Universidade de São Paulo
Instituto de Ciências Matemáticas e Computação
SCC0233 - Aplicações de Aprendizado de Máquina e Mineração de Dados

Documentação - Projeto da Disciplina
Transformação de Fotos para Estilo-Monet usando CycleGan

Grupo:

- Alexandre Norcia Medeiros - 10295583
- Caio Abreu de Oliveira Ribeiro - 10262839
- Daniel Penna Chaves Bertazzo - 10349561
- Gabriel Alfonso Nascimento Salgueiro - 10284368
- Vinícius Torres Dutra Maia da Costa - 10262781

Sumário

1. Introdução

Esse projeto tem como objetivo estudar, desenvolver e realizar o treinamento completo de uma Rede Adversária Generativa (GAN). A proposta escolhida para a rede é a de realizar transformações de amostras de uma distribuição para outra distribuição. Assim, o principal objetivo do trabalho é entender e apresentar o funcionamento desta arquitetura de modelo de *deep learning*, além das escolhas feitas para a implementação da rede e os resultados obtidos.

Mais especificamente, optou-se pela transformação de pinturas do Monet para fotos realistas e fotos de paisagem para pinturas do estilo do Monet. A escolha desse domínio de aplicação foi feita pois o grupo considerou esta uma tarefa muito interessante. Isto, já que Claude Monet é conhecido por seu estilo impressionista e seu gosto de pintar paisagem. Assim, os quadros do Monet compreendem um conjunto interessante de imagens estilizadas, que seguem um mesmo padrão. Desta forma, um algoritmo que conseguisse converter fotos realistas em pinturas desse estilo, ou vice-versa, demonstraria um entendimento do conteúdo da imagem. Sendo o modelo capaz de alterar apenas o domínio da representação do conteúdo, sem alterar de fato o que está sendo representado. Assim, por exemplo, uma imagem que representa um campo de flores pode transitar entre o domínio de foto realista e o domínio de pintura impressionista através do modelo desenvolvido.

Decidiu-se desenvolver o projeto com base em redes adversárias generativas, pois este é um campo em alta na área de *deep learning*. Isto se deve ao fato de que essas redes conseguem resolver de maneira satisfatória diversos desafios de aprendizado de máquina. No caso, o desafio é a geração de novas amostras representativas de uma distribuição. Assim, os autores deste projeto acreditam que entender a estrutura desse modelo e os componentes usados para sua implementação é importante para compreender o estado da arte atual da área. Portanto, um dos principais objetivos foi o estudo sobre essas redes, além dos modelos de *Encoder-Decoders* e Redes Neurais Convolucionais, que podem compor GANs.

2. Desenvolvimento

Nesta seção é discutido sobre o que o grupo pesquisou e desenvolveu para a realização do projeto. Destaca-se que o grupo possuía um conhecimento prático de Redes Neurais anterior ao projeto dessa disciplina. Porém, apesar de alguns membros do grupo conhecerem a teoria dos *Autoencoders* e Redes Adversárias Generativas (*GAN*s), foi necessário uma pesquisa a fundo dos temas para se tornar possível a implementação do projeto. Desta forma, para o projeto da disciplina, o grupo produziu materiais de vídeo dos temas discutidos neste tópico, além da implementação do modelo escolhido.

Considerando que o objetivo principal desse projeto foi estudar a fundo e de maneira prática conceitos sobre GANs, esta seção da documentação apresenta a maior parte dos conceitos que os membros do grupo aprenderam durante o desenvolvimento do projeto. Assim, ela representa a maior parte do esforço dedicado ao desenvolvimento deste trabalho.

2.1 GAN - Rede Adversária Generativa

Redes Adversárias Generativas, ou *Generative Adversarial Networks* (*GAN*s), são redes que seguem uma arquitetura de aprendizado de máquina criada por Ian Goodfellow. Esta arquitetura propõe um processo de treinamento no qual dois modelos treinam simultaneamente, de forma que um aprende baseado no desempenho do outro. Este processo é constituído de um modelo generativo e um discriminativo. O modelo discriminativo tem como função decidir se uma entrada recebida pertence a uma distribuição real (verdadeira) ou gerada (falsa). O modelo generativo, por sua vez, tenta gerar instâncias novas (falsas) de determinada distribuição. Desta forma, o modelo generativo tem como objetivo gerar instâncias cada vez mais próximas de uma distribuição determinada tentando “enganar” o discriminativo, enquanto o discriminativo se aprimora para tentar diferenciar os resultados falsos produzidos daqueles que pertencem a distribuição verdadeira de dados. Assim, é possível estabelecer uma relação onde um modelo consegue usar o desempenho do outro para melhorar sua acurácia, de forma que a cada interação ambos modelos se aperfeiçoam.

Muitas vezes, *GAN*s são usadas em aplicações relacionadas a imagens. Isto acontece pois essa arquitetura se mostrou bastante eficiente em conseguir produzir imagens realistas de forma computacional. Não só isso, como, atualmente, as *GAN*s compõem o estado da arte da área de *Deep Learning* em aplicações de geração de imagens. Assim, ela se mostra um modelo ideal para o projeto desejado, de converter quadros do artista Monet em fotos de paisagem e vice-versa.

Neste escopo, onde a entrada dos modelos são imagens, é muito comum que o gerador da *GAN* seja um modelo do tipo *Encoder-Decoder* e o discriminador seja uma *CNN*.

2.1.1 *Encoder-Decoder*

Encoder-Decoders são modelos que seguem uma arquitetura definida por duas partes: uma que realiza a função de um *Encoder* e outra de um *Decoder*. Assim, é proposta uma divisão na qual o *Encoder* é responsável pela extração de características da entrada e o *Decoder* realiza uma reconstrução a partir dessa extração. Este tipo de modelo se destaca por conseguir construir uma boa representação da entrada em um vetor de característica através do *Encoder*, como se realizasse uma compressão. Além disso, ele

também se destaca por conseguir reconstruir uma imagem em cima das características extraídas, como uma descompressão, através do *Decoder*. Assim, *Encoder-Decoders* são muito interessantes para as aplicações de imagens, pois economizam memória com a etapa de *Encoder* e conseguem reconstruir a imagem, alterando-a conforme desejado, na etapa de *Decoder*.

Para o projeto estabelecido, a principal funcionalidade, de transformar o domínio de uma imagem de quadros do Monet para um domínio de Fotos de paisagem, pode ser obtida através da reconstrução feita pelo *Decoder*. Assim, os *Encoders-Decoders* se mostram ideal para a *GAN* que o grupo pretende implementar.

2.1.2. CNN - Redes Neurais Convolutivas

Redes neurais convolutivas, ou *Convolutional Neural Network*, são redes neurais usadas normalmente para analisar imagens. Elas adicionam camadas de convoluções nos dados de entrada antes deles percorrerem uma rede neural definida. Assim, os filtros aplicados nessas convoluções são aprendidos durante o treinamento, de forma que eles captam as características necessárias para a aplicação desejada. Com isso, as CNN conseguem, de forma dinâmica, identificar e classificar imagens através dos filtros.

Para o projeto definido pelo grupo, as redes convolutivas são uma ótima escolha de discriminadores para uma arquitetura GAN. Isto por conta de que, as redes convolutivas conseguem facilmente realizar a atividade de classificar uma imagem em pertencente (verdadeira) ou gerada (falsa) em relação a uma distribuição.

2.2 Cycle GAN - Modelo Escolhido

O modelo escolhido para realizar este projeto foi a *CycleGAN*, uma arquitetura específica de Redes Adversárias Generativas que tem como objetivo fazer uma tradução imagem-para-imagem *(image-to-image translation)*, ou seja, estabelecer uma função *G* capaz de mapear imagens de um certo domínio *X* para um outro domínio *Y* ($\mathrm{G}\colon \mathrm{X} \rightarrow \mathrm{Y}$), com o objetivo de torná-las indistinguíveis em comparação com imagens originais de *Y* por meio de uma função de perda adversária. Pode-se dizer que a *CycleGAN* é uma evolução do modelo *pix2pix*, o qual é capaz de realizar tal mapeamento, porém com a limitação de depender de um conjunto de dados pareado, isto é, para cada imagem pertencente a *X*, é necessário que exista uma imagem correspondente em *Y* para que o modelo consiga aprender. A arquitetura escolhida não possui essa limitação, sendo capaz de aprender um mapeamento que leva imagens de *X* para *Y* utilizando dados não pareados, como demonstrado na figura 1.

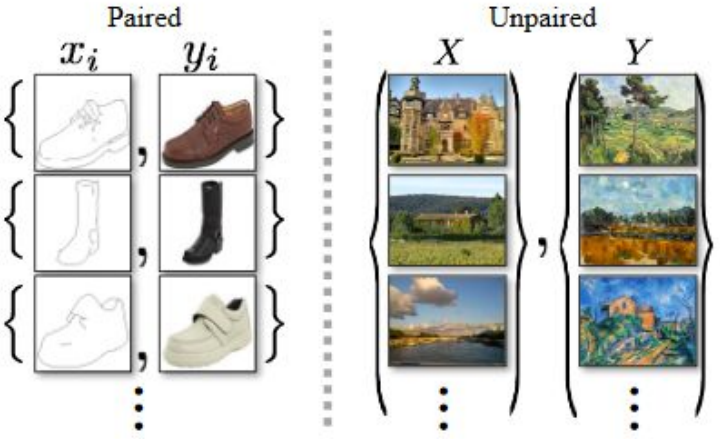


Figura 1: diferença entre um conjunto de dados pareado utilizado na *pix2pix* (esquerda) e um não pareado utilizado na CycleGAN (direita). Perceba que a primeira arquitetura depende dessa associação entre os dados para conseguir aprender, enquanto que a segunda é capaz de inferir um mapeamento entre os domínios sem tal necessidade.

Além disso, a *CycleGAN* realiza um mapeamento *F* inverso entre os domínios, sendo capaz de transformar imagens de *Y* para *X* ($\mathrm{F}: \mathrm{Y} \rightarrow \mathrm{X}$). Dessa forma, cria-se um ciclo e, com isso, é possível estabelecer uma função de custo denominada *cycle consistency loss*, a qual é responsável por induzir os resultados $\mathrm{F}(\mathrm{G}(\mathrm{X})) \approx \mathrm{X}$ e $\mathrm{G}(\mathrm{F}(\mathrm{Y})) \approx \mathrm{Y}$, possibilitando a melhora do desempenho do modelo.

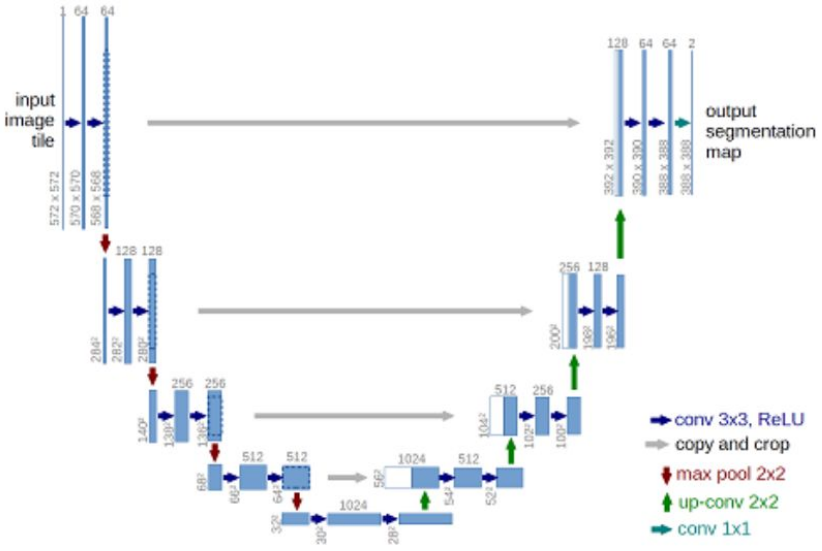
Ademais, existe uma outra forma de *loss function* utilizada no modelo *CycleGAN*. Para este método, convenciona-se que o resultado ideal para quando o discriminador lê uma imagem real de um certo domínio *X* seja uma matriz preenchida com o valor 1, e ao ler uma imagem de outro domínio que foi transformada para *X*, o discriminador retorne uma matriz de zeros. Nesse contexto, o discriminador possui como saída uma matriz denominada *patch matrix,* onde cada índice representa uma região da imagem. Os valores desses índices variam entre 0 e 1, onde 0 representa maior chance de determinada região ser falsa e 1 representa o oposto. Dessa forma, torna-se possível comparar as saídas do discriminador durante o treino com os resultados ideais esperados para cada caso.

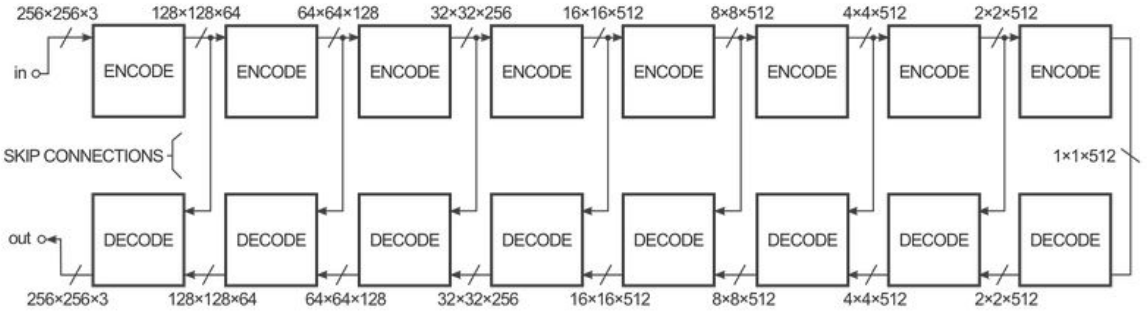
No caso deste projeto, o domínio *X* seria o de imagens reais e *Y* de pinturas do Monet. Devido à capacidade da arquitetura escolhida de mapear elementos de ambos os domínios entre si, é possível transformar tanto imagens reais em pinturas do Monet, quanto o inverso. Nesse contexto, a *CycleGAN* pode ser vista como duas *GAN*s, devido à presença de dois pares gerador-discriminador, um para cada domínio. A diferença é que, no momento do treinamento, esses componentes interagem entre si.

2.2.1 Generator - U-net

Para este projeto, decidiu-se utilizar o modelo U-net para realizar o papel de gerador. Essa arquitetura é uma variação do *Encoder-Decoder*, na qual existe uma recuperação de informação no momento do *upsample (decoder)*, que leva parte da saída obtida durante a etapa correspondente de codificação (*encoder)* e concatena com os dados a serem decodificados. Essa conexão entre as duas etapas é denominada *skip connection*, e a informação passada do *encoder* para o *decoder* é chamada de informação de contexto.

Essa última aumenta a qualidade dos resultados do *upsample*, visto que existe uma ligação com as imagens originais que foram codificadas e tiveram suas características extraídas, gerando resultados melhores no final.





Figuras 2 e 3: Dois diagramas mostrando a arquitetura de uma U-net.

Nesse contexto, definem-se as três funções de custo *(loss functions)* associadas ao gerador da *CycleGAN*. A primeira, citada anteriormente, é denominada *cycle consistency loss* e funciona da seguinte forma: é fornecida ao gerador $G: X \rightarrow Y$ uma imagem pertencente a *X*, resultando em uma imagem em *Y*. Em seguida, essa imagem gerada é fornecida ao gerador inverso $F: Y \rightarrow X$, resultando em uma segunda imagem gerada, desta vez pertencente ao domínio original *X*. Neste momento, calcula-se o erro absoluto médio entre a imagem original e a imagem "ciclada" (gerada por meio das operações descritas acima). Dessa forma, é possível checar se os geradores estão consistentes, visto que o resultado esperado deste ciclo é a própria imagem original ou uma imagem muito parecida.

A segunda função de *loss* utilizada no gerador é chamada de *identity loss*. Pode-se dizer que ela é responsável por confirmar a coerência dos geradores, e é calculada da seguinte maneira: para cada gerador, é fornecida uma imagem do domínio para o qual aquele gerador leva as imagens. Em outras palavras, alimenta-se o gerador $G: X \rightarrow Y$ com uma imagem pertencente a *Y* e, analogamente, ao gerador $F: Y \rightarrow X$ uma pertencente a X.

Por fim, calcula-se o erro médio absoluto entre a imagem fornecida e a gerada, esperando-se que seja próximo de zero, visto que teoricamente deveriam ser iguais.

Finalmente, a terceira função de custo empregada no gerador utiliza as matrizes citadas na seção anterior, que representam os resultados ideais do discriminador. Neste caso, o objetivo do *generator* é criar imagens de um certo domínio de forma precisa o suficiente para que elas sejam indistinguíveis das imagens reais daquele domínio. Logo, para alcançar tal resultado, a *patch matrix* obtida pelo discriminador ao ler a imagem gerada é comparada com a matriz ideal de uma imagem real. Este erro precisa ser minimizado, para induzir uma semelhança extrema entre a imagem criada e uma imagem original daquele domínio.

2.2.2 Discriminator

O discriminador utilizado neste projeto foi retirado da biblioteca *pix2pix*, desenvolvida como um módulo do *Tensorflow*, baseada no modelo *PatchGan*.

Para este componente da *CycleGAN*, calcula-se a função de *loss* utilizando as matrizes *patch* citadas anteriormente. Primeiramente, computa-se a diferença *(binary cross entropy)* entre uma imagem original de um dos domínios e uma matriz preenchida com o valor 1, que seria o resultado ideal dado pelo discriminador para esse caso. Em seguida, realiza-se o mesmo cálculo entre uma imagem criada pelo gerador para este certo domínio e uma matriz preenchida com zeros, que seria a saída ideal do discriminador para uma imagem falsa. Por fim, calcula-se a média entre esses dois resultados, obtendo-se a denominada *discriminator loss.*

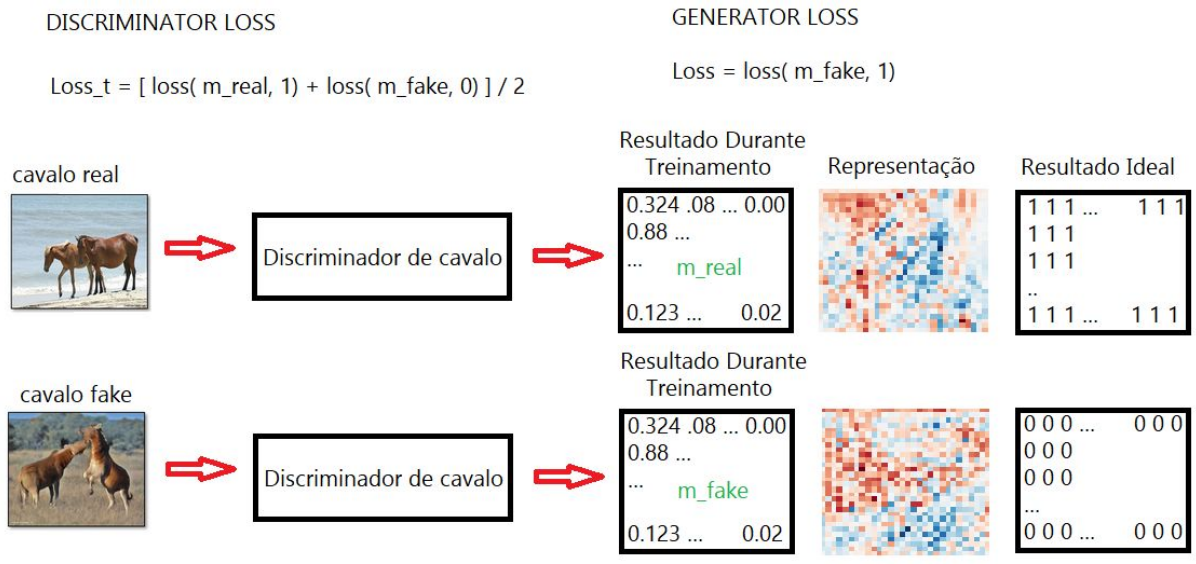


Figura 4: Esquema ilustrando como funciona a função de custo utilizando matrizes *patch*, onde o domínio em questão é de imagens de cavalos.

Nesse contexto, nota-se a importância desta função de custo para o treinamento do modelo. Ao minimizar o valor do resultado acima, o discriminador torna-se cada vez mais capaz de distinguir entre uma imagem real e uma falsa, fazendo com que o gerador precise, simultaneamente, aprender a criar imagens mais fiéis ao domínio original, com o objetivo de "enganar" o *discriminator*. Dessa forma, o modelo aprende como um todo (tanto a criar

imagens de um certo domínio, quanto a distinguir imagens reais de falsas) , apresentando resultados mais satisfatórios.

2.3 Dataset e Pré-Processamentos

Para o treinamento do modelo foram utilizados dois conjuntos de dados, obtidos na plataforma Kaggle. O primeiro conjunto com pinturas do Monet, contendo 1193 imagens, e o outro com fotos, contendo inicialmente cerca de 7000 imagens. Ambos conjuntos apresentavam imagens em uma resolução de 256x256 pixels e em 3 canais de cores, RGB.

Como ambos conjuntos continham imagens na mesma representação, nenhum processamento teve que ser feito para que os conjuntos fossem compatíveis. O único processamento necessário nesse aspecto, foi carregar os dados na memória através de tfrecords, uma estrutura específica da biblioteca *tensorflow*. Além disso, para que as imagens reais (conjunto de fotos) fossem coerentes com com o domínio de pinturas do Monet, foi feita uma seleção visual de imagens. Assim, foram selecionadas fotos que representam paisagens e cenários, removendo as que continham animais, pessoas em primeiro plano ou então fotos à noite com muita luz artificial em destaque. Com isso, o conjunto de fotos foi reduzido para um conjunto contendo apenas 1515 imagens. Por fim, ambos conjuntos foram divididos entre conjuntos de treinamento e conjuntos de validação. Para o conjunto do Monet, 956 imagens são utilizadas para treino e 237 imagens para validação. Já para o conjunto de fotos, 1216 imagens ficaram disponíveis para treino e 299 para validação.

2.4 Implementação e Treinamento do Modelo

A implementação do modelo foi feita utilizando o Google Collab. Essa escolha foi feita pois é fácil desenvolver código em grupo nessa plataforma. Além disso, o ambiente de execução é fornecido pelo Google. Assim, todo tempo e recurso necessário para treinar o modelo pode ser feito na Nuvem utilizando máquinas do Google. Algo muito significante, pois o modelo necessitou de 10 a 15 horas rodando em GPU para executar 120 épocas de treinamento. Note que leva de 10 a 15 horas, pois, dependendo do ambiente conectado, o tempo de treino de época pode variar. O grupo chegou a registrar de 300 a 530 segundos por época. Além disso, a divisão por células ajuda a organizar o código.

Após estabelecido um conjunto de dados para o treinamento do modelo, foi criado um processo de treinamento. Isto foi necessário tendo em vista que modelos generativos levam muito tempo para serem treinados. Assim, estabelecer um bloco de código (célula) para definir o treinamento facilitou muito, pois o mesmo poderia ter sua execução interrompida e continuada posteriormente. Este bloco é responsável por executar o treino por épocas, salvar o progresso do modelo, salvar exemplos de imagens produzidas pelo modelo e ajustar o passo de aprendizado do modelo.

Uma das principais funcionalidades do bloco é o salvamento do modelo conforme ele é treinado. Isto é possível através de checkpoints, onde um modelo pode ter seus pesos salvos em um arquivo .h5 e posteriormente carregados. Uma das maneiras de implementar isso é através de funções de callback para o treinamento, outra é através de salvamentos manuais. Ambos métodos foram utilizados para salvar o progresso de forma redundante. Com isso, e mais algumas variáveis de offset, é possível retornar o treinamento a partir da época que ele foi interrompido.

Mais uma função importante do bloco de treinamento é o ajuste do passo de aprendizado durante o treinamento. Com o *learning rate* sendo constante nas primeiras épocas e após uma quantidade determinada comece a diminuir linearmente. Isso é feito para que nas últimas épocas o passo seja menor. Assim, as últimas épocas apenas refinam a solução encontrada, fazendo um pequeno ajuste local no gradiente.

Outra decisão tomada para o bloco de treinamento foi a de salvar exemplos de imagens produzidas pelo modelo a cada quantidade determinada de épocas. Assim, são obtidas amostras tanto de quadros transformados em fotos quanto de fotos transformadas em quadros ao longo do treinamento. E com isso, é possível gerar gifs selecionando algumas imagens e quadros transformados salvos após o treinamento. Isso é importante para conseguir visualizar o aprendizado do modelo, já que um dos principais métodos de avaliação de GANs é a avaliação visual feita por humanos.

2.4 Avaliação

Quando se trata de GANs, não há um padrão fortemente estabelecido de métricas para ser utilizado. Isto ocorre por conta do propósito principal desta categoria de modelos, que é gerar novas amostras. Assim, quando o objetivo é gerar elementos que não existam posteriormente, não existe necessariamente um elemento alvo para ser utilizado de comparação, além dos elementos originais que compõem a própria distribuição. Assim, apenas métodos sofisticados que conseguem representar toda distribuição existente em um espaço dimensional e estimar a posição, e com isso a distância, do novo elemento gerado dessa distribuição são capazes de gerar métricas numéricas. Desta forma, é difícil estabelecer métricas que funcionem para aplicações diversas de GANs.

Um possível indicador sobre o aprendizado da GAN é o desempenho do seu discriminador. Caso o discriminador tenha uma acurácia igual ou pior ao resultado aleatório e não seja mais capaz de identificar qual elemento é falso ou verdadeiro, a rede, consequentemente, não conseguirá mais evoluir seu gerador. Assim, apesar desse indício parecer significar que a rede conseguiu gerar perfeitamente novas amostras, ele normalmente indica que a rede está com *overfitting*, ou seja, copiando os elementos existentes sem conseguir produzir amostras novas na distribuição. Esse problema é chamado de colapso da rede, ele ocorre quando o gerador gera sempre o mesmo elemento e o discriminador não consegue evoluir o suficiente para distinguir esse elemento ou suas pequenas variações. Desta forma, alcançar um colapso na rede é negativo na maioria das vezes.

Tendo em conta todos os aspectos já citados, o grupo acreditou que um bom método de avaliação para se considerar seria a própria capacidade humana de julgamento. Assim, foram registrados diversas imagens mostrando a evolução da rede em cima do conjunto de treinamento e com elas foram gerados gifs. As imagens finais do conjunto de treinamento junto de amostras do conjunto de teste foram usadas para uma avaliação visual, com isso foi possível determinar se o treinamento foi suficiente. As figuras 5 e 6, encontradas na seção de resultados, demonstram as amostras registradas.

2.4.1 Fréchet Inception Distance

A métrica numérica escolhida para avaliar o desempenho do modelo utilizado neste projeto foi a *Fréchet Inception Distance (FID)*, que foi desenvolvida especificamente com o objetivo de avaliar a qualidade das imagens criadas pelo gerador de uma GAN. Pode-se dizer que essa métrica é uma melhoria de outra medida inventada anteriormente, a *inception score*, que avalia apenas a distribuição das imagens geradas (artificiais), enquanto que a FID leva em conta também a distribuição das imagens reais usadas no treinamento do *generator*, realizando uma comparação entre os domínios das duas.

Nesse contexto, a FID recebe esse nome pois a comparação citada acima é realizada por meio da distância Fréchet, utilizada para calcular a similaridade entre duas curvas (no caso, as distribuições das imagens reais e geradas).

Para calcular essa métrica, foi utilizado a rede neural convolutiva Inception v3, na qual a última camada de *pooling* é usada para capturar mapas de características de um conjunto de imagens reais e outro de imagens geradas. Em seguida, calcula-se a média e covariância das saídas dessa camada, gerando duas distribuições Normais multivariadas (uma das imagens reais e outra das imagens geradas). Por fim, calcula-se a distância Fréchet entre as duas distribuições, resultando em um número real que representa a similaridade entre os dois conjuntos de imagens. O modelo possui um bom desempenho quanto menor for essa distância calculada, visto que uma distância de valor 0.0 indicaria que as duas distribuições são idênticas.

Para alcançar uma boa consistência estatística dos resultados, a *Fréchet Inception Distance* precisa de um número grande de imagens de entrada. Mais especificamente, estima-se que o número mínimo para um resultado confiável e significativo seja por volta de 5 mil instâncias. Contudo, para este projeto, não foi possível utilizar uma quantidade dessa magnitude para calcular a métrica, devido à limitação de memória RAM do Google Colab e ao acervo de pinturas do Monet, que não possui tantos quadros. Por isso, foram utilizadas as 1193 imagens de pinturas do Monet e 3166 imagens geradas pelo modelo para calcular a distância, o que pode influenciar na confiabilidade do valor obtido no final.

3. Resultados

A seguir, apresenta-se, visualmente, diversos resultados obtidos, assim como a evolução desses resultados ao longo das épocas de treinamento do modelo. Para melhor visualização e compreensão, dividiu-se os resultados entre os dois geradores do modelo.

3.1. Resultados do Gerador de fotos partindo de pinturas do Monet

Inicialmente, mostra-se a evolução do modelo em três épocas de treinamento diferentes. Observando essas imagens, pode-se notar os ajustes que o modelo realiza para transformar elementos da pintura em componentes realísticos na foto, mudando a textura e a cor deles. Dessa forma, o modelo se aproxima da distribuição das fotos reais e melhora o resultado das funções de *loss* durante o treinamento.

Figura 6: Coleção de Imagens que representam a evolução do Modelo gerador de paisagens realistas (fotos) durante o treino. Aqui se encontram os resultados apresentados pelo modelo em cima de uma mesma imagem com 3, 63 e 120 épocas de treinamento consecutivamente.

Em seguida, demonstra-se realmente os resultados finais deste gerador, separando o conjunto de imagens destinadas ao treinamento e o conjunto de teste, que não foi visto pelo modelo durante as etapas anteriores. Dessa forma, pode-se garantir a generalização do modelo, uma vez que ele produz resultados visualmente semelhantes em ambos os conjuntos.

Conjunto de treino:

Figura 7: Coleção de Imagens do conjunto de treinamento que representam o resultado do Modelo gerador de paisagens realistas (fotos) após o treino (120 épocas).

Conjunto de teste:

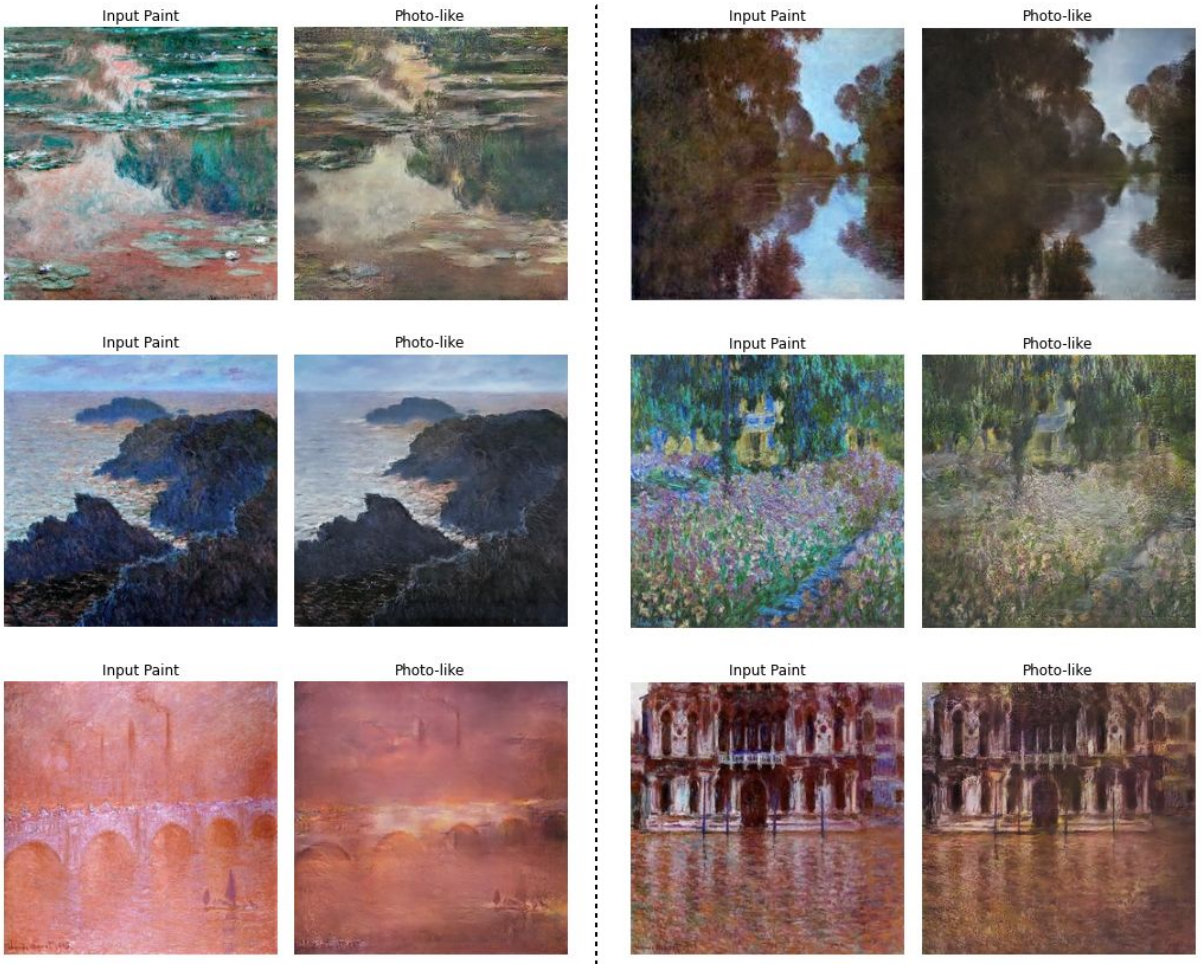


Figura 8: Coleção de Imagens do conjunto de teste que representam o resultado do Modelo gerador de paisagens realistas (fotos) após o treino (120 épocas).

3.2. Resultados do gerador de pinturas no estilo-Monet partindo de fotos

Da mesma forma do gerador anterior, foram estruturados os resultados do gerador de pinturas. As conclusões e análise são semelhantes, percebendo, visualmente, os ajustes e mudanças de textura e cor realizadas ao longo das épocas de treinamento.

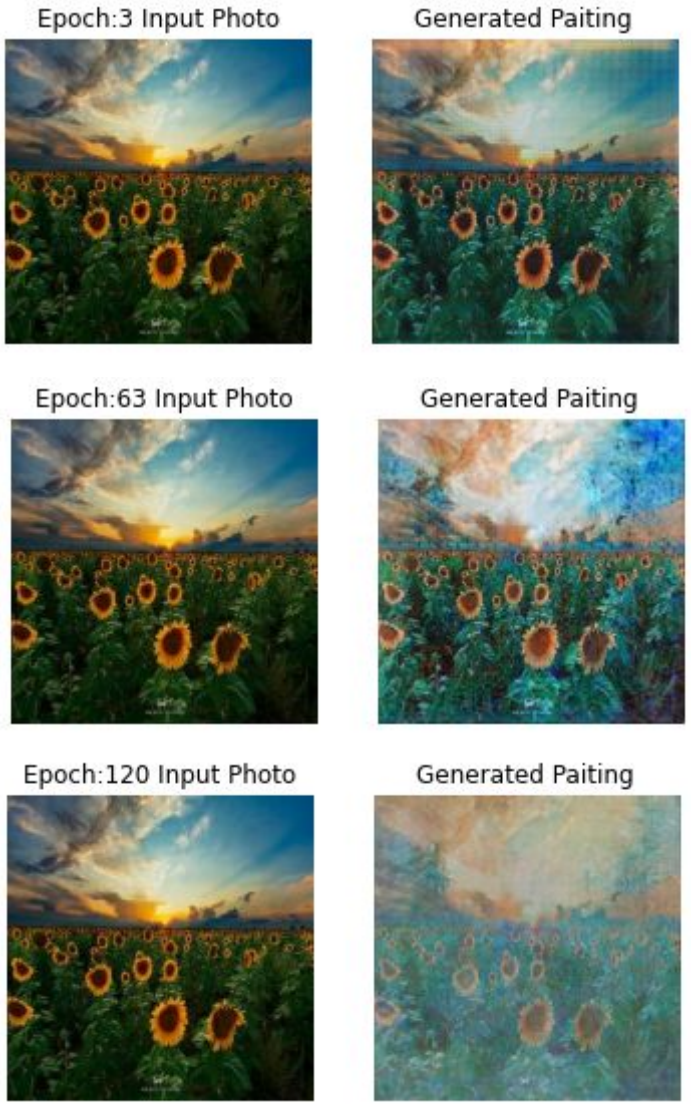


Figura 9: Coleção de Imagens que representam a evolução do Modelo gerador de pinturas do Monet durante o treino. Aqui se encontram os resultados apresentados pelo modelo em cima de uma mesma imagem com 3, 63 e 120 épocas de treinamento consecutivamente.

Novamente, são mostrados os resultados para os dois conjuntos de imagens, treino e teste, para garantir a generalização do modelo. A análise para esse gerador é ainda mais desafiadora, uma vez que vários fenômenos ocorrem nas imagens para que o gerador produza resultados próximos do domínio das pinturas. Contudo, os resultados foram considerados muito satisfatórios visualmente.

Conjunto de treino:

Figura 10: Coleção de Imagens do conjunto de treinamento que representam o resultado do Modelo gerador de pinturas do Monet após o treino (120 épocas).

Conjunto de teste:

Figura 11: Coleção de Imagens do conjunto de teste que representam o resultado do Modelo gerador de pinturas do Monet após o treino (120 épocas).

Por fim, apresenta-se a evolução da métrica FID durante o treinamento do gerador de pinturas no estilo Monet. É notável a redução do valor dessa métrica ao longo das épocas, apesar de flutuações nas últimas épocas. Esperava-se uma redução ainda maior, porém devido a pouca quantidade de imagens utilizadas para calcular a métrica, o resultado é bem satisfatório.

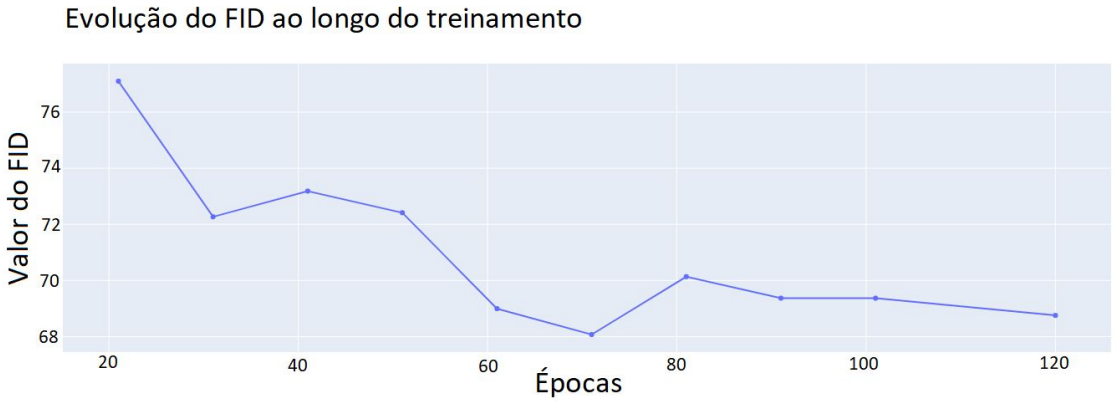


Figura 12: Evolução do valor da *Fréchet Inception Distance* para as 120 épocas, utilizando 3166 fotos não vistas pelo treinamento.

4. Conclusão

Este trabalho fez com que o grupo desenvolvesse um modelo de aprendizado de máquina, implementando todo o *pipeline* necessário, desde a coleta dos dados até a avaliação dos resultados. Por isso, foi um projeto desafiador, onde o grupo pesquisou e apresentou para a turma cada etapa. Dessa forma, o grupo pode se familiarizar com algumas técnicas de processamento dos dados, como a coleta utilizando o tipo "tf record" e repartição do conjunto de treinamento. Além disso, foi possível compreender modelos de redes neurais e *deep learning*, como *Encoder-Decoders,* classificadores de imagens e GANs. Também foram implementadas técnicas para o treinamento, como o passo de aprendizado variável, as funções de *loss* do modelo e os checkpoints. Por fim, foram estudados métodos para avaliações de GANs.

O grupo conseguiu desenvolver uma GAN e realizar seu treinamento de forma a alcançar resultados satisfatórios. Isso foi importante para que nós tivéssemos contato com modelos avançados de *deep learning*, desenvolvendo capacidades que podem ser aplicadas em projetos tanto mais simples, quanto até mais complexos. Os resultados obtidos através do modelo foram surpreendentes, na medida em que o modelo demonstrou conseguir realizar a transformação de domínio entre as imagens ao gerar exemplos com uma boa qualidade visual. Além disso, a redução da distância Fréchet ao longo das épocas oferece suporte a essa afirmação, através de um valor numérico calculado. Portanto, o grupo ficou bem satisfeito por conseguir implementar um modelo que, inicialmente, apresentou-se como um grande desafio na proposta do projeto. Tendo em vista que nós também aprendemos muito com o desenvolvimento realizado.

5. Referências bibliográficas

I'm Somethig of a Painter Myself. Kaggle, Disponível em: <https://www.kaggle.com/c/gan-getting-started/overview/description>. Acesso em: 26 de ago. de 2020.

Dimitri O. Monet paintings (Berkeley) - TFRecords 256x256. Kaggle. Disponível em: <https://www.kaggle.com/dimitreoliveira/monet-paintings-berkeley-tfrecords-256x256>. Acesso em: 30 ago. de 2020.

Obras de Claude Monet. Wikiart. Disponível em: <https://www.wikiart.org/pt/claude-monet>. Acesso em: 30 ago. de 2020.

Cycle-GAN. Git-Hub. Disponível em: <https://github.com/junyanz/CycleGAN>. Acesso em: 3 set. de 2020

J. Zhu, T. Park, P. Isola, A. A. Efros. *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*. Axiv, 24 de ago. de 2020. Disponível em: <https://arxiv.org/pdf/1703.10593.pdf>. Acesso em: 14 de set. de 2020.

J. Zhu. CycleGAN and pix2pix in PyTorch. Git-Hub. Disponível em: <https://github.com/junyanz/pytorch-CycleGAN-and-pix2pixhttps://affinelayer.com/pix2pix/>. Acesso em: 14 de set. de 2020.

Implementação da Unet. Github, Disponível em <https://github.com/zhixuhao/unet>. Acesso em 14 de set. de 2020.

Definição da avaliação em "*I'm Something of a Painter Myself*". Kaggle, Disponível em <https://www.kaggle.com/c/gan-getting-started/overview/evaluation>. Acesso em 11 nov. de 2020.

Implementação em Tensorflow da Distância Fréchet. Github, Disponível em <https://github.com/tsc2017/Frechet-Inception-Distance>. Acesso em 11 nov. de 2020.

Outra Implementação em Tensorflow da Distância Fréchet. Github, Disponível em <https://github.com/taki0112/GAN_Metrics-Tensorflow>. Acesso em 11 nov. de 2020.

Como implementar a Distância Fréchet. Machine Learning Mastery, Disponível em <https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/> . Acesso em 11 nov. de 2020.