 0

            # Collect predictions and labels
            all_preds.append(pred_labels.cpu().numpy())
            all_labels.append(label.cpu().numpy())


    # Flatten arrays for accuracy calculation
    all_preds = np.concatenate([p.flatten() for p in all_preds])
    all_labels = np.concatenate([l.flatten() for l in all_labels])

    # Compute accuracy by class
    num_classes = model.module.n_classes if isinstance(model,
torch.nn.DataParallel) else model.n_classes
    class_accuracies = {}
    for cls in range(num_classes):
        cls_mask = all_labels == cls
        cls_preds = all_preds[cls_mask]
        cls_labels = all_labels[cls_mask]
        if len(cls_labels) > 0:
            accuracy = accuracy_score(cls_labels, cls_preds)
            class_accuracies[cls] = accuracy
        else:
            class_accuracies[cls] = None  # No samples for this class

    # Print accuracy by class with names
    for cls, acc in class_accuracies.items():
        class_name = class_names.get(cls, f"Class {cls}")
        if acc is not None:
            print(f"{class_name.capitalize()} accuracy = {acc * 100:.2f}%")
        else:
            print(f"{class_name}: No samples found.")
```

## Visualizaçao Dos Dados

### Dataset Simulado

```
data_path = '/home/ceia-pedro/residencia/npm3d-kitti-carla/data'
ds = UFGSimDataset(data_path)
```

```
test_ds = UFGSimDataset(data_path, split='test')
val_ds = UFGSimDataset(data_path, split='valid')
print(f'train size:\t{len(ds)}\ntest size:\t{len(test_ds)}\nvalid
size:\t{len(val_ds)}')
```

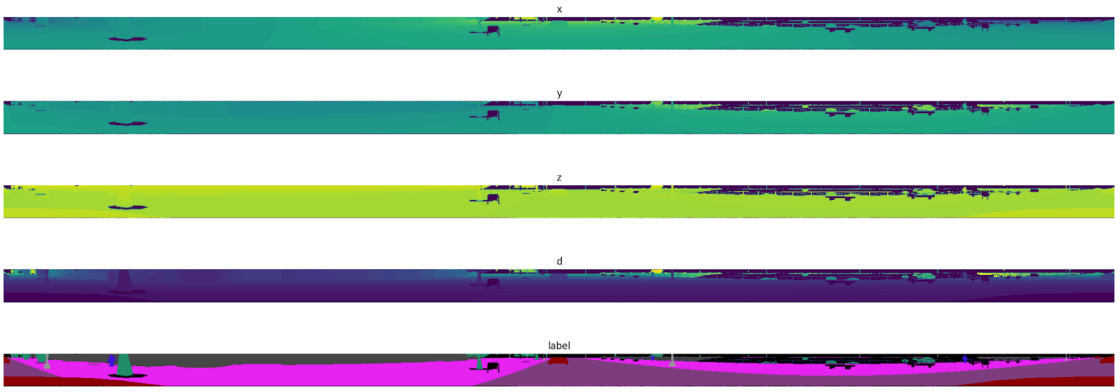
```
train size: 20000
test size:  10000
valid size: 5000
```

```
tfms_viz = v2.Compose([
    ProjectionSimTransform(proj),
    ProjectionSimVizTransform(ds.color_map_rgb_np, ds.learning_map_inv_np),
])
```

```
ds.set_transform(tfms_viz)
item = ds[128]
```

```
img = item['frame']
label = item['label']
channels_map = {"x": 0, "y": 1, "z": 2, "d": 3}
```

```
plot_projections(img, label, channels=['x', 'y', 'z', 'd'],
channels_map=channels_map)
```




## Dataset Real

```
data_path = '/home/ceia-pedro/residencia/skitti'
ds = UFGSimDataset(data_path)
```

```
test_ds = UFGSimDataset(data_path, split='test')
val_ds = UFGSimDataset(data_path, split='valid')
print(f'train size:\t{len(ds)}\ntest size:\t{len(test_ds)}\nvalid
size:\t{len(val_ds)}')
```

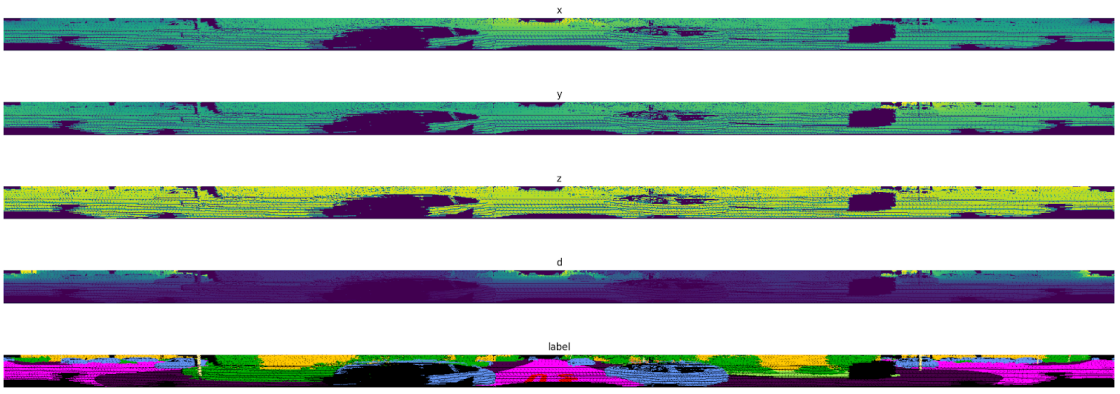
```
train size: 11375
test size:  10656
valid size: 2761
```

```
tfms_viz = v2.Compose([
    ProjectionSimTransform(proj),
    ProjectionSimVizTransform(ds.color_map_rgb_np, ds.learning_map_inv_np),
])
```

```
ds.set_transform(tfms_viz)
item = ds[128]
```

```
img = item['frame']
label = item['label']
channels_map = {"x": 0, "y": 1, "z": 2, "d": 3}
```

```
plot_projections(img, label, channels=['x', 'y', 'z', 'd'],
channels_map=channels_map)
```




## Treinos

```
n_epochs = 5
```

```
train_batchsize = 96
test_batchsize = 16
```

### Treino real

Aqui o modelo será treinado apenas com dados reais para comparativo.

```
data_path = '/home/ceia-pedro/residencia/skitti'
train_dataset = UFGSimDataset(data_path=data_path, split='train',
transform=tfms)
```

```
val_dataset = UFGSimDataset(data_path=data_path, split='valid',
transform=tfms)

print("Size of train dataset: ", len(train_dataset))
print("Size of val dataset: ", len(val_dataset))
```

```
Size of train dataset:  11375
Size of val dataset:  2761
```

```
train_loader = DataLoader(
    train_dataset,
    batch_size=train_batchsize,
    shuffle=True,
    num_workers=16,
    pin_memory=True
)

val_loader = DataLoader(
    val_dataset,
    batch_size=test_batchsize,
    shuffle=False,
    num_workers=16,
    pin_memory=True
)

model = MVLidarNet(in_channels=4, n_classes=13).to(device)

if device == "cuda":
    if torch.cuda.device_count() > 1:
        print(f"Using {torch.cuda.device_count()} GPUs")
        model = torch.nn.DataParallel(model)
    model = model.to(device)

loss_func = torch.nn.CrossEntropyLoss(reduction='none')
opt = AdamW(model.parameters(), lr=5e-4, eps=1e-5)
```

```
Using 2 GPUs
```

```
accuracy = Accuracy(task="multiclass",
num_classes=model.module.n_classes).to(device)
accuracy_dict = {"train": [], "val": []}

miou = MeanIoU(num_classes=model.module.n_classes).to(device)
miou_dict = {"train": [], "val": []}

dice = Dice(num_classes=model.module.n_classes).to(device)
dice_dict = {"train": [], "val": []}
```

```
mcf1s = MulticlassF1Score(num_classes=model.module.n_classes,
average="macro").to(device)
mcf1s_dict = {"train": [], "val": []}

train_steps = len(train_loader) // 96
print(f"Train steps: {train_steps}")
test_steps = len(val_loader) // 16
print(f"Test steps: {test_steps}")
H = {"train_loss": [], "test_loss": []} # store loss history
```

```
Train steps: 1
Test steps: 10
```

```
for epoch in tqdm(range(n_epochs), desc="Epochs"):  # Outer loop for epochs

    model.train()

    total_train_loss = 0
    total_test_loss = 0

    for batch in train_loader:
        train_item = {key: value.to(device) for key, value in
batch.items()}
        img = train_item['frame']
        label = train_item['label']
        mask = train_item['mask']

        label[~mask] = 0

        pred = model(img)
        train_loss = loss_func(pred, label)
        train_loss = train_loss[mask]
        train_loss = train_loss.mean()

        pred_f = torch.permute(pred, (0, 2, 3, 1)) # N,C,H,W -> N,H,W,C
        pred_f = torch.flatten(pred_f, 0, -2)      # N,H,W,C -> N*H*W,C
        mask_f = torch.flatten(mask)               # N,H,W   -> N*H*W
        pred_m = pred_f[mask_f, :]
        label_m = label[mask]
        current_train_acc = accuracy(pred_m, label_m)
        accuracy_dict["train"].append(current_train_acc)

        pred_labels = torch.argmax(pred, dim=1).to(device)
        mask_miou = (label != 0)
```

```
        pred_labels[~mask] = 0
        current_train_miou = miou(pred_labels, label)
        miou_dict["train"].append(current_train_miou)
        current_train_dice = dice(pred_labels, label)
        dice_dict["train"].append(current_train_dice)
        current_train_mcf1s = mcf1s(pred_labels, label)
        mcf1s_dict["train"].append(current_train_mcf1s)

        opt.zero_grad()
        train_loss.backward()
        opt.step()

        total_train_loss += train_loss


    with torch.no_grad():

        model.eval()

        for batch in val_loader:
            test_item = {key: value.to(device) for key, value in
batch.items()}
            img = test_item['frame']
            label = test_item['label']
            mask = test_item['mask']

            label[~mask] = 0

            pred = model(img)
            test_loss = loss_func(pred, label)
            test_loss = test_loss[mask]
            test_loss = test_loss.mean()

            pred_f = torch.permute(pred, (0, 2, 3, 1)) # N,C,H,W -> N,H,W,C
            pred_f = torch.flatten(pred_f, 0, -2)      # N,H,W,C -> N*H*W,C
            mask_f = torch.flatten(mask)               # N,H,W   -> N*H*W
            pred_m = pred_f[mask_f, :]
            label_m = label[mask]
            current_test_acc = accuracy(pred_m, label_m)
            accuracy_dict["val"].append(current_test_acc)

            pred_labels = torch.argmax(pred, dim=1).to(device)
            mask_miou = (label != 0)
            pred_labels[~mask] = 0
            current_test_miou = miou(pred_labels, label)
```

```
            miou_dict["val"].append(current_test_miou)
            current_test_dice = dice(pred_labels, label)
            dice_dict["val"].append(current_test_dice)
            current_test_mcf1s = mcf1s(pred_labels, label)
            mcf1s_dict["val"].append(current_test_mcf1s)

            total_test_loss += test_loss

    avg_train_loss = total_train_loss / train_steps
    avg_test_loss = total_test_loss / test_steps

    # Store loss history for graphical visualization
    H["train_loss"].append(avg_train_loss.cpu().detach().numpy())
    H["test_loss"].append(avg_test_loss.cpu().detach().numpy())

    print(f'Epoch: {epoch+1}/{n_epochs} | Train Loss: {train_loss:20} |
Test loss: {test_loss} | Train Accuracy: {current_train_acc} | Test
Accuracy: {current_test_acc}')

torch.save(model, "/home/ceia-pedro/residencia/models/real-5epocas.pt")
```

```
Epochs:  20%|██        | 1/5 [02:16<09:07, 136.84s/it]

Epoch: 1/5 | Train Loss:    1.133496642112732 | Test loss:
2.036736249923706 | Train Accuracy: 0.7044059038162231 | Test Accuracy:
0.42719218134880066

Epochs:  40%|████      | 2/5 [04:24<06:34, 131.63s/it]

Epoch: 2/5 | Train Loss:   0.9711300134658813 | Test loss:
1.4456897974014282 | Train Accuracy: 0.7482048869132996 | Test Accuracy:
0.6100313067436218

Epochs:  60%|██████    | 3/5 [06:33<04:20, 130.34s/it]

Epoch: 3/5 | Train Loss:   0.8235371708869934 | Test loss:
1.2959394454956055 | Train Accuracy: 0.7871510982513428 | Test Accuracy:
0.6259328722953796

Epochs:  80%|████████  | 4/5 [08:43<02:10, 130.21s/it]

Epoch: 4/5 | Train Loss:   0.7588906288146973 | Test loss:
1.0444179773330688 | Train Accuracy: 0.7966775894165039 | Test Accuracy:
0.7061572074890137

Epochs: 100%|██████████| 5/5 [10:52<00:00, 130.51s/it]
```

```
Epoch: 5/5 | Train Loss:   0.7043237686157227 | Test loss:
0.9789148569107056 | Train Accuracy: 0.8051655888557434 | Test Accuracy:
0.722219705581665
```

```
# Load the saved model
model_path = "/home/ceia-pedro/residencia/models/real-5epocas.pt"
model = torch.load(model_path)

# Move the model to the appropriate device
model = model.to(device)

evaluate_on_large_dataset_by_class()
```

```
Unlabeled accuracy = 100.00%
Building accuracy = 74.05%
Fence accuracy = 23.20%
Person accuracy = 0.11%
Pole accuracy = 9.07%
Road accuracy = 82.08%
Sidewalk accuracy = 77.84%
Vegetation accuracy = 80.57%
Vehicles accuracy = 85.22%
Traffic-sign accuracy = 1.31%
Other-ground accuracy = 0.03%
on-rails: No samples found.
Terrain accuracy = 59.40%
```

```
plt.style.use("ggplot")
plt.figure()
plt.plot(H["train_loss"], label="train_loss")
plt.plot(H["test_loss"], label="test_loss")
plt.title("Training Loss on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend(loc="lower left")
plt.show()

# Accuracy
plt.style.use("ggplot")
plt.figure()

train_accuracy = [x.cpu().numpy() for x in accuracy_dict["train"]]
val_accuracy = [x.cpu().numpy() for x in accuracy_dict["val"]]

plt.plot(train_accuracy, label="train_accuracy")
```

```
plt.plot(val_accuracy, label="val_accuracy")
plt.title("Accuracy")
plt.xlabel("Batch")
plt.ylabel("Accuracy")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

## Treino simulado

Aqui o modelo será treinado apenas com dados simulados.

```
data_path = '/home/ceia-pedro/residencia/npm3d-kitti-carla/data'
train_dataset = UFGSimDataset(data_path=data_path, split='train',
transform=tfms)
val_dataset = UFGSimDataset(data_path=data_path, split='valid',
transform=tfms)

print("Size of train dataset: ", len(train_dataset))
print("Size of val dataset: ", len(val_dataset))

train_loader = DataLoader(
    train_dataset,
    batch_size=train_batchsize,
    shuffle=True,
    num_workers=16,
    pin_memory=True
)

val_loader = DataLoader(
    val_dataset,
    batch_size=test_batchsize,
    shuffle=False,
    num_workers=16,
    pin_memory=True
)

model = MVLidarNet(in_channels=4, n_classes=13).to(device)

if device == "cuda":
    if torch.cuda.device_count() > 1:
        print(f"Using {torch.cuda.device_count()} GPUs")
        model = torch.nn.DataParallel(model)
    model = model.to(device)
```

```
loss_func = torch.nn.CrossEntropyLoss(reduction='none')
opt = AdamW(model.parameters(), lr=5e-4, eps=1e-5)

accuracy = Accuracy(task="multiclass",
num_classes=model.module.n_classes).to(device)
accuracy_dict = {"train": [], "val": []}

miou = MeanIoU(num_classes=model.module.n_classes).to(device)
miou_dict = {"train": [], "val": []}

dice = Dice(num_classes=model.module.n_classes).to(device)
dice_dict = {"train": [], "val": []}

mcf1s = MulticlassF1Score(num_classes=model.module.n_classes,
average="macro").to(device)
mcf1s_dict = {"train": [], "val": []}

train_steps = len(train_loader) // 96
print(f"Train steps: {train_steps}")
test_steps = len(val_loader) // 16
print(f"Test steps: {test_steps}")
H = {"train_loss": [], "test_loss": []} # store loss history

for epoch in tqdm(range(n_epochs), desc="Epochs"):  # Outer loop for epochs

    model.train()

    total_train_loss = 0
    total_test_loss = 0

    for batch in train_loader:
        train_item = {key: value.to(device) for key, value in
batch.items()}
        img = train_item['frame']
        label = train_item['label']
        mask = train_item['mask']

        label[~mask] = 0

        pred = model(img)
        train_loss = loss_func(pred, label)
        train_loss = train_loss[mask]
        train_loss = train_loss.mean()

        pred_f = torch.permute(pred, (0, 2, 3, 1)) # N,C,H,W -> N,H,W,C
```

```
        pred_f = torch.flatten(pred_f, 0, -2)       # N,H,W,C -> N*H*W,C
        mask_f = torch.flatten(mask)                # N,H,W   -> N*H*W
        pred_m = pred_f[mask_f, :]
        label_m = label[mask]
        current_train_acc = accuracy(pred_m, label_m)
        accuracy_dict["train"].append(current_train_acc)

        pred_labels = torch.argmax(pred, dim=1).to(device)
        mask_miou = (label != 0)
        pred_labels[~mask] = 0
        current_train_miou = miou(pred_labels, label)
        miou_dict["train"].append(current_train_miou)
        current_train_dice = dice(pred_labels, label)
        dice_dict["train"].append(current_train_dice)
        current_train_mcf1s = mcf1s(pred_labels, label)
        mcf1s_dict["train"].append(current_train_mcf1s)

        opt.zero_grad()
        train_loss.backward()
        opt.step()

        total_train_loss += train_loss


    with torch.no_grad():

        model.eval()

        for batch in val_loader:
            test_item = {key: value.to(device) for key, value in
batch.items()}
            img = test_item['frame']
            label = test_item['label']
            mask = test_item['mask']

            label[~mask] = 0

            pred = model(img)
            test_loss = loss_func(pred, label)
            test_loss = test_loss[mask]
            test_loss = test_loss.mean()

            pred_f = torch.permute(pred, (0, 2, 3, 1)) # N,C,H,W -> N,H,W,C
            pred_f = torch.flatten(pred_f, 0, -2)       # N,H,W,C -> N*H*W,C
            mask_f = torch.flatten(mask)                # N,H,W   -> N*H*W
```

```
            pred_m = pred_f[mask_f, :]
            label_m = label[mask]
            current_test_acc = accuracy(pred_m, label_m)
            accuracy_dict["val"].append(current_test_acc)

            pred_labels = torch.argmax(pred, dim=1).to(device)
            mask_miou = (label != 0)
            pred_labels[~mask] = 0
            current_test_miou = miou(pred_labels, label)
            miou_dict["val"].append(current_test_miou)
            current_test_dice = dice(pred_labels, label)
            dice_dict["val"].append(current_test_dice)
            current_test_mcf1s = mcf1s(pred_labels, label)
            mcf1s_dict["val"].append(current_test_mcf1s)

            total_test_loss += test_loss

    avg_train_loss = total_train_loss / train_steps
    avg_test_loss = total_test_loss / test_steps

    # Store loss history for graphical visualization
    H["train_loss"].append(avg_train_loss.cpu().detach().numpy())
    H["test_loss"].append(avg_test_loss.cpu().detach().numpy())

    print(f'Epoch: {epoch+1}/{n_epochs} | Train Loss: {train_loss:20} |
Test loss: {test_loss} | Train Accuracy: {current_train_acc} | Test
Accuracy: {current_test_acc}')

torch.save(model, "/home/ceia-pedro/residencia/models/simulado-5epocas.pt")

plt.style.use("ggplot")
plt.figure()
plt.plot(H["train_loss"], label="train_loss")
plt.plot(H["test_loss"], label="test_loss")
plt.title("Training Loss on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend(loc="lower left")
plt.show()

# Accuracy
plt.style.use("ggplot")
plt.figure()

train_accuracy = [x.cpu().numpy() for x in accuracy_dict["train"]]
val_accuracy = [x.cpu().numpy() for x in accuracy_dict["val"]]
```

```
plt.plot(train_accuracy, label="train_accuracy")
plt.plot(val_accuracy, label="val_accuracy")
plt.title("Accuracy")
plt.xlabel("Batch")
plt.ylabel("Accuracy")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

### Avaliaçao no Dataset real

```
evaluate_on_large_dataset_by_class()
```

```
Unlabeled accuracy = 100.00%
Building accuracy = 17.21%
Fence accuracy = 25.75%
Person accuracy = 38.02%
Pole accuracy = 14.06%
Road accuracy = 31.88%
Sidewalk accuracy = 24.92%
Vegetation accuracy = 52.11%
Vehicles accuracy = 4.03%
Traffic-sign accuracy = 0.15%
Other-ground accuracy = 0.05%
on-rails: No samples found.
Terrain accuracy = 1.24%
```

```
evaluation_data_path = '/home/ceia-pedro/residencia/skitti'  # Update to
your new dataset path
eval_dataset = UFGSimDataset(data_path=evaluation_data_path,
transform=tfms)
```

```
%%capture
model.eval()
```

```
# Initialize metrics
accuracy = Accuracy(task="multiclass", num_classes=13).to(device)

dice = Dice(num_classes=13).to(device)
mcf1s = MulticlassF1Score(num_classes=13, average="macro").to(device)

# Evaluation loop
total_eval_loss = 0
total_accuracy = 0
total_dice = 0
total_f1 = 0
```

```
print("Starting evaluation...")
with torch.no_grad():
    for batch in tqdm(evaluation_loader, desc="Evaluation Progress",
leave=False):
        eval_item = {key: value.to(device) for key, value in batch.items()}
        img = eval_item['frame']
        label = eval_item['label']
        mask = eval_item['mask']

        label[~mask] = 0

        pred = model(img)
        eval_loss = loss_func(pred, label)
        eval_loss = eval_loss[mask]
        eval_loss = eval_loss.mean()

        total_eval_loss += eval_loss

        # Compute metrics
        pred_classes = torch.argmax(pred, dim=1)
        total_accuracy += accuracy(pred_classes[mask], label[mask])
        total_dice += dice(pred_classes[mask], label[mask])
        total_f1 += mcf1s(pred_classes[mask], label[mask])
```

```
Starting evaluation...
```

```
# Calculate average metrics
avg_eval_loss = total_eval_loss / len(evaluation_loader)
avg_accuracy = total_accuracy / len(evaluation_loader)
avg_dice = total_dice / len(evaluation_loader)
avg_f1 = total_f1 / len(evaluation_loader)

# Print evaluation results
print("\nEvaluation Results:")
print(f"Average Loss: {avg_eval_loss:.4f}")
print(f"Accuracy: {avg_accuracy:.4f}")
print(f"Dice Coefficient: {avg_dice:.4f}")
print(f"Macro F1 Score: {avg_f1:.4f}")
```

```
Evaluation Results:
Average Loss: 2.7098
Accuracy: 0.2683
```

```
Dice Coefficient: 0.2683
Macro F1 Score: 0.1272
```

A acurácia ficou baixa.

## Experimentos - Conjunto 1

Aqui irei criar um novo dataset composto por uma mistura da seguinte forma:

- Cenas 0, 1, 2, 3 e 4: Vindas do dataset real
- Cenas 5, 6, 7, 8 e 9: Vindas do dataset simulado

### Experimento 1

Irei treinar somente com as cenas 0, 1 e 2

A cena 3 será utilizada para validaçao e a 4 para teste

```
split: # sequence numbers
    train:
      - 0
      - 1
      - 2
    valid:
      - 3
    test:
      - 4
```

```
data_path = '/home/ceia-pedro/residencia/mixed_dataset'
train_dataset = UFGSimDataset(data_path=data_path, split='train',
transform=tfms)
val_dataset = UFGSimDataset(data_path=data_path, split='valid',
transform=tfms)

print("Size of train dataset: ", len(train_dataset))
print("Size of val dataset: ", len(val_dataset))
```

```
Size of train dataset:  10303
Size of val dataset:  801
```

```
train_loader = DataLoader(
    train_dataset,
    batch_size=train_batchsize,
    shuffle=True,
    num_workers=16,
    pin_memory=True
)
```

```
val_loader = DataLoader(
    val_dataset,
    batch_size=test_batchsize,
    shuffle=False,
    num_workers=16,
    pin_memory=True
)

model = MVLidarNet(in_channels=4, n_classes=13).to(device)

if device == "cuda":
    if torch.cuda.device_count() > 1:
        print(f"Using {torch.cuda.device_count()} GPUs")
        model = torch.nn.DataParallel(model)
    model = model.to(device)

loss_func = torch.nn.CrossEntropyLoss(reduction='none')
opt = AdamW(model.parameters(), lr=5e-4, eps=1e-5)
```

```
Using 2 GPUs
```

```
accuracy = Accuracy(task="multiclass",
num_classes=model.module.n_classes).to(device)
accuracy_dict = {"train": [], "val": []}

miou = MeanIoU(num_classes=model.module.n_classes).to(device)
miou_dict = {"train": [], "val": []}

dice = Dice(num_classes=model.module.n_classes).to(device)
dice_dict = {"train": [], "val": []}

mcf1s = MulticlassF1Score(num_classes=model.module.n_classes,
average="macro").to(device)
mcf1s_dict = {"train": [], "val": []}

train_steps = len(train_loader) // 96
print(f"Train steps: {train_steps}")
test_steps = len(val_loader) // 16
print(f"Test steps: {test_steps}")
H = {"train_loss": [], "test_loss": []} # store loss history
```

```
Train steps: 1
Test steps: 3
```

```
for epoch in tqdm(range(n_epochs), desc="Epochs"):  # Outer loop for epochs

    model.train()

    total_train_loss = 0
    total_test_loss = 0

    for batch in train_loader:
        train_item = {key: value.to(device) for key, value in
batch.items()}
        img = train_item['frame']
        label = train_item['label']
        mask = train_item['mask']

        label[~mask] = 0

        pred = model(img)
        train_loss = loss_func(pred, label)
        train_loss = train_loss[mask]
        train_loss = train_loss.mean()

        pred_f = torch.permute(pred, (0, 2, 3, 1)) # N,C,H,W -> N,H,W,C
        pred_f = torch.flatten(pred_f, 0, -2)      # N,H,W,C -> N*H*W,C
        mask_f = torch.flatten(mask)               # N,H,W   -> N*H*W
        pred_m = pred_f[mask_f, :]
        label_m = label[mask]
        current_train_acc = accuracy(pred_m, label_m)
        accuracy_dict["train"].append(current_train_acc)

        pred_labels = torch.argmax(pred, dim=1).to(device)
        mask_miou = (label != 0)
        pred_labels[~mask] = 0
        current_train_miou = miou(pred_labels, label)
        miou_dict["train"].append(current_train_miou)
        current_train_dice = dice(pred_labels, label)
        dice_dict["train"].append(current_train_dice)
        current_train_mcf1s = mcf1s(pred_labels, label)
        mcf1s_dict["train"].append(current_train_mcf1s)

        opt.zero_grad()
        train_loss.backward()
        opt.step()

        total_train_loss += train_loss
```

```
    with torch.no_grad():

        model.eval()

        for batch in val_loader:
            test_item = {key: value.to(device) for key, value in
batch.items()}
            img = test_item['frame']
            label = test_item['label']
            mask = test_item['mask']

            label[~mask] = 0

            pred = model(img)
            test_loss = loss_func(pred, label)
            test_loss = test_loss[mask]
            test_loss = test_loss.mean()

            pred_f = torch.permute(pred, (0, 2, 3, 1)) # N,C,H,W -> N,H,W,C
            pred_f = torch.flatten(pred_f, 0, -2)      # N,H,W,C -> N*H*W,C
            mask_f = torch.flatten(mask)               # N,H,W   -> N*H*W
            pred_m = pred_f[mask_f, :]
            label_m = label[mask]
            current_test_acc = accuracy(pred_m, label_m)
            accuracy_dict["val"].append(current_test_acc)

            pred_labels = torch.argmax(pred, dim=1).to(device)
            mask_miou = (label != 0)
            pred_labels[~mask] = 0
            current_test_miou = miou(pred_labels, label)
            miou_dict["val"].append(current_test_miou)
            current_test_dice = dice(pred_labels, label)
            dice_dict["val"].append(current_test_dice)
            current_test_mcf1s = mcf1s(pred_labels, label)
            mcf1s_dict["val"].append(current_test_mcf1s)

            total_test_loss += test_loss

    avg_train_loss = total_train_loss / train_steps
    avg_test_loss = total_test_loss / test_steps

    # Store loss history for graphical visualization
    H["train_loss"].append(avg_train_loss.cpu().detach().numpy())
    H["test_loss"].append(avg_test_loss.cpu().detach().numpy())
```

```
    print(f'Epoch: {epoch+1}/{n_epochs} | Train Loss: {train_loss:20} |
Test loss: {test_loss} | Train Accuracy: {current_train_acc} | Test
Accuracy: {current_test_acc}')

torch.save(model,
"/home/ceia-pedro/residencia/models/C1E1-mixed-somente-3real-5epocas.pt")
```

```
Epochs:  20%|██        | 1/5 [01:47<07:08, 107.16s/it]

Epoch: 1/5 | Train Loss:   1.1841132640838623 | Test loss:
1.8666799068450928 | Train Accuracy: 0.6833579540252686 | Test Accuracy:
0.5179570913314819

Epochs:  40%|████      | 2/5 [03:34<05:21, 107.25s/it]

Epoch: 2/5 | Train Loss:   0.9709581136703491 | Test loss:
1.5306026935577393 | Train Accuracy: 0.7605248689651489 | Test Accuracy:
0.5771651864051819

Epochs:  60%|██████    | 3/5 [05:22<03:34, 107.39s/it]

Epoch: 3/5 | Train Loss:   0.8978733420372009 | Test loss:
1.4146476984024048 | Train Accuracy: 0.7692199945449829 | Test Accuracy:
0.6086363196372986

Epochs:  80%|████████  | 4/5 [07:08<01:47, 107.10s/it]

Epoch: 4/5 | Train Loss:   0.8407495021820068 | Test loss:
1.2408912181854248 | Train Accuracy: 0.7660348415374756 | Test Accuracy:
0.6403295993804932

Epochs: 100%|██████████| 5/5 [08:54<00:00, 106.94s/it]

Epoch: 5/5 | Train Loss:   0.7314092516899109 | Test loss:
1.1355966329574585 | Train Accuracy: 0.7985321283340454 | Test Accuracy:
0.6944388747215271
```

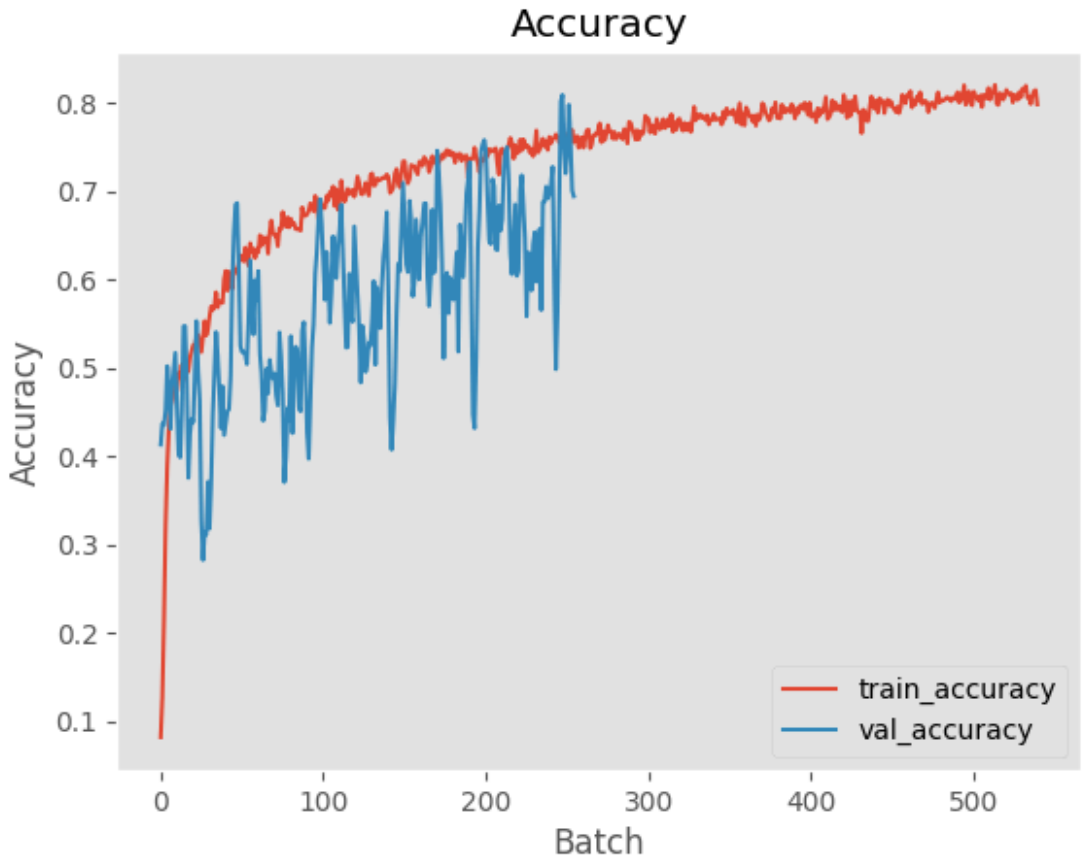
```
evaluate_on_large_dataset_by_class()

Unlabeled accuracy = 100.00%
Building accuracy = 82.10%
Fence accuracy = 22.08%
Person accuracy = 0.02%
Pole accuracy = 2.13%
Road accuracy = 88.77%
Sidewalk accuracy = 75.53%
```

```
Vegetation accuracy = 66.61%
Vehicles accuracy = 87.49%
Traffic-sign accuracy = 1.22%
Other-ground accuracy = 0.06%
on-rails: No samples found.
Terrain accuracy = 34.73%
```

```
plt.style.use("ggplot")
plt.figure()
plt.plot(H["train_loss"], label="train_loss")
plt.plot(H["test_loss"], label="test_loss")
plt.title("Training Loss on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend(loc="lower left")
plt.show()
```


```
# Accuracy
plt.style.use("ggplot")
plt.figure()
```

```
train_accuracy = [x.cpu().numpy() for x in accuracy_dict["train"]]
val_accuracy = [x.cpu().numpy() for x in accuracy_dict["val"]]

plt.plot(train_accuracy, label="train_accuracy")
plt.plot(val_accuracy, label="val_accuracy")
plt.title("Accuracy")
plt.xlabel("Batch")
plt.ylabel("Accuracy")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```




## Experimento 2

Irei acrescentar uma cena do dataset simulado ao treino: cena 5

```
split: # sequence numbers
    train:
      - 0
```

```
      - 1
      - 2
      - 5
    valid:
      - 3
    test:
      - 4

data_path = '/home/ceia-pedro/residencia/mixed_dataset'
train_dataset = UFGSimDataset(data_path=data_path, split='train',
transform=tfms)
val_dataset = UFGSimDataset(data_path=data_path, split='valid',
transform=tfms)

print("Size of train dataset: ", len(train_dataset))
print("Size of val dataset: ", len(val_dataset))

Size of train dataset:  15303
Size of val dataset:  801

train_loader = DataLoader(
    train_dataset,
    batch_size=train_batchsize,
    shuffle=True,
    num_workers=16,
    pin_memory=True
)

val_loader = DataLoader(
    val_dataset,
    batch_size=test_batchsize,
    shuffle=False,
    num_workers=16,
    pin_memory=True
)

model = MVLidarNet(in_channels=4, n_classes=13).to(device)

if device == "cuda":
    if torch.cuda.device_count() > 1:
        print(f"Using {torch.cuda.device_count()} GPUs")
        model = torch.nn.DataParallel(model)
    model = model.to(device)

loss_func = torch.nn.CrossEntropyLoss(reduction='none')
opt = AdamW(model.parameters(), lr=5e-4, eps=1e-5)
```

```
Using 2 GPUs
```

```
accuracy = Accuracy(task="multiclass",
num_classes=model.module.n_classes).to(device)
accuracy_dict = {"train": [], "val": []}

miou = MeanIoU(num_classes=model.module.n_classes).to(device)
miou_dict = {"train": [], "val": []}

dice = Dice(num_classes=model.module.n_classes).to(device)
dice_dict = {"train": [], "val": []}

mcf1s = MulticlassF1Score(num_classes=model.module.n_classes,
average="macro").to(device)
mcf1s_dict = {"train": [], "val": []}

train_steps = len(train_loader) // 96
print(f"Train steps: {train_steps}")
test_steps = len(val_loader) // 16
print(f"Test steps: {test_steps}")
H = {"train_loss": [], "test_loss": []} # store loss history
```

```
Train steps: 1
Test steps: 3
```

```
for epoch in tqdm(range(n_epochs), desc="Epochs"):  # Outer loop for epochs

    model.train()

    total_train_loss = 0
    total_test_loss = 0

    for batch in train_loader:
        train_item = {key: value.to(device) for key, value in
batch.items()}
        img = train_item['frame']
        label = train_item['label']
        mask = train_item['mask']

        label[~mask] = 0

        pred = model(img)
        train_loss = loss_func(pred, label)
        train_loss = train_loss[mask]
        train_loss = train_loss.mean()
```

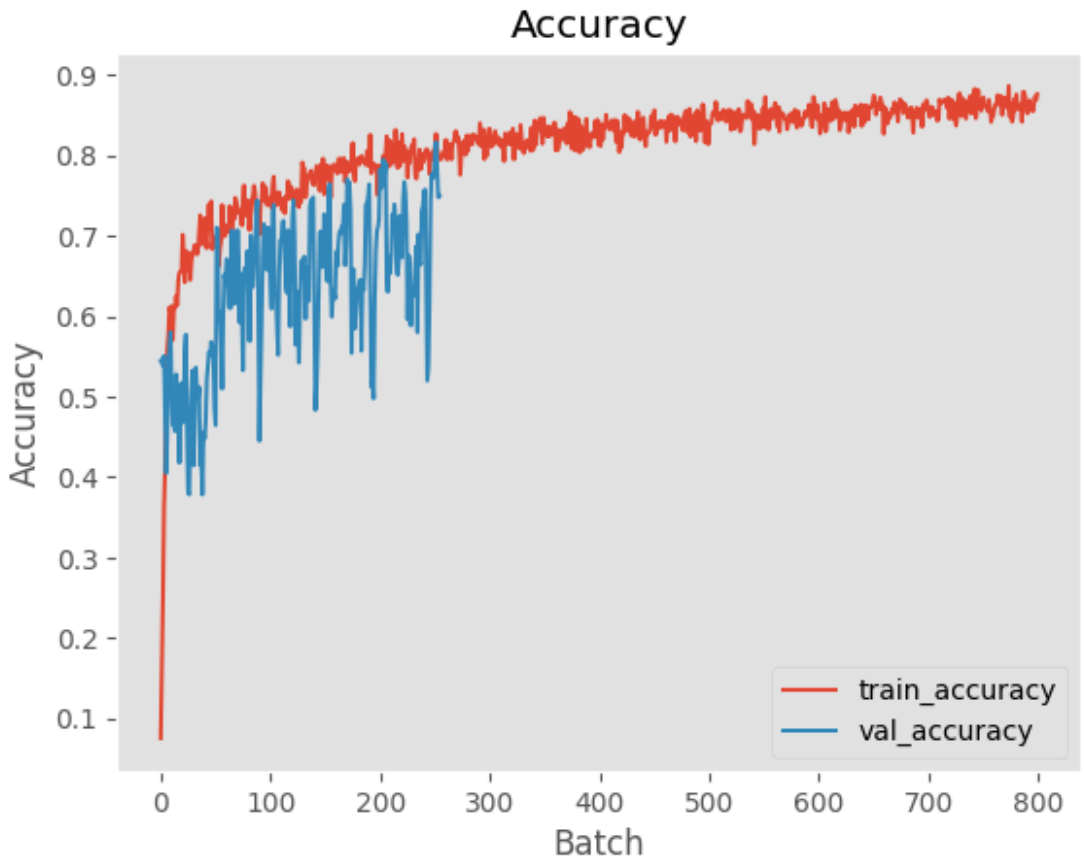
```
        pred_f = torch.permute(pred, (0, 2, 3, 1)) # N,C,H,W -> N,H,W,C
        pred_f = torch.flatten(pred_f, 0, -2)      # N,H,W,C -> N*H*W,C
        mask_f = torch.flatten(mask)               # N,H,W   -> N*H*W
        pred_m = pred_f[mask_f, :]
        label_m = label[mask]
        current_train_acc = accuracy(pred_m, label_m)
        accuracy_dict["train"].append(current_train_acc)

        pred_labels = torch.argmax(pred, dim=1).to(device)
        mask_miou = (label != 0)
        pred_labels[~mask] = 0
        current_train_miou = miou(pred_labels, label)
        miou_dict["train"].append(current_train_miou)
        current_train_dice = dice(pred_labels, label)
        dice_dict["train"].append(current_train_dice)
        current_train_mcf1s = mcf1s(pred_labels, label)
        mcf1s_dict["train"].append(current_train_mcf1s)

        opt.zero_grad()
        train_loss.backward()
        opt.step()

        total_train_loss += train_loss


    with torch.no_grad():

        model.eval()

        for batch in val_loader:
            test_item = {key: value.to(device) for key, value in
batch.items()}
            img = test_item['frame']
            label = test_item['label']
            mask = test_item['mask']

            label[~mask] = 0

            pred = model(img)
            test_loss = loss_func(pred, label)
            test_loss = test_loss[mask]
            test_loss = test_loss.mean()

            pred_f = torch.permute(pred, (0, 2, 3, 1)) # N,C,H,W -> N,H,W,C
            pred_f = torch.flatten(pred_f, 0, -2)      # N,H,W,C -> N*H*W,C
```

```
            mask_f = torch.flatten(mask)                # N,H,W   -> N*H*W
            pred_m = pred_f[mask_f, :]
            label_m = label[mask]
            current_test_acc = accuracy(pred_m, label_m)
            accuracy_dict["val"].append(current_test_acc)

            pred_labels = torch.argmax(pred, dim=1).to(device)
            mask_miou = (label != 0)
            pred_labels[~mask] = 0
            current_test_miou = miou(pred_labels, label)
            miou_dict["val"].append(current_test_miou)
            current_test_dice = dice(pred_labels, label)
            dice_dict["val"].append(current_test_dice)
            current_test_mcf1s = mcf1s(pred_labels, label)
            mcf1s_dict["val"].append(current_test_mcf1s)

            total_test_loss += test_loss

    avg_train_loss = total_train_loss / train_steps
    avg_test_loss = total_test_loss / test_steps

    # Store loss history for graphical visualization
    H["train_loss"].append(avg_train_loss.cpu().detach().numpy())
    H["test_loss"].append(avg_test_loss.cpu().detach().numpy())

    print(f'Epoch: {epoch+1}/{n_epochs} | Train Loss: {train_loss:20} |
Test loss: {test_loss} | Train Accuracy: {current_train_acc} | Test
Accuracy: {current_test_acc}')

torch.save(model,
"/home/ceia-pedro/residencia/models/C1E2-mixed-3real-1simulado-5epocas.pt")
```

```
Epochs:  20%|██        | 1/5 [02:32<10:09, 152.28s/it]

Epoch: 1/5 | Train Loss:   0.9656155109405518 | Test loss:
1.8444141149520874 | Train Accuracy: 0.7830221056938171 | Test Accuracy:
0.4646729528903961

Epochs:  40%|████      | 2/5 [05:05<07:38, 152.97s/it]

Epoch: 2/5 | Train Loss:   0.8217054605484009 | Test loss:
1.4610023498535156 | Train Accuracy: 0.8135089874267578 | Test Accuracy:
0.6095672249794006

Epochs:  60%|██████    | 3/5 [07:38<05:05, 152.89s/it]
```

```
Epoch: 3/5 | Train Loss:   0.6375949382781982 | Test loss:
1.2740142345428467 | Train Accuracy: 0.8554984927177429 | Test Accuracy:
0.643429160118103

Epochs:  80%|████████  | 4/5 [10:11<02:32, 152.79s/it]

Epoch: 4/5 | Train Loss:   0.6253005266189575 | Test loss:
0.898826003074646 | Train Accuracy: 0.8363531231880188 | Test Accuracy:
0.794818639755249

Epochs: 100%|██████████| 5/5 [12:42<00:00, 152.50s/it]

Epoch: 5/5 | Train Loss:   0.5159882307052612 | Test loss:
0.9153730869293213 | Train Accuracy: 0.8760408163070679 | Test Accuracy:
0.7495107650756836


evaluate_on_large_dataset_by_class()

Unlabeled accuracy = 100.00%
Building accuracy = 64.06%
Fence accuracy = 25.95%
Person accuracy = 0.01%
Pole accuracy = 17.52%
Road accuracy = 90.31%
Sidewalk accuracy = 75.30%
Vegetation accuracy = 80.89%
Vehicles accuracy = 84.36%
Traffic-sign accuracy = 0.06%
Other-ground accuracy = 0.01%
on-rails: No samples found.
Terrain accuracy = 37.68%

plt.style.use("ggplot")
plt.figure()
plt.plot(H["train_loss"], label="train_loss")
plt.plot(H["test_loss"], label="test_loss")
plt.title("Training Loss on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend(loc="lower left")
plt.show()
```


```
# Accuracy
plt.style.use("ggplot")
plt.figure()

train_accuracy = [x.cpu().numpy() for x in accuracy_dict["train"]]
val_accuracy = [x.cpu().numpy() for x in accuracy_dict["val"]]

plt.plot(train_accuracy, label="train_accuracy")
plt.plot(val_accuracy, label="val_accuracy")
plt.title("Accuracy")
plt.xlabel("Batch")
plt.ylabel("Accuracy")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```




## Experimento 3

Irei acrescentar duas cenas do dataset simulado ao treino: cenas 5 e 6

```
split: # sequence numbers
    train:
      - 0
      - 1
      - 2
      - 5
      - 6
    valid:
      - 3
    test:
      - 4
```

```
data_path = '/home/ceia-pedro/residencia/mixed_dataset'
train_dataset = UFGSimDataset(data_path=data_path, split='train',
transform=tfms)
```

```
val_dataset = UFGSimDataset(data_path=data_path, split='valid',
transform=tfms)

print("Size of train dataset: ", len(train_dataset))
print("Size of val dataset: ", len(val_dataset))
```

```
Size of train dataset:  20303
Size of val dataset:  801
```

```
train_loader = DataLoader(
    train_dataset,
    batch_size=train_batchsize,
    shuffle=True,
    num_workers=16,
    pin_memory=True
)

val_loader = DataLoader(
    val_dataset,
    batch_size=test_batchsize,
    shuffle=False,
    num_workers=16,
    pin_memory=True
)

model = MVLidarNet(in_channels=4, n_classes=13).to(device)

if device == "cuda":
    if torch.cuda.device_count() > 1:
        print(f"Using {torch.cuda.device_count()} GPUs")
        model = torch.nn.DataParallel(model)
    model = model.to(device)

loss_func = torch.nn.CrossEntropyLoss(reduction='none')
opt = AdamW(model.parameters(), lr=5e-4, eps=1e-5)
```

```
Using 2 GPUs
```

```
accuracy = Accuracy(task="multiclass",
num_classes=model.module.n_classes).to(device)
accuracy_dict = {"train": [], "val": []}

miou = MeanIoU(num_classes=model.module.n_classes).to(device)
miou_dict = {"train": [], "val": []}

dice = Dice(num_classes=model.module.n_classes).to(device)
dice_dict = {"train": [], "val": []}
```

```
mcf1s = MulticlassF1Score(num_classes=model.module.n_classes,
average="macro").to(device)
mcf1s_dict = {"train": [], "val": []}

train_steps = len(train_loader) // 96
print(f"Train steps: {train_steps}")
test_steps = len(val_loader) // 16
print(f"Test steps: {test_steps}")
H = {"train_loss": [], "test_loss": []} # store loss history
```

```
Train steps: 2
Test steps: 3
```

```
for epoch in tqdm(range(n_epochs), desc="Epochs"):  # Outer loop for epochs

    model.train()

    total_train_loss = 0
    total_test_loss = 0

    for batch in train_loader:
        train_item = {key: value.to(device) for key, value in
batch.items()}
        img = train_item['frame']
        label = train_item['label']
        mask = train_item['mask']

        label[~mask] = 0

        pred = model(img)
        train_loss = loss_func(pred, label)
        train_loss = train_loss[mask]
        train_loss = train_loss.mean()

        pred_f = torch.permute(pred, (0, 2, 3, 1)) # N,C,H,W -> N,H,W,C
        pred_f = torch.flatten(pred_f, 0, -2)      # N,H,W,C -> N*H*W,C
        mask_f = torch.flatten(mask)               # N,H,W   -> N*H*W
        pred_m = pred_f[mask_f, :]
        label_m = label[mask]
        current_train_acc = accuracy(pred_m, label_m)
        accuracy_dict["train"].append(current_train_acc)

        pred_labels = torch.argmax(pred, dim=1).to(device)
        mask_miou = (label != 0)
```

```
        pred_labels[~mask] = 0
        current_train_miou = miou(pred_labels, label)
        miou_dict["train"].append(current_train_miou)
        current_train_dice = dice(pred_labels, label)
        dice_dict["train"].append(current_train_dice)
        current_train_mcf1s = mcf1s(pred_labels, label)
        mcf1s_dict["train"].append(current_train_mcf1s)

        opt.zero_grad()
        train_loss.backward()
        opt.step()

        total_train_loss += train_loss


    with torch.no_grad():

        model.eval()

        for batch in val_loader:
            test_item = {key: value.to(device) for key, value in
batch.items()}
            img = test_item['frame']
            label = test_item['label']
            mask = test_item['mask']

            label[~mask] = 0

            pred = model(img)
            test_loss = loss_func(pred, label)
            test_loss = test_loss[mask]
            test_loss = test_loss.mean()

            pred_f = torch.permute(pred, (0, 2, 3, 1)) # N,C,H,W -> N,H,W,C
            pred_f = torch.flatten(pred_f, 0, -2)      # N,H,W,C -> N*H*W,C
            mask_f = torch.flatten(mask)               # N,H,W   -> N*H*W
            pred_m = pred_f[mask_f, :]
            label_m = label[mask]
            current_test_acc = accuracy(pred_m, label_m)
            accuracy_dict["val"].append(current_test_acc)

            pred_labels = torch.argmax(pred, dim=1).to(device)
            mask_miou = (label != 0)
            pred_labels[~mask] = 0
            current_test_miou = miou(pred_labels, label)
```

```
            miou_dict["val"].append(current_test_miou)
            current_test_dice = dice(pred_labels, label)
            dice_dict["val"].append(current_test_dice)
            current_test_mcf1s = mcf1s(pred_labels, label)
            mcf1s_dict["val"].append(current_test_mcf1s)

            total_test_loss += test_loss

    avg_train_loss = total_train_loss / train_steps
    avg_test_loss = total_test_loss / test_steps

    # Store loss history for graphical visualization
    H["train_loss"].append(avg_train_loss.cpu().detach().numpy())
    H["test_loss"].append(avg_test_loss.cpu().detach().numpy())

    print(f'Epoch: {epoch+1}/{n_epochs} | Train Loss: {train_loss:20} |
Test loss: {test_loss} | Train Accuracy: {current_train_acc} | Test
Accuracy: {current_test_acc}')

torch.save(model,
"/home/ceia-pedro/residencia/models/C1E3-mixed-3real-2simulado-5epocas.pt")
```

```
Epochs:  20%|██        | 1/5 [03:17<13:11, 197.80s/it]

Epoch: 1/5 | Train Loss:   0.8577293157577515 | Test loss:
1.8945338726043701 | Train Accuracy: 0.8233469724655151 | Test Accuracy:
0.3710211515426636

Epochs:  40%|████      | 2/5 [06:35<09:53, 197.98s/it]

Epoch: 2/5 | Train Loss:   0.7602049112319946 | Test loss:
1.742506742477417 | Train Accuracy: 0.8166345357894897 | Test Accuracy:
0.5019940733909607

Epochs:  60%|██████    | 3/5 [09:54<06:36, 198.22s/it]

Epoch: 3/5 | Train Loss:    0.570449709892273 | Test loss:
1.3147865533828735 | Train Accuracy: 0.8640643358230591 | Test Accuracy:
0.6147401332855225

Epochs:  80%|████████  | 4/5 [13:13<03:18, 198.40s/it]

Epoch: 4/5 | Train Loss:   0.4920496344566345 | Test loss:
0.9136136174201965 | Train Accuracy: 0.8784542679786682 | Test Accuracy:
0.741016149520874

Epochs: 100%|██████████| 5/5 [16:30<00:00, 198.17s/it]
```

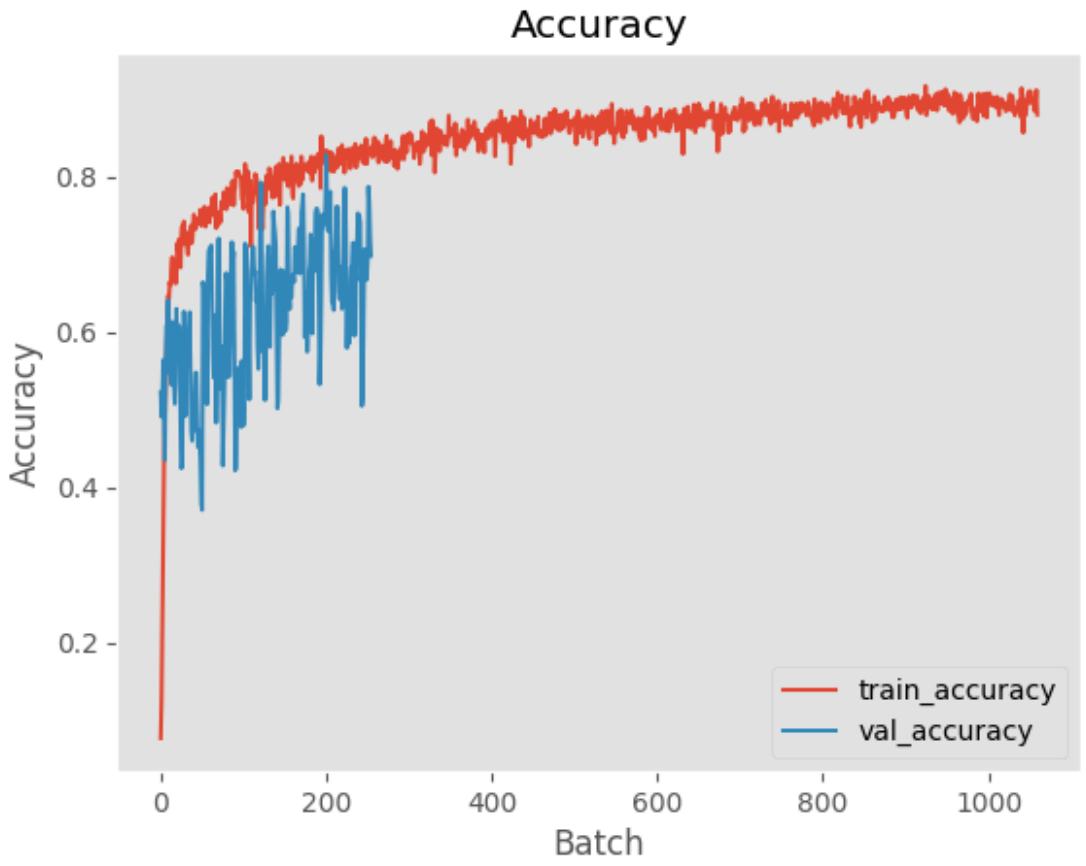
```
Epoch: 5/5 | Train Loss:   0.4579845368862152 | Test loss:
0.9446731209754944 | Train Accuracy: 0.880017101764679 | Test Accuracy:
0.6983000040054321
```

```
evaluate_on_large_dataset_by_class()
```

```
Unlabeled accuracy = 100.00%
Building accuracy = 71.95%
Fence accuracy = 37.20%
Person accuracy = 0.73%
Pole accuracy = 29.72%
Road accuracy = 89.36%
Sidewalk accuracy = 74.63%
Vegetation accuracy = 77.64%
Vehicles accuracy = 81.04%
Traffic-sign accuracy = 0.51%
Other-ground accuracy = 0.01%
on-rails: No samples found.
Terrain accuracy = 49.43%
```

```
plt.style.use("ggplot")
plt.figure()
plt.plot(H["train_loss"], label="train_loss")
plt.plot(H["test_loss"], label="test_loss")
plt.title("Training Loss on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend(loc="lower left")
plt.show()
```


```
# Accuracy
plt.style.use("ggplot")
plt.figure()

train_accuracy = [x.cpu().numpy() for x in accuracy_dict["train"]]
val_accuracy = [x.cpu().numpy() for x in accuracy_dict["val"]]

plt.plot(train_accuracy, label="train_accuracy")
plt.plot(val_accuracy, label="val_accuracy")
plt.title("Accuracy")
plt.xlabel("Batch")
plt.ylabel("Accuracy")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```




## Experimento 4

Irei acrescentar três cenas do dataset simulado ao treino: cenas 5, 6 e 7

```
split: # sequence numbers
    train:
      - 0
      - 1
      - 2
      - 5
      - 6
      - 7
    valid:
      - 3
    test:
      - 4
```

```
data_path = '/home/ceia-pedro/residencia/mixed_dataset'
train_dataset = UFGSimDataset(data_path=data_path, split='train',
transform=tfms)
```

```
val_dataset = UFGSimDataset(data_path=data_path, split='valid',
transform=tfms)

print("Size of train dataset: ", len(train_dataset))
print("Size of val dataset: ", len(val_dataset))
```

```
Size of train dataset:  25303
Size of val dataset:  801
```

```
train_loader = DataLoader(
    train_dataset,
    batch_size=train_batchsize,
    shuffle=True,
    num_workers=16,
    pin_memory=True
)

val_loader = DataLoader(
    val_dataset,
    batch_size=test_batchsize,
    shuffle=False,
    num_workers=16,
    pin_memory=True
)

model = MVLidarNet(in_channels=4, n_classes=13).to(device)

if device == "cuda":
    if torch.cuda.device_count() > 1:
        print(f"Using {torch.cuda.device_count()} GPUs")
        model = torch.nn.DataParallel(model)
    model = model.to(device)

loss_func = torch.nn.CrossEntropyLoss(reduction='none')
opt = AdamW(model.parameters(), lr=5e-4, eps=1e-5)
```

```
Using 2 GPUs
```

```
accuracy = Accuracy(task="multiclass",
num_classes=model.module.n_classes).to(device)
accuracy_dict = {"train": [], "val": []}

miou = MeanIoU(num_classes=model.module.n_classes).to(device)
miou_dict = {"train": [], "val": []}

dice = Dice(num_classes=model.module.n_classes).to(device)
dice_dict = {"train": [], "val": []}
```

```
mcf1s = MulticlassF1Score(num_classes=model.module.n_classes,
average="macro").to(device)
mcf1s_dict = {"train": [], "val": []}

train_steps = len(train_loader) // 96
print(f"Train steps: {train_steps}")
test_steps = len(val_loader) // 16
print(f"Test steps: {test_steps}")
H = {"train_loss": [], "test_loss": []} # store loss history
```

```
Train steps: 2
Test steps: 3
```

```
for epoch in tqdm(range(n_epochs), desc="Epochs"):  # Outer loop for epochs

    model.train()

    total_train_loss = 0
    total_test_loss = 0

    for batch in train_loader:
        train_item = {key: value.to(device) for key, value in
batch.items()}
        img = train_item['frame']
        label = train_item['label']
        mask = train_item['mask']

        label[~mask] = 0

        pred = model(img)
        train_loss = loss_func(pred, label)
        train_loss = train_loss[mask]
        train_loss = train_loss.mean()

        pred_f = torch.permute(pred, (0, 2, 3, 1)) # N,C,H,W -> N,H,W,C
        pred_f = torch.flatten(pred_f, 0, -2)      # N,H,W,C -> N*H*W,C
        mask_f = torch.flatten(mask)               # N,H,W   -> N*H*W
        pred_m = pred_f[mask_f, :]
        label_m = label[mask]
        current_train_acc = accuracy(pred_m, label_m)
        accuracy_dict["train"].append(current_train_acc)

        pred_labels = torch.argmax(pred, dim=1).to(device)
        mask_miou = (label != 0)
```

```
        pred_labels[~mask] = 0
        current_train_miou = miou(pred_labels, label)
        miou_dict["train"].append(current_train_miou)
        current_train_dice = dice(pred_labels, label)
        dice_dict["train"].append(current_train_dice)
        current_train_mcf1s = mcf1s(pred_labels, label)
        mcf1s_dict["train"].append(current_train_mcf1s)

        opt.zero_grad()
        train_loss.backward()
        opt.step()

        total_train_loss += train_loss


    with torch.no_grad():

        model.eval()

        for batch in val_loader:
            test_item = {key: value.to(device) for key, value in
batch.items()}
            img = test_item['frame']
            label = test_item['label']
            mask = test_item['mask']

            label[~mask] = 0

            pred = model(img)
            test_loss = loss_func(pred, label)
            test_loss = test_loss[mask]
            test_loss = test_loss.mean()

            pred_f = torch.permute(pred, (0, 2, 3, 1)) # N,C,H,W -> N,H,W,C
            pred_f = torch.flatten(pred_f, 0, -2)      # N,H,W,C -> N*H*W,C
            mask_f = torch.flatten(mask)               # N,H,W   -> N*H*W
            pred_m = pred_f[mask_f, :]
            label_m = label[mask]
            current_test_acc = accuracy(pred_m, label_m)
            accuracy_dict["val"].append(current_test_acc)

            pred_labels = torch.argmax(pred, dim=1).to(device)
            mask_miou = (label != 0)
            pred_labels[~mask] = 0
            current_test_miou = miou(pred_labels, label)
```

```
            miou_dict["val"].append(current_test_miou)
            current_test_dice = dice(pred_labels, label)
            dice_dict["val"].append(current_test_dice)
            current_test_mcf1s = mcf1s(pred_labels, label)
            mcf1s_dict["val"].append(current_test_mcf1s)

            total_test_loss += test_loss

    avg_train_loss = total_train_loss / train_steps
    avg_test_loss = total_test_loss / test_steps

    # Store loss history for graphical visualization
    H["train_loss"].append(avg_train_loss.cpu().detach().numpy())
    H["test_loss"].append(avg_test_loss.cpu().detach().numpy())

    print(f'Epoch: {epoch+1}/{n_epochs} | Train Loss: {train_loss:20} |
Test loss: {test_loss} | Train Accuracy: {current_train_acc} | Test
Accuracy: {current_test_acc}')

torch.save(model,
"/home/ceia-pedro/residencia/models/C1E4-mixed-3real-3simulado-5epocas.pt")
```

```
Epochs:  20%|██        | 1/5 [04:04<16:19, 244.88s/it]

Epoch: 1/5 | Train Loss:   0.7940828204154968 | Test loss:
1.8105934858322144 | Train Accuracy: 0.8527279496192932 | Test Accuracy:
0.43552908301353455

Epochs:  40%|████      | 2/5 [08:09<12:14, 244.71s/it]

Epoch: 2/5 | Train Loss:   0.6238391399383545 | Test loss:
1.208860993385315 | Train Accuracy: 0.8672217130661011 | Test Accuracy:
0.6548327803611755

Epochs:  60%|██████    | 3/5 [12:13<08:09, 244.59s/it]

Epoch: 3/5 | Train Loss:  0.49471890926361084 | Test loss:
0.9978861212730408 | Train Accuracy: 0.9018779397010803 | Test Accuracy:
0.7161883115768433

Epochs:  80%|████████  | 4/5 [16:17<04:04, 244.19s/it]

Epoch: 4/5 | Train Loss:  0.39941859245300293 | Test loss:
0.9768701195716858 | Train Accuracy: 0.9189314246177673 | Test Accuracy:
0.7193407416343689

Epochs: 100%|██████████| 5/5 [20:20<00:00, 244.02s/it]
```

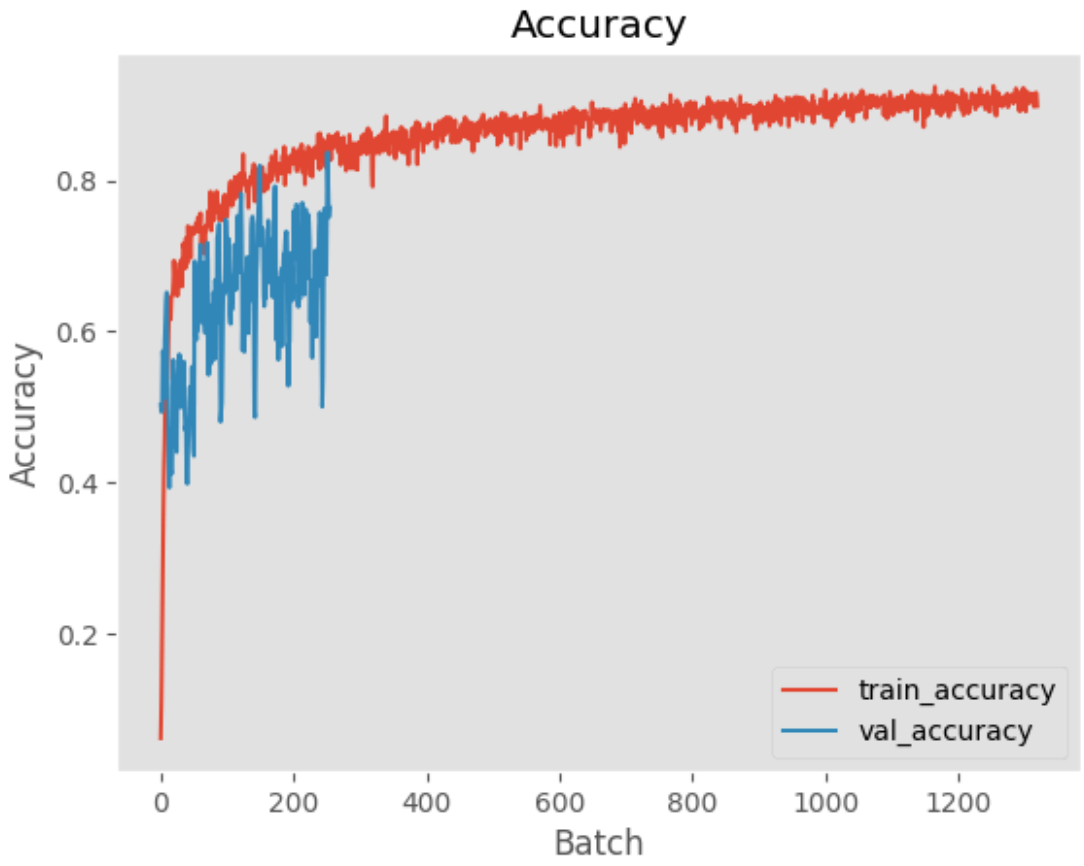
```
Epoch: 5/5 | Train Loss:  0.38714179396629333 | Test loss:
0.8498181104660034 | Train Accuracy: 0.8984812498092651 | Test Accuracy:
0.7640985250473022
```

```
evaluate_on_large_dataset_by_class()
```

```
Unlabeled accuracy = 100.00%
Building accuracy = 67.43%
Fence accuracy = 22.96%
Person accuracy = 0.22%
Pole accuracy = 18.52%
Road accuracy = 86.22%
Sidewalk accuracy = 76.46%
Vegetation accuracy = 84.29%
Vehicles accuracy = 77.31%
Traffic-sign accuracy = 0.16%
Other-ground accuracy = 0.00%
on-rails: No samples found.
Terrain accuracy = 52.33%
```

```
plt.style.use("ggplot")
plt.figure()
plt.plot(H["train_loss"], label="train_loss")
plt.plot(H["test_loss"], label="test_loss")
plt.title("Training Loss on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend(loc="lower left")
plt.show()
```


```
# Accuracy
plt.style.use("ggplot")
plt.figure()

train_accuracy = [x.cpu().numpy() for x in accuracy_dict["train"]]
val_accuracy = [x.cpu().numpy() for x in accuracy_dict["val"]]

plt.plot(train_accuracy, label="train_accuracy")
plt.plot(val_accuracy, label="val_accuracy")
plt.title("Accuracy")
plt.xlabel("Batch")
plt.ylabel("Accuracy")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```




## Experimentos - Conjunto 2

Irei usar o mesmo dataset do Conjunto 1

### Experimento 1

Irei treinar somente com as cenas 0 e 1

As cenas 2 e 3 serão utilizada para validaçao e a 4 para teste

```
split: # sequence numbers
    train:
      - 0
      - 1
    valid:
      - 2
      - 3
```

```
    test:
      - 4

data_path = '/home/ceia-pedro/residencia/mixed_dataset'
train_dataset = UFGSimDataset(data_path=data_path, split='train',
transform=tfms)
val_dataset = UFGSimDataset(data_path=data_path, split='valid',
transform=tfms)

print("Size of train dataset: ", len(train_dataset))
print("Size of val dataset: ", len(val_dataset))
```

```
Size of train dataset:  5642
Size of val dataset:  5462
```

```
train_loader = DataLoader(
    train_dataset,
    batch_size=train_batchsize,
    shuffle=True,
    num_workers=16,
    pin_memory=True
)

val_loader = DataLoader(
    val_dataset,
    batch_size=test_batchsize,
    shuffle=False,
    num_workers=16,
    pin_memory=True
)

model = MVLidarNet(in_channels=4, n_classes=13).to(device)

if device == "cuda":
    if torch.cuda.device_count() > 1:
        print(f"Using {torch.cuda.device_count()} GPUs")
        model = torch.nn.DataParallel(model)
    model = model.to(device)

loss_func = torch.nn.CrossEntropyLoss(reduction='none')
opt = AdamW(model.parameters(), lr=5e-4, eps=1e-5)
```

```
Using 2 GPUs
```

```
accuracy = Accuracy(task="multiclass",
num_classes=model.module.n_classes).to(device)
accuracy_dict = {"train": [], "val": []}
```

```
miou = MeanIoU(num_classes=model.module.n_classes).to(device)
miou_dict = {"train": [], "val": []}

dice = Dice(num_classes=model.module.n_classes).to(device)
dice_dict = {"train": [], "val": []}

mcf1s = MulticlassF1Score(num_classes=model.module.n_classes,
average="macro").to(device)
mcf1s_dict = {"train": [], "val": []}

train_steps = len(train_loader) // 96
print(f"Train steps: {train_steps}")
test_steps = len(val_loader) // 16
print(f"Test steps: {test_steps}")
H = {"train_loss": [], "test_loss": []} # store loss history
```

```
Train steps: 0
Test steps: 21
```

```
for epoch in tqdm(range(n_epochs), desc="Epochs"):  # Outer loop for epochs

    model.train()

    total_train_loss = 0
    total_test_loss = 0

    for batch in train_loader:
        train_item = {key: value.to(device) for key, value in
batch.items()}
        img = train_item['frame']
        label = train_item['label']
        mask = train_item['mask']

        label[~mask] = 0

        pred = model(img)
        train_loss = loss_func(pred, label)
        train_loss = train_loss[mask]
        train_loss = train_loss.mean()

        pred_f = torch.permute(pred, (0, 2, 3, 1)) # N,C,H,W -> N,H,W,C
        pred_f = torch.flatten(pred_f, 0, -2)      # N,H,W,C -> N*H*W,C
        mask_f = torch.flatten(mask)               # N,H,W   -> N*H*W
        pred_m = pred_f[mask_f, :]
```

```
        label_m = label[mask]
        current_train_acc = accuracy(pred_m, label_m)
        accuracy_dict["train"].append(current_train_acc)

        pred_labels = torch.argmax(pred, dim=1).to(device)
        mask_miou = (label != 0)
        pred_labels[~mask] = 0
        current_train_miou = miou(pred_labels, label)
        miou_dict["train"].append(current_train_miou)
        current_train_dice = dice(pred_labels, label)
        dice_dict["train"].append(current_train_dice)
        current_train_mcf1s = mcf1s(pred_labels, label)
        mcf1s_dict["train"].append(current_train_mcf1s)

        opt.zero_grad()
        train_loss.backward()
        opt.step()

        total_train_loss += train_loss


    with torch.no_grad():

        model.eval()

        for batch in val_loader:
            test_item = {key: value.to(device) for key, value in
batch.items()}
            img = test_item['frame']
            label = test_item['label']
            mask = test_item['mask']

            label[~mask] = 0

            pred = model(img)
            test_loss = loss_func(pred, label)
            test_loss = test_loss[mask]
            test_loss = test_loss.mean()

            pred_f = torch.permute(pred, (0, 2, 3, 1)) # N,C,H,W -> N,H,W,C
            pred_f = torch.flatten(pred_f, 0, -2)      # N,H,W,C -> N*H*W,C
            mask_f = torch.flatten(mask)               # N,H,W   -> N*H*W
            pred_m = pred_f[mask_f, :]
            label_m = label[mask]
            current_test_acc = accuracy(pred_m, label_m)
```

```
            accuracy_dict["val"].append(current_test_acc)

            pred_labels = torch.argmax(pred, dim=1).to(device)
            mask_miou = (label != 0)
            pred_labels[~mask] = 0
            current_test_miou = miou(pred_labels, label)
            miou_dict["val"].append(current_test_miou)
            current_test_dice = dice(pred_labels, label)
            dice_dict["val"].append(current_test_dice)
            current_test_mcf1s = mcf1s(pred_labels, label)
            mcf1s_dict["val"].append(current_test_mcf1s)

            total_test_loss += test_loss

    avg_train_loss = total_train_loss / train_steps
    avg_test_loss = total_test_loss / test_steps

    # Store loss history for graphical visualization
    H["train_loss"].append(avg_train_loss.cpu().detach().numpy())
    H["test_loss"].append(avg_test_loss.cpu().detach().numpy())

    print(f'Epoch: {epoch+1}/{n_epochs} | Train Loss: {train_loss:20} |
Test loss: {test_loss} | Train Accuracy: {current_train_acc} | Test
Accuracy: {current_test_acc}')

torch.save(model,
"/home/ceia-pedro/residencia/models/C2E1-mixed-somente-2real-5epocas.pt")
```

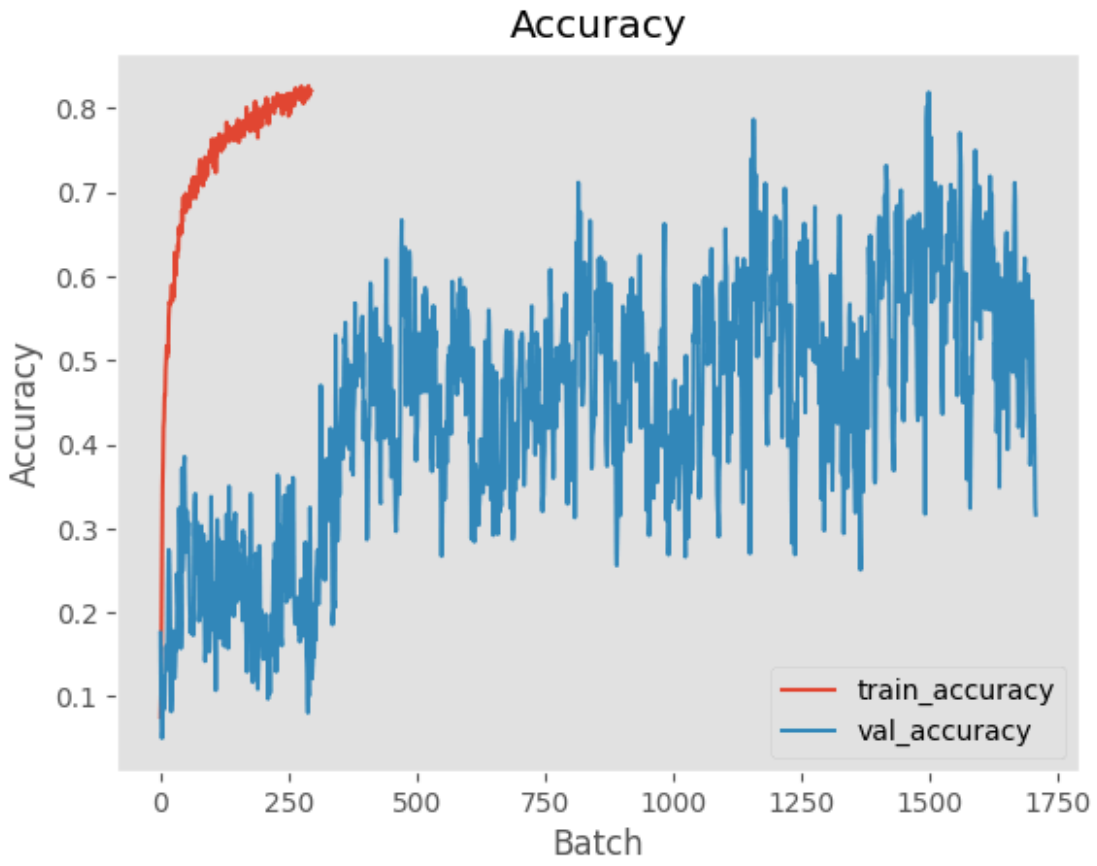
```
Epochs:  20%|██        | 1/5 [01:34<06:18, 94.60s/it]

Epoch: 1/5 | Train Loss:   1.1578325033187866 | Test loss:
2.7010693550109863 | Train Accuracy: 0.7078611254692078 | Test Accuracy:
0.20679818093776703

Epochs:  40%|████      | 2/5 [03:09<04:43, 94.54s/it]

Epoch: 2/5 | Train Loss:   0.9897028803825378 | Test loss:
2.1330580711364746 | Train Accuracy: 0.7609114646911621 | Test Accuracy:
0.32420167326927185

Epochs:  60%|██████    | 3/5 [04:43<03:08, 94.39s/it]

Epoch: 3/5 | Train Loss:   0.9083791971206665 | Test loss:
2.2064881324768066 | Train Accuracy: 0.7809157371520996 | Test Accuracy:
0.26625388860702515
```

```
Epochs:  80%|████████  | 4/5 [06:19<01:35, 95.02s/it]

Epoch: 4/5 | Train Loss:   0.8048232197761536 | Test loss:
2.260066509246826 | Train Accuracy: 0.8130600452423096 | Test Accuracy:
0.2507114112377167

Epochs: 100%|██████████| 5/5 [08:02<00:00, 96.52s/it]

Epoch: 5/5 | Train Loss:   0.7797973155975342 | Test loss:
2.0179052352905273 | Train Accuracy: 0.821016788482666 | Test Accuracy:
0.31635501980781555
```

```
evaluate_on_large_dataset_by_class()

Unlabeled accuracy = 100.00%
Building accuracy = 83.58%
Fence accuracy = 3.08%
Person accuracy = 0.10%
Pole accuracy = 5.72%
Road accuracy = 54.23%
Sidewalk accuracy = 86.38%
Vegetation accuracy = 64.41%
Vehicles accuracy = 76.42%
Traffic-sign accuracy = 1.03%
Other-ground accuracy = 0.00%
on-rails: No samples found.
Terrain accuracy = 25.88%
```

```
plt.style.use("ggplot")
plt.figure()
plt.plot(H["train_loss"], label="train_loss")
plt.plot(H["test_loss"], label="test_loss")
plt.title("Training Loss on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend(loc="lower left")
plt.show()
```


```
# Accuracy
plt.style.use("ggplot")
plt.figure()

train_accuracy = [x.cpu().numpy() for x in accuracy_dict["train"]]
val_accuracy = [x.cpu().numpy() for x in accuracy_dict["val"]]

plt.plot(train_accuracy, label="train_accuracy")
plt.plot(val_accuracy, label="val_accuracy")
plt.title("Accuracy")
plt.xlabel("Batch")
plt.ylabel("Accuracy")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```




## Experimento 2

Irei acrescentar uma cena do dataset simulado ao treino: cena 5

```
split: # sequence numbers
    train:
      - 0
      - 1
      - 5
    valid:
      - 2
      - 3
    test:
      - 4
```

```
data_path = '/home/ceia-pedro/residencia/mixed_dataset'
train_dataset = UFGSimDataset(data_path=data_path, split='train',
transform=tfms)
```

```
val_dataset = UFGSimDataset(data_path=data_path, split='valid',
transform=tfms)

print("Size of train dataset: ", len(train_dataset))
print("Size of val dataset: ", len(val_dataset))
```

```
Size of train dataset:  10642
Size of val dataset:  5462
```

```
train_loader = DataLoader(
    train_dataset,
    batch_size=train_batchsize,
    shuffle=True,
    num_workers=16,
    pin_memory=True
)

val_loader = DataLoader(
    val_dataset,
    batch_size=test_batchsize,
    shuffle=False,
    num_workers=16,
    pin_memory=True
)

model = MVLidarNet(in_channels=4, n_classes=13).to(device)

if device == "cuda":
    if torch.cuda.device_count() > 1:
        print(f"Using {torch.cuda.device_count()} GPUs")
        model = torch.nn.DataParallel(model)
    model = model.to(device)

loss_func = torch.nn.CrossEntropyLoss(reduction='none')
opt = AdamW(model.parameters(), lr=5e-4, eps=1e-5)
```

```
Using 2 GPUs
```

```
accuracy = Accuracy(task="multiclass",
num_classes=model.module.n_classes).to(device)
accuracy_dict = {"train": [], "val": []}

miou = MeanIoU(num_classes=model.module.n_classes).to(device)
miou_dict = {"train": [], "val": []}

dice = Dice(num_classes=model.module.n_classes).to(device)
dice_dict = {"train": [], "val": []}
```

```
mcf1s = MulticlassF1Score(num_classes=model.module.n_classes,
average="macro").to(device)
mcf1s_dict = {"train": [], "val": []}

train_steps = len(train_loader) // 96
print(f"Train steps: {train_steps}")
test_steps = len(val_loader) // 16
print(f"Test steps: {test_steps}")
H = {"train_loss": [], "test_loss": []} # store loss history
```

```
Train steps: 1
Test steps: 5
```

```
for epoch in tqdm(range(n_epochs), desc="Epochs"):  # Outer loop for epochs

    model.train()

    total_train_loss = 0
    total_test_loss = 0

    for batch in train_loader:
        train_item = {key: value.to(device) for key, value in
batch.items()}
        img = train_item['frame']
        label = train_item['label']
        mask = train_item['mask']

        label[~mask] = 0

        pred = model(img)
        train_loss = loss_func(pred, label)
        train_loss = train_loss[mask]
        train_loss = train_loss.mean()

        pred_f = torch.permute(pred, (0, 2, 3, 1)) # N,C,H,W -> N,H,W,C
        pred_f = torch.flatten(pred_f, 0, -2)      # N,H,W,C -> N*H*W,C
        mask_f = torch.flatten(mask)               # N,H,W   -> N*H*W
        pred_m = pred_f[mask_f, :]
        label_m = label[mask]
        current_train_acc = accuracy(pred_m, label_m)
        accuracy_dict["train"].append(current_train_acc)

        pred_labels = torch.argmax(pred, dim=1).to(device)
        mask_miou = (label != 0)
```

```
        pred_labels[~mask] = 0
        current_train_miou = miou(pred_labels, label)
        miou_dict["train"].append(current_train_miou)
        current_train_dice = dice(pred_labels, label)
        dice_dict["train"].append(current_train_dice)
        current_train_mcf1s = mcf1s(pred_labels, label)
        mcf1s_dict["train"].append(current_train_mcf1s)

        opt.zero_grad()
        train_loss.backward()
        opt.step()

        total_train_loss += train_loss


    with torch.no_grad():

        model.eval()

        for batch in val_loader:
            test_item = {key: value.to(device) for key, value in
batch.items()}
            img = test_item['frame']
            label = test_item['label']
            mask = test_item['mask']

            label[~mask] = 0

            pred = model(img)
            test_loss = loss_func(pred, label)
            test_loss = test_loss[mask]
            test_loss = test_loss.mean()

            pred_f = torch.permute(pred, (0, 2, 3, 1)) # N,C,H,W -> N,H,W,C
            pred_f = torch.flatten(pred_f, 0, -2)      # N,H,W,C -> N*H*W,C
            mask_f = torch.flatten(mask)               # N,H,W   -> N*H*W
            pred_m = pred_f[mask_f, :]
            label_m = label[mask]
            current_test_acc = accuracy(pred_m, label_m)
            accuracy_dict["val"].append(current_test_acc)

            pred_labels = torch.argmax(pred, dim=1).to(device)
            mask_miou = (label != 0)
            pred_labels[~mask] = 0
            current_test_miou = miou(pred_labels, label)
```

```
            miou_dict["val"].append(current_test_miou)
            current_test_dice = dice(pred_labels, label)
            dice_dict["val"].append(current_test_dice)
            current_test_mcf1s = mcf1s(pred_labels, label)
            mcf1s_dict["val"].append(current_test_mcf1s)

            total_test_loss += test_loss

    avg_train_loss = total_train_loss / train_steps
    avg_test_loss = total_test_loss / test_steps

    # Store loss history for graphical visualization
    H["train_loss"].append(avg_train_loss.cpu().detach().numpy())
    H["test_loss"].append(avg_test_loss.cpu().detach().numpy())

    print(f'Epoch: {epoch+1}/{n_epochs} | Train Loss: {train_loss:20} |
Test loss: {test_loss} | Train Accuracy: {current_train_acc} | Test
Accuracy: {current_test_acc}')

torch.save(model,
"/home/ceia-pedro/residencia/models/C2E2-mixed-2real-1simulado-5epocas.pt")
```

```
Epochs:  20%|██        | 1/5 [02:07<08:28, 127.25s/it]

Epoch: 1/5 | Train Loss:   0.9488118290901184 | Test loss: 2.39351749420166
| Train Accuracy: 0.804414689540863 | Test Accuracy: 0.21418748795986176

Epochs:  40%|████      | 2/5 [04:13<06:20, 126.86s/it]

Epoch: 2/5 | Train Loss:   0.8351630568504333 | Test loss:
1.7683930397033691 | Train Accuracy: 0.823197603225708 | Test Accuracy:
0.4789848029613495

Epochs:  60%|██████    | 3/5 [06:21<04:14, 127.13s/it]

Epoch: 3/5 | Train Loss:   0.7033883929252625 | Test loss:
1.6209205389022827 | Train Accuracy: 0.8578355312347412 | Test Accuracy:
0.5101271867752075

Epochs:  80%|████████  | 4/5 [08:32<02:08, 128.79s/it]

Epoch: 4/5 | Train Loss:   0.6427152752876282 | Test loss:
1.353434443473816 | Train Accuracy: 0.8645479679107666 | Test Accuracy:
0.6239138245582581

Epochs: 100%|██████████| 5/5 [10:44<00:00, 128.81s/it]
```

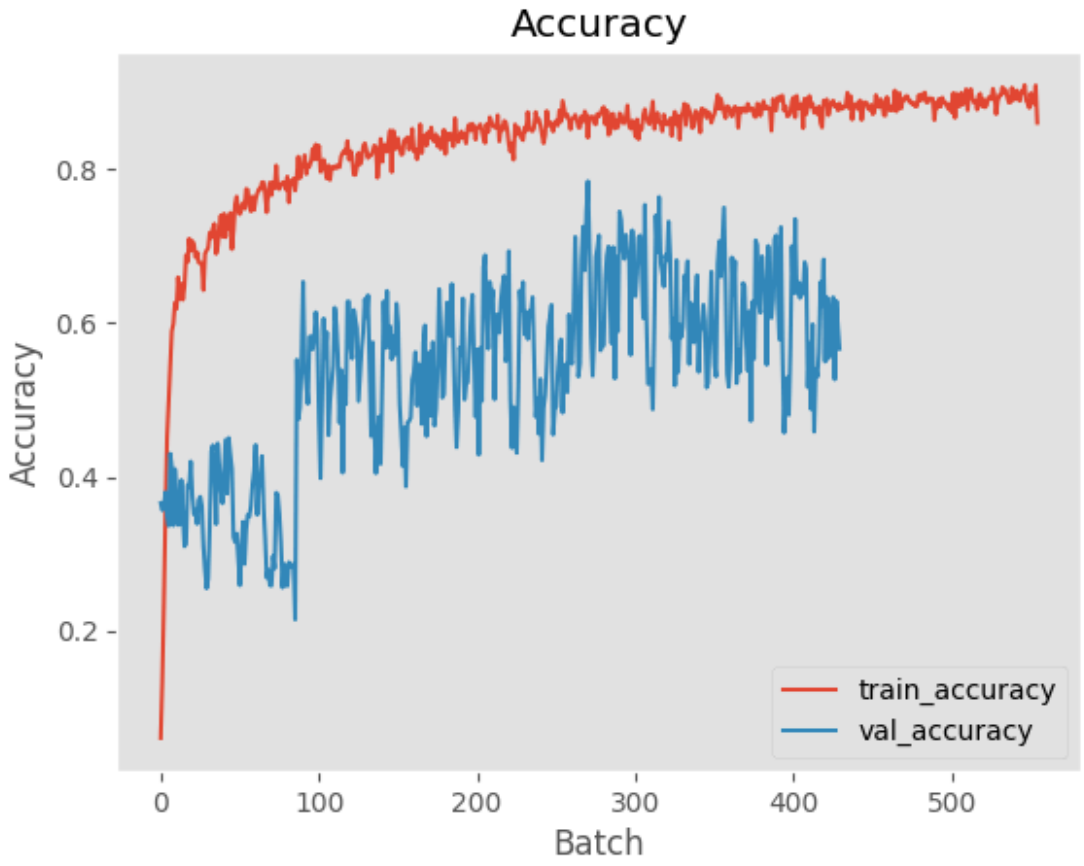
```
Epoch: 5/5 | Train Loss:   0.6070221662521362 | Test loss:
1.4171483516693115 | Train Accuracy: 0.8608864545822144 | Test Accuracy:
0.5666915774345398
```

```
evaluate_on_large_dataset_by_class()
```

```
Unlabeled accuracy = 100.00%
Building accuracy = 80.51%
Fence accuracy = 14.91%
Person accuracy = 0.02%
Pole accuracy = 13.61%
Road accuracy = 81.78%
Sidewalk accuracy = 62.23%
Vegetation accuracy = 70.19%
Vehicles accuracy = 82.93%
Traffic-sign accuracy = 0.03%
Other-ground accuracy = 0.00%
on-rails: No samples found.
Terrain accuracy = 56.41%
```

```
plt.style.use("ggplot")
plt.figure()
plt.plot(H["train_loss"], label="train_loss")
plt.plot(H["test_loss"], label="test_loss")
plt.title("Training Loss on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend(loc="lower left")
plt.show()
```


```
# Accuracy
plt.style.use("ggplot")
plt.figure()

train_accuracy = [x.cpu().numpy() for x in accuracy_dict["train"]]
val_accuracy = [x.cpu().numpy() for x in accuracy_dict["val"]]

plt.plot(train_accuracy, label="train_accuracy")
plt.plot(val_accuracy, label="val_accuracy")
plt.title("Accuracy")
plt.xlabel("Batch")
plt.ylabel("Accuracy")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```




## Experimento 3

Irei acrescentar duas cenas do dataset simulado ao treino: cenas 5 e 6

```
split: # sequence numbers
    train:
      - 0
      - 1
      - 5
      - 6
    valid:
      - 2
      - 3
    test:
      - 4
```

```
data_path = '/home/ceia-pedro/residencia/mixed_dataset'
train_dataset = UFGSimDataset(data_path=data_path, split='train',
transform=tfms)
```

```
val_dataset = UFGSimDataset(data_path=data_path, split='valid',
transform=tfms)

print("Size of train dataset: ", len(train_dataset))
print("Size of val dataset: ", len(val_dataset))
```

```
Size of train dataset:  15642
Size of val dataset:  5462
```

```
train_loader = DataLoader(
    train_dataset,
    batch_size=train_batchsize,
    shuffle=True,
    num_workers=16,
    pin_memory=True
)

val_loader = DataLoader(
    val_dataset,
    batch_size=test_batchsize,
    shuffle=False,
    num_workers=16,
    pin_memory=True
)

model = MVLidarNet(in_channels=4, n_classes=13).to(device)

if device == "cuda":
    if torch.cuda.device_count() > 1:
        print(f"Using {torch.cuda.device_count()} GPUs")
        model = torch.nn.DataParallel(model)
    model = model.to(device)

loss_func = torch.nn.CrossEntropyLoss(reduction='none')
opt = AdamW(model.parameters(), lr=5e-4, eps=1e-5)
```

```
Using 2 GPUs
```

```
accuracy = Accuracy(task="multiclass",
num_classes=model.module.n_classes).to(device)
accuracy_dict = {"train": [], "val": []}

miou = MeanIoU(num_classes=model.module.n_classes).to(device)
miou_dict = {"train": [], "val": []}

dice = Dice(num_classes=model.module.n_classes).to(device)
dice_dict = {"train": [], "val": []}
```

```
mcf1s = MulticlassF1Score(num_classes=model.module.n_classes,
average="macro").to(device)
mcf1s_dict = {"train": [], "val": []}

train_steps = len(train_loader) // 96
print(f"Train steps: {train_steps}")
test_steps = len(val_loader) // 16
print(f"Test steps: {test_steps}")
H = {"train_loss": [], "test_loss": []} # store loss history
```

```
Train steps: 1
Test steps: 5
```

```
for epoch in tqdm(range(n_epochs), desc="Epochs"):  # Outer loop for epochs

    model.train()

    total_train_loss = 0
    total_test_loss = 0

    for batch in train_loader:
        train_item = {key: value.to(device) for key, value in
batch.items()}
        img = train_item['frame']
        label = train_item['label']
        mask = train_item['mask']

        label[~mask] = 0

        pred = model(img)
        train_loss = loss_func(pred, label)
        train_loss = train_loss[mask]
        train_loss = train_loss.mean()

        pred_f = torch.permute(pred, (0, 2, 3, 1)) # N,C,H,W -> N,H,W,C
        pred_f = torch.flatten(pred_f, 0, -2)      # N,H,W,C -> N*H*W,C
        mask_f = torch.flatten(mask)               # N,H,W   -> N*H*W
        pred_m = pred_f[mask_f, :]
        label_m = label[mask]
        current_train_acc = accuracy(pred_m, label_m)
        accuracy_dict["train"].append(current_train_acc)

        pred_labels = torch.argmax(pred, dim=1).to(device)
        mask_miou = (label != 0)
```

```
        pred_labels[~mask] = 0
        current_train_miou = miou(pred_labels, label)
        miou_dict["train"].append(current_train_miou)
        current_train_dice = dice(pred_labels, label)
        dice_dict["train"].append(current_train_dice)
        current_train_mcf1s = mcf1s(pred_labels, label)
        mcf1s_dict["train"].append(current_train_mcf1s)

        opt.zero_grad()
        train_loss.backward()
        opt.step()

        total_train_loss += train_loss


    with torch.no_grad():

        model.eval()

        for batch in val_loader:
            test_item = {key: value.to(device) for key, value in
batch.items()}
            img = test_item['frame']
            label = test_item['label']
            mask = test_item['mask']

            label[~mask] = 0

            pred = model(img)
            test_loss = loss_func(pred, label)
            test_loss = test_loss[mask]
            test_loss = test_loss.mean()

            pred_f = torch.permute(pred, (0, 2, 3, 1)) # N,C,H,W -> N,H,W,C
            pred_f = torch.flatten(pred_f, 0, -2)      # N,H,W,C -> N*H*W,C
            mask_f = torch.flatten(mask)               # N,H,W   -> N*H*W
            pred_m = pred_f[mask_f, :]
            label_m = label[mask]
            current_test_acc = accuracy(pred_m, label_m)
            accuracy_dict["val"].append(current_test_acc)

            pred_labels = torch.argmax(pred, dim=1).to(device)
            mask_miou = (label != 0)
            pred_labels[~mask] = 0
            current_test_miou = miou(pred_labels, label)
```

```
            miou_dict["val"].append(current_test_miou)
            current_test_dice = dice(pred_labels, label)
            dice_dict["val"].append(current_test_dice)
            current_test_mcf1s = mcf1s(pred_labels, label)
            mcf1s_dict["val"].append(current_test_mcf1s)

            total_test_loss += test_loss

    avg_train_loss = total_train_loss / train_steps
    avg_test_loss = total_test_loss / test_steps

    # Store loss history for graphical visualization
    H["train_loss"].append(avg_train_loss.cpu().detach().numpy())
    H["test_loss"].append(avg_test_loss.cpu().detach().numpy())

    print(f'Epoch: {epoch+1}/{n_epochs} | Train Loss: {train_loss:20} |
Test loss: {test_loss} | Train Accuracy: {current_train_acc} | Test
Accuracy: {current_test_acc}')

torch.save(model,
"/home/ceia-pedro/residencia/models/C2E3-mixed-2real-2simulado-5epocas.pt")
```

```
Epochs:  20%|██        | 1/5 [02:54<11:39, 174.83s/it]

Epoch: 1/5 | Train Loss:   0.8908557295799255 | Test loss:
1.6367090940475464 | Train Accuracy: 0.8364602327346802 | Test Accuracy:
0.5803087949752808

Epochs:  40%|████      | 2/5 [05:50<08:46, 175.53s/it]

Epoch: 2/5 | Train Loss:   0.6410854458808899 | Test loss:
1.5913350582122803 | Train Accuracy: 0.9100509285926819 | Test Accuracy:
0.5596159100532532

Epochs:  60%|██████    | 3/5 [08:43<05:48, 174.18s/it]

Epoch: 3/5 | Train Loss:    0.569119393825531 | Test loss:
1.3814268112182617 | Train Accuracy: 0.9030068516731262 | Test Accuracy:
0.5973405838012695

Epochs:  80%|████████  | 4/5 [11:41<02:55, 175.70s/it]

Epoch: 4/5 | Train Loss:   0.4700583219528198 | Test loss:
1.3876988887786865 | Train Accuracy: 0.9221217632293701 | Test Accuracy:
0.5937120318412781

Epochs: 100%|██████████| 5/5 [14:34<00:00, 174.97s/it]
```

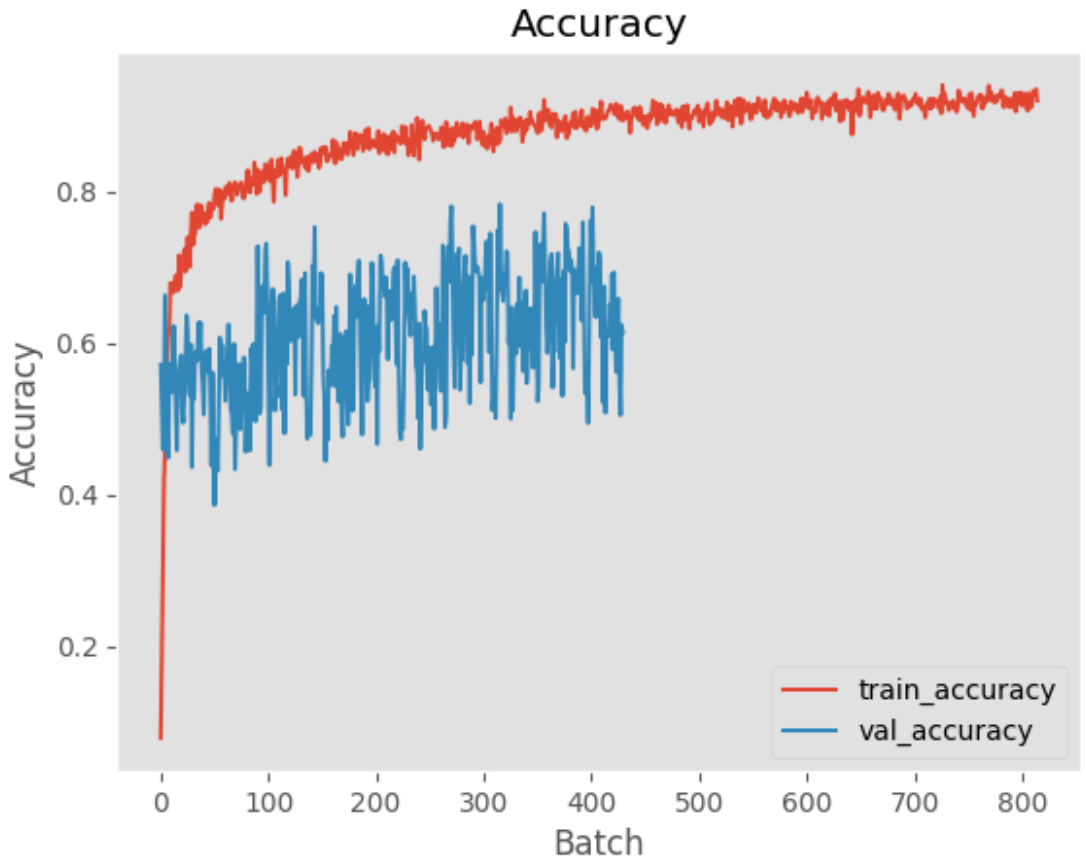
```
Epoch: 5/5 | Train Loss:   0.4189074635505676 | Test loss:
1.3424447774887085 | Train Accuracy: 0.9191882014274597 | Test Accuracy:
0.6135743856430054
```

```
evaluate_on_large_dataset_by_class()
```

```
Unlabeled accuracy = 100.00%
Building accuracy = 69.99%
Fence accuracy = 21.93%
Person accuracy = 0.35%
Pole accuracy = 27.46%
Road accuracy = 86.06%
Sidewalk accuracy = 67.72%
Vegetation accuracy = 74.56%
Vehicles accuracy = 81.27%
Traffic-sign accuracy = 0.75%
Other-ground accuracy = 0.04%
on-rails: No samples found.
Terrain accuracy = 50.59%
```

```
plt.style.use("ggplot")
plt.figure()
plt.plot(H["train_loss"], label="train_loss")
plt.plot(H["test_loss"], label="test_loss")
plt.title("Training Loss on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend(loc="lower left")
plt.show()
```


```
# Accuracy
plt.style.use("ggplot")
plt.figure()

train_accuracy = [x.cpu().numpy() for x in accuracy_dict["train"]]
val_accuracy = [x.cpu().numpy() for x in accuracy_dict["val"]]

plt.plot(train_accuracy, label="train_accuracy")
plt.plot(val_accuracy, label="val_accuracy")
plt.title("Accuracy")
plt.xlabel("Batch")
plt.ylabel("Accuracy")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```


Accuracy
Accuracy
0.8
0.6
0.4
0.2
0
100
200
300
400
500
600
700
800
Batch
train_accuracy
val_accuracy