



João Victor Farrulla Darze

easyGAN

Projeto Final de Programação

Projeto apresentado ao Programa de Pós-graduação em Informática do Departamento de Informática da PUC-Rio como requisito parcial para obtenção do grau de Mestre em Informática.

Orientador: Prof. Alberto Sardinha

Rio de Janeiro
Dezembro 2023

Sumário

1	Introdução	3
2	Especificação	4
2.1	Escopo	4
2.2	Especificação de Requisitos	4
3	Projeto do Programa	12
3.1	Diagrama de Classes	12
3.2	Tecnologias Utilizadas	13
3.3	Modelagem de Dados	13
4	Código Fonte	15
5	Testes	17
5.1	Critérios de Teste	17
5.2	Teste Manual	17
6	Documentação para o Usuário	18
6.1	Público-Alvo	18
6.2	Contexto de Atividade	18
6.3	Instalação e Execução	18
6.4	Contato	19
7	Bibliografia	20

1

Introdução

Modelos Machine Learning (ML), ou aprendizado de máquina, representa uma revolução significativa na sociedade contemporânea, promovendo avanços notáveis em diversas áreas. Desde seu início, o campo tem evoluído exponencialmente, moldando a forma como interagimos com a tecnologia e transformando setores inteiros. O ML utiliza algoritmos capazes de aprender padrões a partir de dados, permitindo que sistemas automatizados tomem decisões complexas com base em experiências passadas.

O início do ML remonta aos anos 50 e 60, com pioneiros como Arthur Samuel, que cunhou o termo "machine learning". Entretanto, foi a disponibilidade de grandes conjuntos de dados e poder computacional que impulsionou progressos significativos nas últimas décadas. A ascensão do deep learning, uma subcategoria do ML, trouxe à tona modelos mais complexos capazes de representar e aprender características abstratas.

Uma vertente fascinante do ML é a criação de modelos generativos de inteligência artificial (IA). Esses modelos têm a capacidade não apenas de aprender com dados existentes, mas também de gerar novos dados realistas. Entre os mais notáveis estão as Redes Generativas Adversárias (GANs). (GOODFELLOW et al., 2014) introduzem um paradigma único, onde dois modelos, um gerador e um discriminador, são treinados simultaneamente em um processo competitivo.

As GANs, em particular, destacam-se em aplicações que vão desde a geração de imagens realistas até a produção de conteúdo artístico inovador. Na indústria de entretenimento, as GANs têm sido usadas para criar faces humanas fictícias, paisagens e até mesmo obras de arte. Além disso, elas desempenham um papel crucial em setores como medicina, onde podem gerar imagens médicas sintéticas para treinamento de algoritmos de diagnóstico.

2

Especificação

Com o intuito de explorar a implementação de modelos Generativos Adversariais (GANs) na atualidade, analisar as práticas e desafios enfrentados por desenvolvedores e pesquisadores, eu concebi um projeto focado na criação e avaliação de um código-fonte para um modelo GAN específico, o DCGAN (Deep Convolutional GAN).

Para atingir esse objetivo, optei por desenvolver um código-fonte para um modelo GAN, abordando aspectos relevantes como arquitetura da rede, treinamento e avaliação de resultados. Este projeto visa não apenas criar um modelo funcional, mas também fornecer uma estrutura que facilite a compreensão e modificação por outros desenvolvedores interessados.

2.1

Escopo

De acordo com a dificuldade de ser desenvolver modelos de IA Generativa, buscamos implementar classes de acordo com o padrão de desenvolvimento sugerido pela tecnologia basilar que usamos, o *PyTorch*. Esse estabelece um conjunto de melhores práticas no desenvolvimento de arquiteturas de redes neurais.

Sendo assim abstraímos a dificuldade de se trabalhar com redes neurais e outros procedimentos normais para aqueles que desejam construir modelos GANs quando definindo e treinando tal modelo.

O sistema deve ser capaz de permitir que o desenvolvedor utilize os datasets provenientes da biblioteca *torchvision datasets* de maneira simples, e que tenha algum nível de liberdade quanto a criar a arquitetura da rede neural.

2.2

Especificação de Requisitos

Esta seção apresenta tanto os requisitos funcionais quanto os não-funcionais do projeto desenvolvido.

2.2.1

Requisitos Funcionais

[RF1] O sistema deve ser capaz de gerar um grid de imagens.

[RF2] O sistema deve permitir que o usuário receba informações sobre a arquitetura utilizada no modelo

[RF3] O sistema deve ser capaz de receber qualquer tipo de dataset de imagem proveniente da biblioteca *torchvision*

[RF4] O sistema deve fornecer a visualizações sobre os resultados de treinamento do modelo, os gráficos da *loss function*

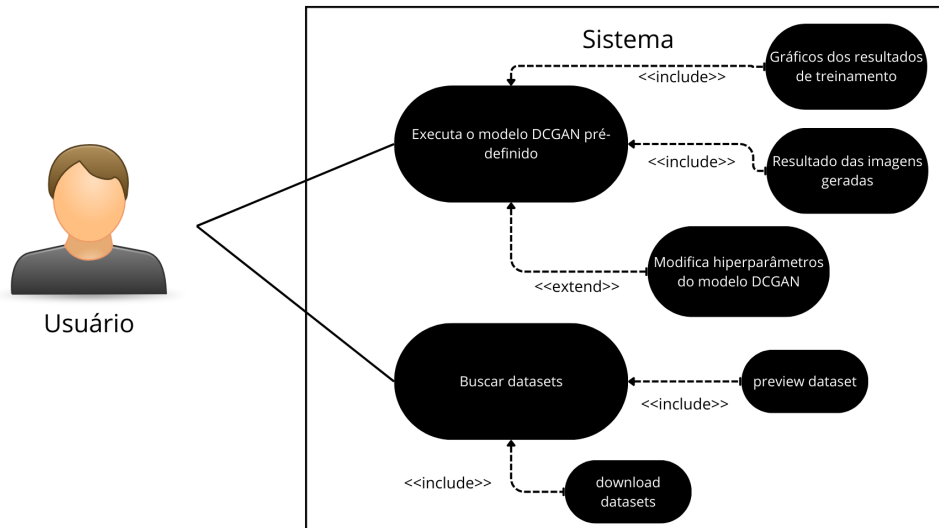


Figure 2.1: Diagrama de Caso de Uso

2.2.2

Requisitos Não-Funcionais

[RNF1] O sistema deve executar em qualquer uma das três distribuições: Windows, Linux e MacOS.

[RNF2] O sistema não deve aceitar datasets que não sejam provenientes do *Torchvision*.

[RNF3] O sistema deve ser possível de ser utilizado após a leitura do Guia do Usuário.

[RNF4] O sistema deve executar dentro de um ambiente virtual *venv* criado.

2.2.3

Casos de Uso

O diagrama de caso de uso é mostrado na Figura 2.1

2.2.4

Descrição dos Casos de Uso

Esta subseção descreve os casos de uso encontrados na Figura 2.1.

O primeiro caso de uso descreve o relacionamento do usuário com as visualizações pré-estabelecidas, como segue:

UC1 - Pesquisa Acadêmica em Ciência de Dados

Ator: Usuário é Doutorando em Ciência de Dados, com conhecimento técnico avançado em Python e aprendizado de máquina. Usuário faz pesquisa acadêmica.

Visão Geral: O usuário está conduzindo uma pesquisa na área de geração de imagens sintéticas para análise de dados. O uso do software contribuirá significativamente para gerar conjuntos de dados sintéticos que complementam suas amostras reais, melhorando a diversidade e a robustez de seu estudo. O usuário abre o terminal dentro da pasta do projeto e faz o procedimento para entrar em seu ambiente virtual *venv*. Após os devidos comandos, o usuário deve esperar a conclusão do mesmo para então digitar o comando: *jupyter notebook*. Uma aba no navegador irá se abrir e o usuário terá acesso ao fluxo corrente de escolha do dataset e treinamento do modelo para a geração de imagens.

Trigger: Não existe nenhum.

Pré-Condições: O usuário deve assegurar que sua máquina possui o *Python 3.9* e as dependências decalradas no arquivo *requirements.txt* instalados

Pós-Condições: O usuário terá acesso a imagens já geradas.

Sequência Esperada de Ações:

Ações do Ator	Respostas do Sistema
1. Ator clona o repositório do projeto em seu ambiente de desenvolvimento	
	2. O sistema instala as dependências e abre uma aba no navegador padrão do usuário
3. Ator seleciona um conjunto de dados relevante para sua pesquisa.	
4. Ator executa todas as células para treinar o modelo GAN	
	5. Sistema calcula e exibe os resultados das gerações. Imagens geradas e gráficos da <i>loss</i> resultante
6. Ator analisa os resultados, comparando as imagens geradas com as reais.	
6. Ator integra as imagens sintéticas em seu conjunto de dados para análise	

Fluxos Alternativos:

- 2a: O sistema não consegue instalar todas as dependências e exibe uma mensagem de erro.
- 4a: O ator não executa todas as células, mas somente algumas em específicos.

Avaliação:

- O usuário avalia que as imagens geradas são de alta qualidade e diversidade, contribuindo significativamente para a validade de sua pesquisa.

- Os resultados são consistentes com as expectativas, e o usuário incorpora as imagens geradas em seus relatórios e artigos acadêmicos.

UC2 - Desenvolvimento de Aplicações Artísticas

Ator: Usuário é desenvolvedor de jogos independentes. Tem conhecimento técnico em programação e design de jogos, também produz conteúdo artístico para jogos.

Visão Geral: O usuário deseja criar assets visuais únicos para seu próximo jogo, explorando a geração de imagens com o modelo GAN. O uso do software contribuirá para acelerar o processo de criação de assets visuais exclusivos e cativantes. O desenvolvedor faz o download do projeto e instala as dependências. Configura o ambiente virtual *venv* para garantir compatibilidade. Inicia o Jupyter Notebook para acessar a interface do projeto. Escolhe um conjunto de dados que se alinha ao estilo visual desejado para o jogo. Executa todas as células para treinar o modelo GAN e gerar imagens únicas.

Trigger: Não existe nenhum.

Pré-Condições: O usuário deve assegurar que sua máquina possui o *Python 3.9* e as dependências decalradas no arquivo *requirements.txt* instalados

Pós-Condições: O usuário terá acesso a imagens já geradas assim como os resultados gráficos do critério de treinamento selecionado.

Sequência Esperada de Ações:

Ações do Ator	Respostas do Sistema
1. Ator digita os comandos no terminal para criar seu ambiente virtual	
	2. O sistema instala as dependências e abre uma aba no navegador padrão do usuário
3. Ator seleciona uma base de dados	
4. Ator executa todas as células	
	5. Sistema calcula e exibe os resultados das gerações. Imagens geradas e gráficos da <i>loss</i> resultante

Fluxos Alternativos:

- 2a: O sistema não consegue instalar todas as dependências e exibe uma mensagem de erro.
- 4a: O ator não executa todas as células, mas somente algumas em específicos.

Avaliação:

- O desenvolvedor avalia que o modelo GAN proporcionou imagens que agregam valor estético ao seu jogo.

- A eficiência do processo de geração permite ao usuário economizar tempo no desenvolvimento artístico, concentrando-se em outras áreas do projeto.
- As imagens geradas são integradas de alguma forma aos assets do jogo.

O segundo, e último, caso de uso descreve o relacionamento do usuário com modificações na arquitetura das redes neurais e seus hiperparâmetros, como segue:

UC3 - Falha na Modificação da Arquitetura da Rede e Hiperparâmetros

Ator: Usuário é um estudante de pós-graduação em estatística, com conhecimento intermediário em programação Python e aprendizado de máquina.

Visão Geral: O usuário está explorando a possibilidade de otimizar a geração de imagens ajustando a arquitetura da rede neural e os hiperparâmetros do modelo GAN. Ele deseja testar diferentes configurações para obter resultados aprimorados.

Trigger: O problema ocorre durante a tentativa de modificar a arquitetura da rede e os hiperparâmetros.

Pré-Condições:

- O usuário deve assegurar que sua máquina possui o *Python 3.9* e as dependências declaradas no arquivo *requirements.txt*.
- O usuário deve modificar a declaração de hiperparâmetros e as especificações da rede neural.

Pós-Condições: O usuário não conseguirá realizar testes bem-sucedidos de hiperparâmetros e novas arquiteturas, resultando em falhas na geração de imagens.

Sequência Esperada de Ações:

Ações do Ator	Respostas do Sistema
1. Ator digita os comandos no terminal	
	2. O sistema instala as dependências e abre uma aba no navegador padrão do usuário
3. Ator seleciona novos hiperparâmetros conflitantes	
4. Ator gera uma nova especificação da rede neural com configurações incompatíveis	
5. Ator roda todas as células	
	6. Sistema exibe mensagem de erro indicando que as configurações de hiperparâmetros são conflitantes e impossibilitam o treinamento da rede neural.

Problema: Configurações de hiperparâmetros conflitantes impossibilitam o treinamento da rede neural.

Diagnóstico e Recuperação:

- O usuário pode revisar as configurações de hiperparâmetros, garantindo que não há conflitos entre eles.
- Verificar mensagens de erro detalhadas para identificar quais parâmetros estão causando o conflito.
- Consultar a documentação do modelo GAN para entender melhor as restrições e recomendações de configuração.

Indícios de Dificuldades Não Contornáveis:

- Se o problema persistir após ajustes, pode indicar limitações específicas do modelo GAN em lidar com determinadas combinações de hiperparâmetros.
- Mensagens de erro específicas podem apontar para questões mais profundas, como incompatibilidades entre versões de bibliotecas ou problemas de implementação.

UC4 - Problemas no Treinamento do Modelo GAN

Ator: Usuário é um estudante de graduação em ciência da computação com conhecimento básico em programação Python.

Visão Geral: O usuário está explorando a aplicação do modelo GAN para gerar imagens sintéticas como parte de um projeto acadêmico. Ele está interessado em entender o processo de treinamento do modelo e analisar os resultados para seu estudo.

Trigger: Durante a execução das células de treinamento, o usuário percebe que o processo parece travar ou leva muito tempo.

Pré-Condições: O usuário possui o Python 3.9 e as dependências declaradas no arquivo *requirements.txt* instalados.

Pós-Condições: O usuário deveria ter acesso às imagens geradas e aos resultados do treinamento.

Sequência Esperada de Ações:

Ações do Ator	Respostas do Sistema
1. Ator digita os comandos no terminal para criar seu ambiente virtual	
	2. O sistema instala as dependências e abre uma aba no navegador padrão do usuário
3. Ator seleciona uma base de dados	
4. Ator executa as células de treinamento	
	5. Sistema começa o treinamento do modelo GAN, exibindo progresso e resultados parciais

Problema: O treinamento demora excessivamente ou parece travar.

Diagnóstico e Recuperação:

- O usuário poderia verificar a quantidade de dados de treinamento, ajustar hiperparâmetros ou optar por um conjunto de dados menor para acelerar o processo.
- Monitorar a saída do terminal ou logs para identificar mensagens de erro ou advertências relacionadas ao treinamento.
- Caso o problema persista, o usuário poderia consultar a documentação para otimização ou procurar assistência em fóruns online.

Indícios de Dificuldades Não Contornáveis:

- Se o treinamento continua a falhar, mesmo após ajustes, pode indicar uma limitação de recursos computacionais do usuário.
- Mensagens de erro específicas podem indicar problemas mais profundos que exigem intervenção técnica especializada.

UC5 - Configuração de Ambiente Virtual

Ator: Usuário é um professor universitário de estatística, com conhecimento intermediário em Python e ambiente virtual.

Visão Geral: O usuário está explorando o modelo GAN como uma ferramenta de ensino para seus alunos em um curso de machine learning. Ele deseja configurar o ambiente virtual e executar o modelo para fornecer exemplos práticos aos seus estudantes.

Trigger: O usuário encontra dificuldades ao tentar configurar o ambiente virtual.

Pré-Condições: O usuário possui o Python 3.9 e as dependências declaradas no arquivo *requirements.txt* instalados.

Pós-Condições: O usuário deveria ter um ambiente virtual configurado e ser capaz de executar o modelo GAN.

Sequência Esperada de Ações:

Ações do Ator	Respostas do Sistema
1. Ator tenta criar um ambiente virtual utilizando os comandos padrões	
	2. O sistema inicia o processo de criação do ambiente virtual e instala as dependências
3. Ator ativa o ambiente virtual	
4. Ator tenta executar o Jupyter Notebook	
	5. Sistema abre uma aba no navegador com a interface do projeto

Problema: O ambiente virtual não é ativado corretamente.

Diagnóstico e Recuperação:

- O usuário pode verificar se o ambiente virtual foi ativado corretamente utilizando comandos específicos.
- Verificar se há conflitos com outros ambientes virtuais ou instalações globais de Python.
- Caso o problema persista, o usuário pode consultar a documentação do ambiente virtual ou buscar ajuda online.

Indícios de Dificuldades Não Contornáveis:

- Se o ambiente virtual não pode ser ativado devido a conflitos persistentes, pode indicar problemas de configuração mais complexos.
- Mensagens de erro específicas durante a ativação podem indicar problemas específicos que exigem análise mais aprofundada.

3

Projeto do Programa

Para atingir os objetivos do sistema, bem como os requisitos especificados, o sistema foi desenvolvido em **Python** em conjunto com os **notebooks** da biblioteca **Jupyter**. A liberdade e flexibilidade que os **notebooks** oferecem permitiu que fosse desenvolvido um *notebook* base com funções e classes úteis para o gerenciamento dos dados do *survey* para cada uma das questões. Contudo, essa estrutura não é restrita apenas às visualizações e estratégias já definidas, dado que o usuário pode aproveitar as variáveis e métodos definidos para criar novos gráficos e gerar novos *insights* com esforço reduzido. A Figura 3.1 apresenta uma visão macro da arquitetura da solução construída.

3.1

Diagrama de Classes

Os recursos desenvolvidos em Python foram baseados em três classes principais, que reúnem atributos e métodos específicos para gerenciamento dos dados de imagem provenientes do *torchvision*. As classes são definidas como segue e de acordo com a Figura 3.2.

- **Generator**, responsável pela criação e gerenciamento dos dados da rede neural geradora. Seu único método faz o *forward pass* dos dados pela rede neural.
- **Discriminator**, responsável pela criação e gerenciamento dos dados da rede neural discriminadora. Seu único método faz o *forward pass* dos dados pela rede neural.
- **GANTrainer**, responsável pela gestão do treinamento do modelo GAN. Inclui todos os métodos que se referem as etapas de treinamento.

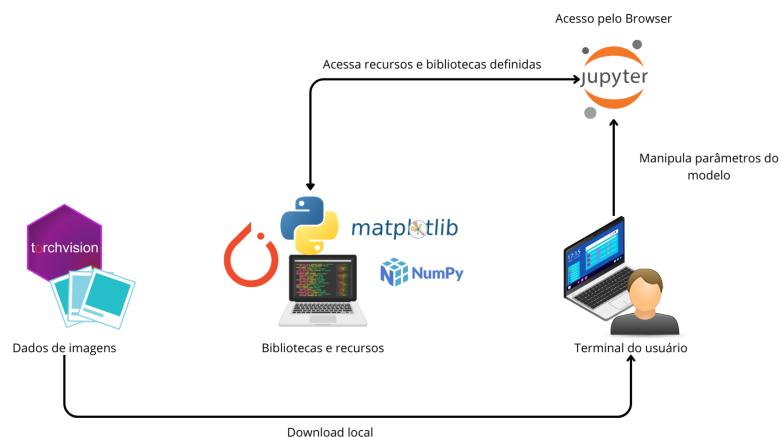


Figure 3.1: Arquitetura da Solução

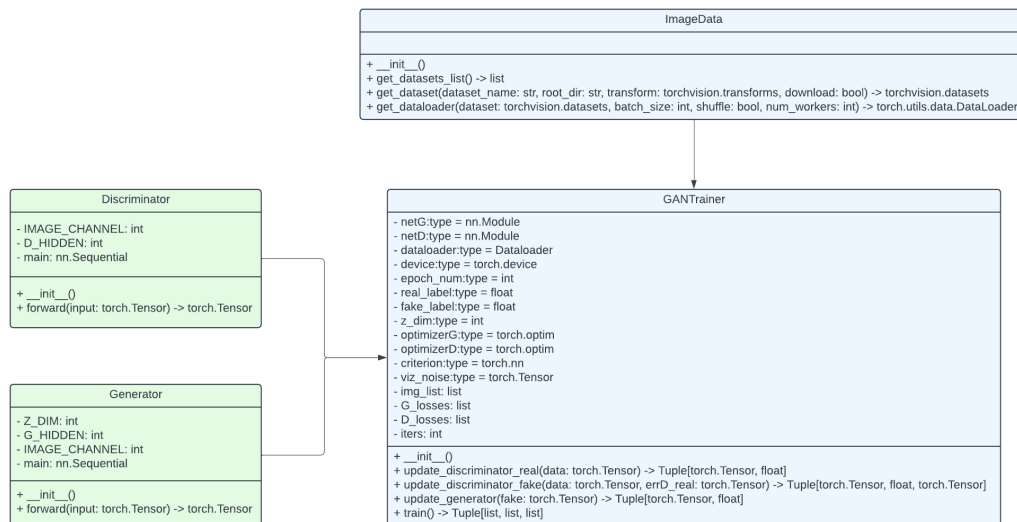


Figure 3.2: Diagrama de Classe

- **ImageData**, responsável pela escolha, download e preparação das bases de dados provenientes do *torchvision*.

3.2

Tecnologias Utilizadas

Por se tratar de um sistema desenvolvido em Python, que possui diversas bibliotecas disponíveis, é importante listar as que são específicas do projeto. Neste caso:

- *pytorch*
- *torchvision*
- *matplotlib*
- *numpy*
- *ipynb*

3.3

Modelagem de Dados

Ao utilizar a biblioteca *torchvision* para a obtenção dos dados, as imagens são inicialmente padronizadas e já vêm com características bem definidas e rotuladas. No entanto, o usuário tem a opção de realizar transformações nos dados para potencialmente melhorar os resultados do modelo. Essas transformações são frequentemente motivadas pelo processo de passagem dos dados por redes neurais, que muitas vezes apresentam um desempenho aprimorado quando os dados são normalizados, por exemplo.

Além disso, visando uma maior capacidade de generalização, é possível recorrer a técnicas como rotação e recorte de imagens para aumentar o conjunto de dados de treinamento. A alteração das dimensões das imagens também é uma prática comum para melhorar a definição do produto final.

É crucial destacar que, ao adicionar dados de fontes externas, o usuário deve estender o modelo para que ele possa acomodar a entrada dessas novas fontes de dados. Para isso, o modelo deve estender a classe *ImageData* ou similar, garantindo que as transformações necessárias para a compatibilidade com o *PyTorch* sejam aplicadas. Isso pode envolver a normalização e pré-processamento adequado para garantir que os novos dados se integrem harmoniosamente ao fluxo de treinamento da rede neural.

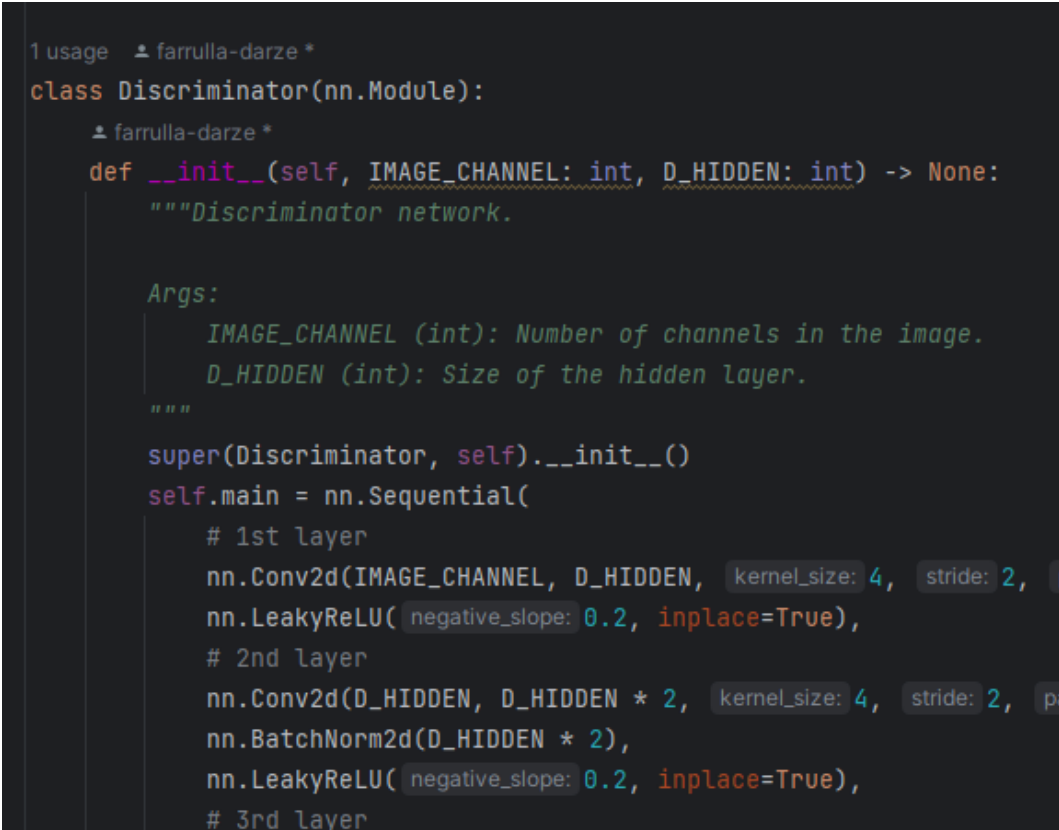
4

Código Fonte

O código deste projeto está hospedado no GitHub ¹, servindo como um repositório central para o acesso dos usuários e o versionamento do código. A escolha desta plataforma amplamente utilizada na comunidade de desenvolvimento de software facilita a colaboração e o controle de alterações ao longo do tempo.

O projeto foi implementado em Python, adotando os padrões de estilo e design definidos na PEP 8 ², que é uma referência para boas práticas na linguagem. Esse alinhamento às diretrizes estabelecidas contribui para a legibilidade e consistência do código, elementos essenciais em projetos colaborativos e para a manutenção a longo prazo.

Embora Python seja uma linguagem dinamicamente tipada, o código demonstra a prática de especificar tipos nos cabeçalhos de funções e nos retornos correspondentes. Essa abordagem não apenas adiciona uma camada de confiabilidade às funções, mas também aprimora a legibilidade do código, tornando mais claro o propósito e o uso de cada função. A combinação de tipagem e comentários no início de cada função, conforme ilustrado na Figura 4.1, aprimora a compreensão do código.



```
1 usage  👤 farrulla-darze *
class Discriminator(nn.Module):
    👤 farrulla-darze *
    def __init__(self, IMAGE_CHANNEL: int, D_HIDDEN: int) -> None:
        """Discriminator network.

        Args:
            IMAGE_CHANNEL (int): Number of channels in the image.
            D_HIDDEN (int): Size of the hidden layer.
        """
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
            # 1st layer
            nn.Conv2d(IMAGE_CHANNEL, D_HIDDEN, kernel_size: 4, stride: 2,
            nn.LeakyReLU(negative_slope: 0.2, inplace=True),
            # 2nd layer
            nn.Conv2d(D_HIDDEN, D_HIDDEN * 2, kernel_size: 4, stride: 2, pa
            nn.BatchNorm2d(D_HIDDEN * 2),
            nn.LeakyReLU(negative_slope: 0.2, inplace=True),
            # 3rd layer
```

Figure 4.1: Anotações e Tipagem

¹<https://github.com/farrulla-darze/easygan>

²<https://peps.python.org/pep-0008/>

Um aspecto significativo do desenvolvimento é a implementação do código dentro de um ambiente virtual Python. Esse ambiente virtual garante a isolamento e a consistência das bibliotecas utilizadas, bem como suas versões específicas. Ao ativar o ambiente virtual, o código pode ser executado com as dependências corretas, mitigando conflitos de versão e assegurando o correto funcionamento do projeto.

Esses detalhes de implementação evidenciam a preocupação com boas práticas, colaboração eficiente e consistência na execução do código, destacando a maturidade do desenvolvimento do projeto.

5

Testes

A verificação automatizada por meio de Testes Unitários foi conduzida para garantir a confiabilidade do sistema. Utilizei exclusivamente o pacote ***unittest*** para aplicar testes em todas as minhas classes e em algumas funções. Os códigos dos testes podem ser encontrados na pasta *tests* do repositório do GitHub e cada um possui descrições e motivações para terem sido realizados.

5.1

Critérios de Teste

Os critérios de teste utilizados consideraram todos os módulos disponíveis e passíveis de terem o seu retorno, de alguma forma, avaliados. Avaliamos inclusive funções que deveriam retornar gráficos ou até mesmo permitir a visualização de imagens. Como por exemplo funções no módulo ***utils*** que são destinadas a fazer coisas periféricas mas que enriquecem a experiência do usuário com diversas visualizações.

Todavia, existem alguns testes que não se mostraram como poderiam ser propriamente avaliados. Esse foi o caso da classe de treino da ***GANTrainer***

5.2

Teste Manual

Os testes manuais desempenham um papel crucial na avaliação de modelos generativos, como as Redes Generativas Adversariais (GANs). A interpretação humana continua a ser um componente essencial, especialmente na análise da qualidade perceptual das imagens geradas.

Além disso, nos testes manuais, é importante avaliar a diversidade e generalização das imagens geradas. A capacidade do modelo em produzir resultados variados e realistas em diferentes contextos é essencial para assegurar sua utilidade prática e evitar a geração de conteúdo monótono ou excessivamente específico. Em diversos casos as GANs se mostram instáveis no processo de treinamento e geram resultados visuais pobres.

6

Documentação para o Usuário

Nesta seção, alguns pontos sobre a utilidade e utilização do sistema serão respondidas.

6.1

Público-Alvo

Os usuários do projeto são desenvolvedores iniciando seus estudos sobre modelos generativos. Nesse caso focado em geração de imagens, esse código busca deixar o desenvolvedor livre para fazer a modificação dos componentes principais quanto desempenho das GANs. O usuário deve estar minimamente familiarizado com a linguagem de programação Python e deve ter uma boa compreensão sobre como redes neurais funcionam, mais especificamente usamos a biblioteca *PyTorch*. Um dos pontos positivos é a capacidade de abstração, em um primeiro momento, que o usuário pode ter quanto a tecnologia empregada para implementar redes neurais. Sendo assim o usuário deve focar no conhecimento basilar dos componentes de uma rede neural a fim de fazer modificações contundentes.

Contudo, o projeto também pode ser útil para desenvolvedores mais versados na biblioteca PyTorch, os quais poderiam estender as capacidades do código-fonte a comportar novos modelos de GANs os quais demandariam extensões relevantes no código.

6.2

Contexto de Atividade

O projeto está no contexto da rápida geração de resultados a partir do modelo DCGANs(Deep Convolutional GANS). Ele não permite rodar outras variações das GANs, ou gerar imagens a partir de *datasets* que não estão compreendidos dentro da biblioteca *torchvision*. Dessa forma buscamos evitar complexidades a princípios desnecessárias como preparar um *dataset* e assegurar que ele está nos padrões corretos para que os dados possam ser enviados para as redes neurais.

6.3

Instalação e Execução

A instalação e execução do projeto depende exclusivamente do *clone* do projeto do GitHub e acopmanhamento das instruções presentes no *README* do repositório. De modo resumido, para instalar corretamente o projeto é necessário ter a versão do **Python 3.9**, bem como usar o **venv** para criar seu ambiente virtual.

Em caso de insucesso na instalação, o GitHub conta com a aba *Issues*, na qual eu serei notificado caso algum usuário tenha algum problema nesta etapa.

6.4

Contato

Em caso de melhorias e sugestões, na descrição do projeto no GitHub, está o meu e-mail para contato.

7

Bibliografia

GOODFELLOW, I. J. et al. Generative adversarial networks. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2014. v. 27. Cited in page 3.