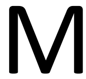


ESCOLA
SUPERIOR
DE MEDIA
ARTES
E DESIGN
POLITÉCNICO
DO PORTO

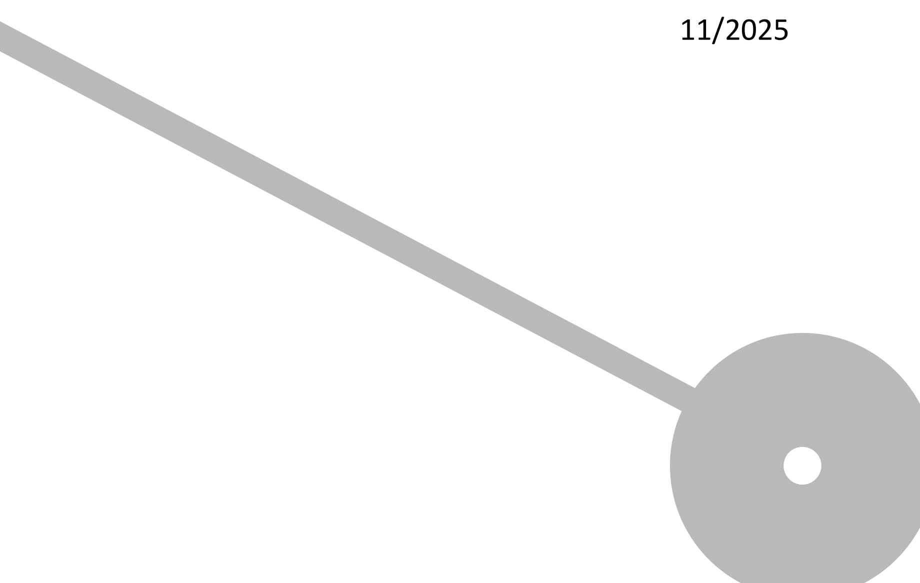
M

MESTRADO
MEDIA DIGITAIS INTERATIVOS

Gerador de Azulejos: Inteligência Artificial para a reprodução da estética patrimonial portuguesa

Gustavo Ventorim Glória Leal

11/2025



Politécnico do Porto
Escola Superior de Media Artes e Design

Gustavo Ventorim Glória Leal

Gerador de Azulejos: Inteligência Artificial para a reprodução da estética patrimonial portuguesa

Trabalho de Projeto

Mestrado em Media Digitais Interativos

Orientação: Prof. Manuel Jorge de Abreu Antunes Lima

Vila do Conde, novembro de 2025

Gustavo Ventorim Glória Leal

Gerador de Azulejos: Inteligência Artificial para a reprodução da estética patrimonial portuguesa

Trabalho de Projeto

Mestrado em Media Digitais Interativo

Membros do Júri

Presidente

Prof.ª Doutora Teresa Cristina de Souza Azevedo Terroso

Escola Superior de Media Artes e Design – Instituto Politécnico do Porto

Vogal - Orientador

Prof. Especialista Manuel Jorge Abreu Antunes Lima

Escola Superior de Media Artes e Design – Instituto Politécnico do Porto

Vogal - Arguente

Prof.ª Doutora Brígida Mônica Teixeira de Faria

Escola Superior de Saúde – Instituto Politécnico do Porto

Vila do Conde, novembro de 2025

AGRADECIMENTOS

Agradeço ao meu orientador, Professor Jorge Lima, pelo apoio e paciência, por ter acompanhado este projeto até o final, apesar dos percalços do caminho.

À Mariana, esposa, colega de mestrado e incentivadora para que eu concluísse essa tarefa. Sou grato ao seu amor e companheirismo, sempre!

Sou grato, também, a minha família. Minha mãe Rita, sempre confiante e obstinada em me motivar para a conclusão deste trabalho. Ao meu pai, que não pode ver o fim do projeto, mas que me apoiou deste sempre e ao meu irmão pela amizade.

RESUMO ANALÍTICO

Este trabalho tem por objetivo a criação de um *software* gerador de imagens utilizando Inteligência Artificial. O *output* esperado desta aplicação será um módulo inspirado nos padrões da azulejaria portuguesa. Para isso, será criado um conjunto de dados a partir de fotografias captadas da superfície das casas e edifícios portugueses. Esta informação será formatada e utilizada para treinar um algoritmo de inteligência artificial. Para chegar ao resultado, este projeto passa por uma explicação sobre o design de superfície, conceituando-o. Descreve a azulejaria portuguesa como fator de identidade do povo português e representante cultural de identidade, bem como seus padrões de estrutura e forma de arte. Por fim, destaca a inteligência artificial, com foco em redes neurais, suas técnicas de processamento, estruturas e algoritmos. Ressalta a importância do aprendizado de máquina e faz uma explicação sobre as redes adversárias generativas, técnica utilizada para a realização deste projeto. Na metodologia está descrito as etapas executadas para a criação do *dataset* e os passos necessários para o desenvolvimento da aplicação. A seguir é detalhado o desenvolvimento da aplicação e posteriormente uma apresentação dos resultados e análise, terminando com uma conclusão sobre o resultado gerado pelo *software*.

Palavras-chave: redes neurais artificiais; azulejo português; redes adversárias generativas; design de superfície; inteligência artificial.

ABSTRACT

This work aims to create an image-generating software, using Artificial Inteligence. The expected output of this application will be a module inspired by the patterns of Portuguese tilework. To achieve this, a dataset will be created from photographs of the surfaces of Portuguese houses and buildings. This information will be formatted and used to train an artificial intelligence algorithm. To reach the desired result, the project includes an explanation of surface design, providing its conceptual framework. It describes Portuguese tilework as a marker of Portuguese identity, cultural and artistic representative, as well as its structural patterns and artistic form. Finally, it highlights artificial intelligence, focusing on neural networks, their processing techniques, structures, and algorithms. It emphasizes the importance of machine learning and provides an explanation of generative adversarial networks, the technique used in this project. The methodology outlines the steps carried out to create the dataset and the necessary procedures for the application's development. Afterward, the development of the application is detailed, followed by a presentation of the results and analysis, concluding with a summary of the outcome generated by the software.

Keywords: artificial neural network; Portuguese tilework; generative adversarial networks; surface design; artificial intelligence.

SUMÁRIO

LISTA DE ILUSTRAÇÕES 9
LISTA DE SIGLAS 11

INTRODUÇÃO 12

1 REVISÃO DA LITERATURA 15
1.1 Design de Superfície 15
1.2 Azulejaria portuguesa 17
1.3 Inteligência Artificial 19
1.3.1 Aprendizado de máquina, redes neurais artificiais e *deep learning* 22
1.3.2 Redes neurais convolucionais 26
1.3.3 Redes adversárias generativas 28

2 METODOLOGIA 31
2.1 Coleta da amostra 31
2.2 Preparação das imagens 33
2.3 Criação do *dataset* 34
2.4 Estudos e testes de algoritmos 36
2.5 Treinamento da GAN 37
2.6 Implementação de API para "consumir" o modelo treinado 37

3 DESENVOLVIMENTO 38
3.1 Configuração 38
3.2 Implementação 40
3.2.1 Estrutura da aplicação 41
3.2.2 REDIMENSIONAR 44
3.2.3 TREINAR 47

3.2.3.1 Comparativo entre *dcgan.py* e *dcgan_keras3.py* 48
3.2.3.2 Etapa de treinamento 53
3.2.4 API 54

4 RESULTADOS **58**
4.1 Análise geral 58
4.2 Estabilidade da rede neural 62
4.3 Influência da quantidade de épocas 65
4.4 Comparativo empírico entre *dcgan.py* e *dcgan_keras3.py* 66

5 CONCLUSÃO **70**

REFERÊNCIAS BIBLIOGRÁFICAS **72**

LISTA DE ILUSTRAÇÕES

Figura 1 – Leão andante da Avenida Processional da Babilônia 16
Figura 2 – Exemplificação de módulos e seus respectivos padrões....................................... 17
Figura 3 – Tipologias para padrões de azulejos.. 19
Figura 4 – Diagrama adaptado, sobre as áreas de estudo e definições sobre Inteligência Artificial.. 21
Figura 5 – *Deep learning* em perspectiva as grandes áreas de estudo da inteligência artificial 23
Figura 6 – *Perceptron* em comparação com um neurônio biológico 25
Figura 7 – Representação de uma RNA. Imagem criada a partir do *website* mencionado na fonte. 26
Figura 8 – CNNs em perspectiva as grandes áreas de estudo da inteligência artificial 27
Figura 9 – Esquema representativo de convoluções sendo operadas em uma entrada de duas dimensões... 28
Figura 10 – A estrutura de uma GAN.. 29
Figura 11 – Linha do tempo com a evolução das GANs 30
Figura 12 – Fotografia de fachada, primeira abordagem utilizada na recolha de imagens para o *dataset* 32
Figura 13 – Fotografia de fachada, abordagem final utilizada na recolha de imagens para o *dataset* 32
Figura 14 – Fotografia coletada, sem tratamento.. 33
Figura 15 – Módulo de azulejo após tratamento de fotografia coletada 34
Figura 16 – Imagens rotacionadas para aumentar o *dataset* 35
Figura 17 – Exemplo de camadas de convolução.. 36
Figura 18 – Versão de driver Nvidia e CUDA 39
Figura 19 – Versão do WSL2 e distribuição Linux.. 40
Figura 20 – Hierarquia de pastas para a aplicação: gerador-azulejos.................................... 41
Figura 21 – Exemplo de hierarquia da pasta *resultado* 43
Figura 22 – Estrutura da classe FormatorImagem 45
Figura 23 – Construtor da classe FormatadorImagem 46

Figura 24 – Exemplo de uso do recurso REDIMENSIONAR 46
Figura 25 – Exemplo de uso do recurso TREINAR 48
Figura 26 – Função de preparação do *dataset* para classe DCGAN 50
Figura 27 – Gráfico da função sigmóide 50
Figura 28 – Gráfico de uma função tangente hiperbólica...51
Figura 29 – Função da classe DCGAN que realiza a construção do gerador 52
Figura 30 – Exemplo de uso do recurso API 54
Figura 31 – Trecho de código que usa o *framework* Flask e trata todas as questões relacionadas ao protocolo HTTP 55
Figura 32 – Trecho de código que recupera um arquivo aleatório de uma pasta ou utiliza o arquivo passado como parâmetro 56
Figura 33 – Trecho de código que apresenta a conversão de um Tensor em uma imagem PNG 57
Figura 34 – Trecho de código onde é feita a inclusão de um texto na imagem para identificar o modelo gerador 57
Figura 35 – Exemplo de módulo gerado por modelo treinado (20250609113406) 59
Figura 36 – Exemplo de módulo gerado por modelo treinado comparado com uma fotografia do *dataset* (20250716112722) 60
Figura 37 – Exemplos de módulos gerados em teste de treinamento (20250716135944).....61
Figura 38 – Comparação entre os resultados da função de perda de dois treinamentos.......63
Figura 39 – Comparativo entre os resultados de modelos estabilizados 64
Figura 40 – Comparativo de treinamento relativo a quantidade de épocas para a implementação *dcgan.py* 65
Figura 41 – Treinamento feito com Keras 3, exemplo de imagens e resultado de perdas do discriminador e gerador por épocas 67
Figura 42 - Treinamento feito com Keras 3, exemplo de imagens e resultado de perdas do discriminador e gerador por épocas. Parâmetro beta_1 = 0,5. 68
Figura 43 – Treinamento feito com Keras 3, exemplo de imagens e resultado de perdas do discriminador e gerador por épocas. Parâmetros iguais ao do treinamento feito com a classe do arquivo *dcgan.py*. 69

LISTA DE SIGLAS

CNN – *Convolutional Neural Network* (Rede Neural Convolucional)

CPU – *Central Processing Unit* (Unidade de processamento principal)

DL – *Deep Learning* (Aprendizado Profundo)

DS – Design de Superfície

GAN – *Generative Adversarial Network* (Rede Adversária Generativa)

GPU – *Graphics Processing Unit* (Unidade de processamento gráfico)

IA – Inteligência Artificial

IDE – Integrated Development Environment (Ambiente de Desenvolvimento Integrado)

ML – *Machine Learning* (Aprendizado de Máquina)

PEP – Python *Enhancement Proposals*

RNA – Rede Neural Artificial

WSL – Windows Subsystem for Linux

INTRODUÇÃO

Desde a disponibilização do ChatGPT[1] ao público em geral, em novembro de 2022 (Teixeira, 2023), o tema Inteligência Artificial (IA) está em voga. Junto com a popularidade desta tecnologia, revive-se também o temor relativo ao seu potencial negativo (Arntz et al., 2017). Porém, o certo é que as IA's vieram para ficar (Howley, 2023) e, assim como a prensa de Gutenberg, esta tecnologia tem a capacidade de revolucionar o desenvolvimento da Humanidade. É um tema complexo e que desafia o senso comum da população. Cria no imaginário das pessoas uma abstração próxima de distopias. Enxergam nesta tecnologia um futuro com máquinas sencientes e a superação da Humanidade pela máquina. Na opinião de Miguel Nicolelis (2023), prestigiado neurocientista, esta realidade jamais acontecerá e o termo Inteligência Artificial não passa de uma campanha de marketing.

Apesar do exposto, o fato é que esta ferramenta pode acelerar e aprimorar as diversas áreas do conhecimento. O uso de inteligência artificial pode estar nas recomendações de um filme em plataformas de *streaming* (Solà, 2022), reconhecimento facial (Conceição et al., 2020), diagnósticos de imagem na área médica (Conceição et al., 2020), categorização de processos judiciais (Maia Filho & Junquilho, 2018) etc. As possibilidades são infinitas.

Nas áreas criativas e artísticas não é diferente (Solà, 2022). A Inteligência Artificial desponta como recurso de grande impacto para artistas, designers e produtores de Mídias digitais. Seja na área de videojogos, modelagem 3D, instalações e outros tipos de expressões.

É neste contexto que este trabalho se baseia e pretende experimentar. A fim de enveredar nesta "nova realidade" em que a máquina é instrumento para geração de novos e improváveis resultados. Onde o volume de informação e a capacidade de processamento dos computadores permite produzir um novo *output*. A arte pode estar na forma da manipulação e na experimentação de uma massa de dados "crua".

[1] https://openai.com/blog/chatgpt

Questão de investigação

Em Mosaic Virus (Ridler, 2019), a artista Anna Ridler[2] cria um mural de telas que apresentam imagens de tulipas. Estas flores se modificam com o tempo, baseado na flutuação dos valores do Bitcoin e são geradas a partir de um programa de inteligência artificial.

Sua obra faz reflexões sobre o capitalismo, valor e um momento histórico marcantes da Holanda: a tulipomania, quando as vendas de tulipas no país tiveram variações de preços absurdas (Ridler, 2019). É um trabalho reflexivo que coloca em perspectiva uso da IA e desmistifica alguns dos procedimentos para a geração de imagens de forma automatizada pela máquina (Raley & Rhee, 2023). Para construção deste trabalho, Ridler fotografou e classificou tulipas para criar o *dataset* de treinamento do *software* utilizado em Mosaic Virus. Este primeiro trabalho foi exposto e intitulado Myriad (Tulips) (Ridler, 2018).

Esse conjunto de instalações feitas por Ridler consegue, de certa forma, apresentar ao público a complexidade necessária para construção de uma “Inteligência Artificial”, desvelando o trabalho por trás da construção destas ferramentas, tão populares atualmente. Além da crítica social, frente ao capital, consumo e sobre questões de uso dos conteúdos para o treinamento destas aplicações (Raley & Rhee, 2023), suas obras exemplificam de forma prática como as expressões artísticas e as Mídias digitais se cruzam e interrelacionam.

Com inspiração nestes dois trabalhos da artista e, considerando os aspectos culturais relativos ao qual estamos inseridos, ou seja, a cultura portuguesa, este projeto pretende desenvolver um *software* gerador de imagens, utilizando inteligência artificial, para reproduzir a estética da azulejaria portuguesa. Experimentará sobre as texturas de módulos dos azulejos. Esta experiência, visa compreender se os resultados obtidos a partir de uma reprodução feita pela máquina é satisfatória do ponto de vista estético e de semelhança.

Objetivos

O que se pretende com este projeto é codificar uma aplicação que, a partir de inteligência artificial, consiga gerar “novos” módulos que remetem aos da azulejaria

---

[2] https://annaridler.com

portuguesa. Para isso, serão necessárias fotografias dos azulejos a fim de exemplificar à "máquina" o que é um módulo de azulejo. Assim, constituir uma massa de dados que possa ser utilizada para o treinamento de uma rede adversária generativa. Para este projeto serão considerados apenas desenhos de azulejos que sejam modulares, aqueles que representam uma tela ou pintura não fazem parte do escopo.

Para que seja possível a implementação da solução, faz-se necessário abordar alguns temas que irão auxiliar na delineação e delimitação do desenvolvimento. São eles:

- Design de superfície: definição de padrões e módulos. Busca-se nesta área de conhecimento a delimitação de quais modelos de azulejo serão utilizados e que se pretende recriar através da inteligência artificial;
- Concepção da construção cultural da azulejaria portuguesa: compreender a construção do azulejo como elemento constituinte da identidade portuguesa e para ajudar na delimitação de escopo junto com o design de superfície;
- Inteligência Artificial e seus subcampos: aprofundar nos temas de ciência da computação relacionados diretamente com a implementação deste projeto. Necessário para compreensão de como deve ser feita a execução e codificação da solução proposta.

Estrutura do documento

Este documento está dividido em seis partes. Esta introdução que detalha os objetivos e contextualiza a proposta do projeto. O primeiro capítulo contém uma revisão de literatura que aborda os temas necessários e que embasam a construção deste projeto: já descrito nos objetivos desta introdução. O capítulo dois traz a metodologia, que descreve as tecnologias utilizadas e o processo de implementação da solução. A terceira parte trata do desenvolvimento e apresenta os detalhes de codificação, descrevendo de forma mais detalhada como se deu a implementação do projeto. O quinto capítulo apresenta os resultados com análise. Por fim, o último e sexto capítulo termina com uma conclusão.

1 REVISÃO DA LITERATURA

Nesta seção, serão explorados e elucidados os conceitos fundamentais utilizados neste projeto, além de fornecer uma contextualização sucinta dos domínios de conhecimento que orientam este trabalho.

O objeto central de análise é o azulejo, cuja compreensão requer uma apreciação do conceito de módulo, conforme delineado no campo do design de superfície. Posteriormente, destaca-se a relevância do azulejo como uma expressão artística significativa na cultura portuguesa.

Por fim, este estudo abordará a vertente tecnológica associada à Ciência da Computação, caracterizando a Inteligência Artificial e as Redes Neurais Artificiais. Serão explorados os subcampos destas, incluindo as Redes Neurais Convolucionais (Convolutional Neural Networks - CNNs), o Aprendizado de Máquina e Aprendizado Profundo (Machine Learning – ML e Deep Learning - DL) e as Redes Adversárias Generativas (Generative Adversarial Networks - GANs).

1.1 Design de Superfície

Para falar sobre design de superfície (DS), inicialmente, é imprescindível definir o que é Superfície. De forma geral a superfície é a face de alguma coisa, está relacionado geometricamente com o conceito de área, definido por comprimento e largura, mas também com a parte de fora dos objetos e com a aparência (Schwartz, 2008). Segundo o Dicionário Priberam, uma superfície é a “parte exterior e visível dos corpos” (Priberam, n.d.). Contudo, a Superfície, ao ser trazida para a materialidade, desempenha mais do que um simples revestimento e pode ser abordada segundo três principais temas. Primeiro, com o cunho representacional, envolve a sua geometria e representação gráfica. Depois de forma constitucional, quando se trata daquilo que a constitui, relaciona-se ao material e as técnicas para sua produção. Por último, o seu carácter relacional: diz respeito as relações entre o sujeito, o objeto e o meio, criando significado cultural, ergonômico, mercadológico etc. (Schwartz, 2008).

Conceituado o que é uma superfície, é possível perceber que o fato de decorar, pintar, revestir, entre outros processos, está presente na humanidade desde muito tempo. O

conjunto cerâmico da Avenida Processional da Babilônia (Figura 1), exposto no Museu Pergamon[3] em Berlin, data do século VII a.C. e já apresenta preocupação exemplar em ornamentar e significar as vias públicas.

Fonte: (Pergamon Museum, n.d.)

Figura 1 – Leão andante da Avenida Processional da Babilônia

Reinado do Rei Nabucodonosor II (604-562 a.C.)

Apesar da humanidade “manipular” a superfície há milhares de anos, o design de superfície como área de estudo tem em torno de 20 anos (Levinbook, 2008). Segundo Levinbook (2008), dentro do âmbito do DS, estudam-se diversos materiais como papel, vidro, pisos em geral, cerâmica e tecidos, apesar de o design têxtil despontar como principal esfera de estudo (Schwartz, 2008).

Mas então, o que é design de superfície? Segundo Evelise Anicet Rüthschilling:

> Design de Superfície é uma atividade técnica e criativa cujo objetivo é a criação de imagens bidimensionais (texturas visuais e tácteis), projetadas especificamente para o tratamento de superfícies, apresentando soluções estéticas e funcionais adequadas ao diferentes materiais e processos industriais. (Ruthschilling, 2002, p. 16)

No DS, a repetição de desenhos ou padrões é essencial, apesar de não ser uma obrigação. Este fato se dá pela necessidade, comum, de escalar e automatizar os processos de

[3] https://www.smb.museum/en/museums-institutions/pergamonmuseum/home/

confecção das superfícies. Além disso, a técnica de replicar pode ter resultados visuais interessantes, com grande possibilidade expressiva. Para esta técnica utiliza-se desenhos geométricos e modulares (Schwartz, 2008).

O módulo é a parte mais básica de um todo. "Cada módulo tem representado dentro de si todos os elementos do desenho organizados dentro de uma estrutura preestabelecida, de maneira que, quando colocados lado a lado umas das outras, formam um padrão contínuo" (Ruthschilling, 2002, p. 42). Ou seja, é a unidade do desenho que tem em si todos os dados básicos que, replicados e organizados, gera um padrão que gera um novo desenho ou textura (Schwartz, 2008). A Figura 2 exemplifica o conceito de módulo.

Fonte: (Schwartz, 2008)

Figura 2 – Exemplificação de módulos e seus respectivos padrões

1.2 Azulejaria portuguesa

Conforme definido por Mateus Miguel de Souza:

> Azulejo refere-se ao ladrilho cerâmico de superfície regular, quadrada ou poligonal com uma das faces decorada com esmaltes, destinado, por multiplicação, a ornamentar superfícies parietais ou pavimentares. A palavra deriva do árabe, *az-zullaiju*, que significa pedra lisa ou pedra polida. Alguns etimologistas também vinculam o vocábulo azulejo ao persa *lazawardou* mesmo *lápis-lazuliou*, ainda, a *zallaja*, que quer dizer liso ou escorregadio. (Souza, 2019, p. 32)

Ainda em seu texto, Souza (2019) salienta que antes da utilização da cerâmica na Península Ibérica, os azulejos já haviam sido usados largamente por outras civilizações (Figura 1), principalmente pelos árabes. Considerando o período de dominação muçulmana na região (Areán-García, 2009), pode ser que venha daí a influência dos azulejos na cultura portuguesa.

Susana Nunes, em sua obra "Azulejos de Padrão e Relevo – Uma Proposta Infográfica", considera o azulejo:

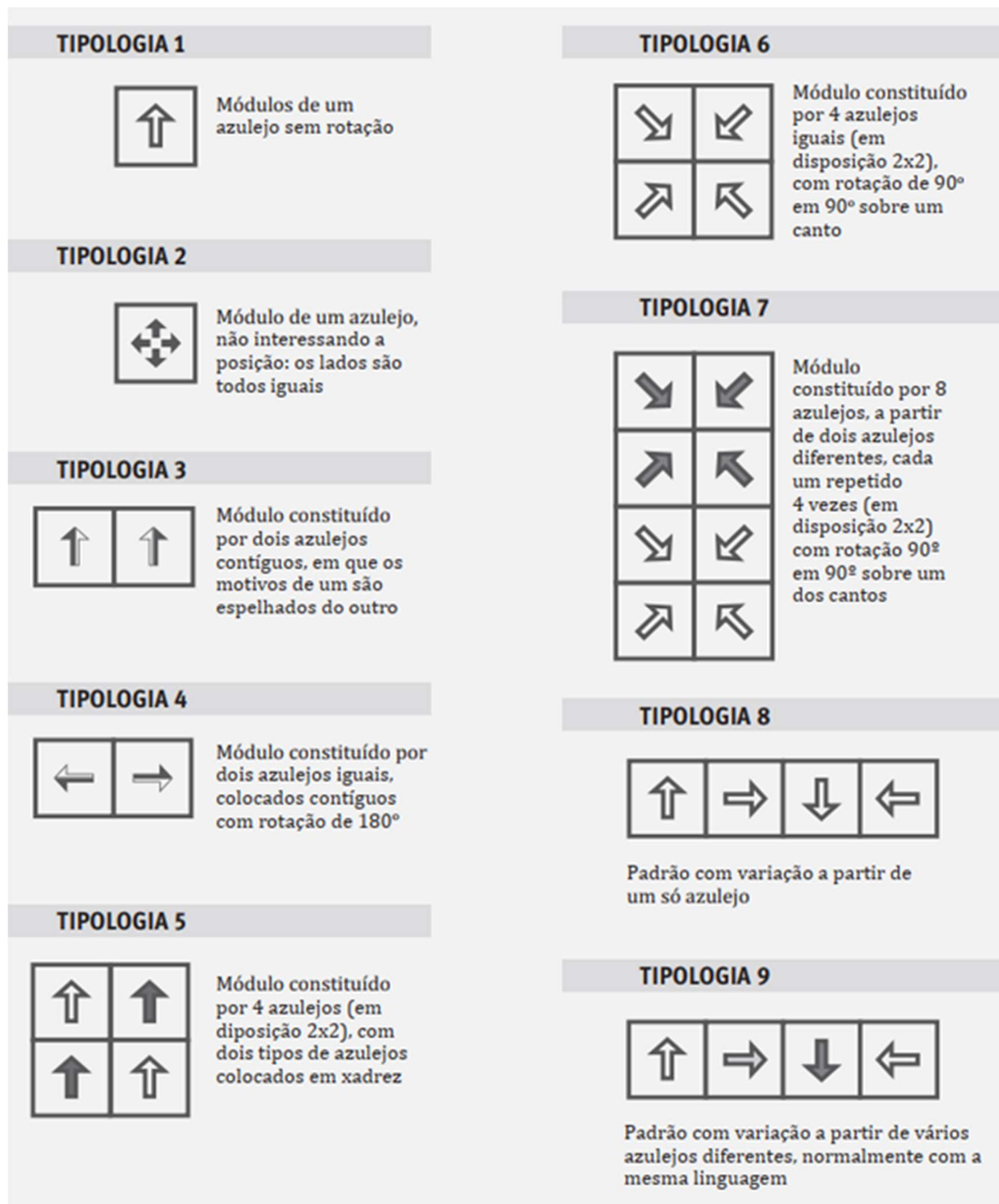
> Uma das manifestações mais significativas da cultura portuguesa. Em Portugal, o azulejo ultrapassou largamente a simples função utilitária ou o papel de arte ornamental, ascendendo ao estatuto de arte. Este tipo de revestimento cerâmico foi, desde o século XV, um componente determinante na arquitectura portuguesa, cobrindo paredes interiores e exteriores, jardins e telhados, estações de comboio e igrejas. (Nunes, 2014, p. 36)

Ainda sobre a identidade portuguesa relacionada ao azulejo, a autora cita como que o contato de Portugal com outros povos, principalmente fruto da expansão marítima, reflete-se na azulejaria. Seja pelo fato da incorporação de traços geométrico da cultura oriental ou pelas técnicas de fabrico da faiança italiana, os portugueses incorporaram diversas influências na arte do azulejo. Outro ponto a ser considerado é a relação mencionada de que Portugal se apropria ainda mais da azulejaria como forma de expressão artística, ao perceber como os demais países europeus muitas vezes "menosprezavam" esta expressão por não a considerar erudita (Nunes, 2014).

Em Portugal a criação e utilização de azulejos sempre estiveram divididas em duas concepções. A primeira utiliza a cerâmica como base para pintura, fazendo assim a criação de painéis figurativos, ignorando o formato do revestimento e tratando o revestimento apenas como tela. A segunda abordagem diz respeito ao azulejo de padrão, respeita a forma quadrangular e seu desenho, é pensado considerando a repetição que dá ritmo e forma para a superfície (Nunes, 2014)

O padrão será definido por uma composição decorativa regulada pela repetição de um módulo, gerada pelas possibilidades de posicionamento da peça base. É o desenho do azulejo, projetado para tal finalidade, que vai dar o ritmo e fluidez ornamental à superfície. Os padrões podem ser finitos ou infinitos, pois podem não encerrar em si próprio. Alguns padrões podem ter variações de cor e forma, sem, contudo, descaracterizar-se (Nunes, 2014).

Para este trabalho serão considerados apenas os azulejos passíveis de composição de forma a constituir padrões. A fim de ilustrar estas possibilidades, a Figura 3 apresenta algumas possibilidades de composição para a criação de uma superfície de azulejo conforme o exposto:



Fonte: (Nunes, 2014)

Figura 3 – Tipologias para padrões de azulejos

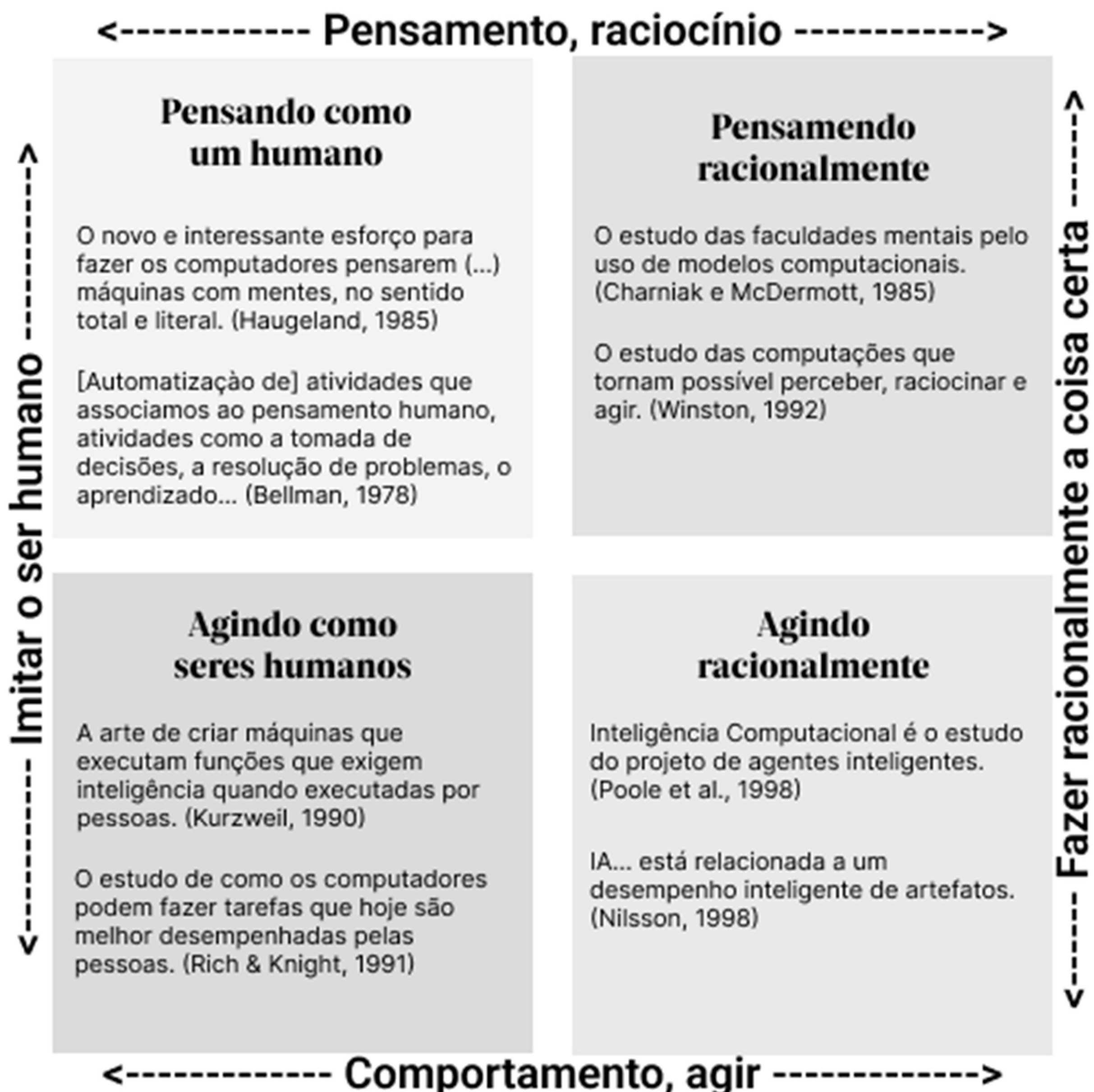
1.3 Inteligência Artificial

A inteligência artificial (IA) é um campo universal (Russell & Norvig, 2013), abrange diversas áreas do conhecimento e, por isso, pode ser um conceito difícil de delimitar.

A inteligência artificial como ideia, remonta a Grécia (Ribeiro & Matos, 2022). É claro que naquela época a noção do termo não era o que atualmente percebemos como IA, perpassava uma concepção filosófica que está atrelada ao próprio conceito de inteligência, conhecimento, cognição humana e a capacidade de raciocínio que existe de forma inata na espécie humana, mas que se manifesta, também, em outras entidades. Até mesmo a concepção de inteligência continua sendo algo amplo que não possui uma delimitação final (Legg & Hutter, 2007) e é estudada em campos diversos como a psicologia, a ciência cognitiva, neurociência etc.

Segundo Russell & Norvig (2013), a caracterização de Inteligência Artificial passa por quatro estratégias que foram pensadas, estudadas ou trabalhadas de alguma forma no decorrer da história. Dividem-se em iniciativas centradas em reproduzir o desempenho humano ou realizar atividades de forma racional, caracterizando o comportamento de um agente, tenta-se reproduzir uma conduta ou repetir um pensamento. Abaixo a Figura 4 apresenta um diagrama que resume de forma sucinta o exposto, com proposta de definições feitas por autores da área.

Neste trabalho o foco é na abordagem em que a máquina deva agir, seja como um ser humano ou de forma racional. O campo da IA se desenvolveu de forma satisfatória ao utilizar a estratégia comportamental aliada à matemática. Apesar de existir áreas de estudos que misturam a ciência cognitiva com modelos computacionais, essa ainda não é uma realidade. As principais ferramentas baseadas em IA operam na solução de problemas simulando o comportamento humano, mas não replicando o funcionamento biológico humano. Um exemplo interessante que ilustra a questão entre comportamento e replicação biológica é o ato de voar. As primeiras tentativas humanas de conseguir alçar voos foi tentando emular a forma como os pássaros faziam. Projetos eram feitos a simular o bater de asas. O “voo artificial” só aconteceu quando pararam de imitar as aves e começaram a usar túneis de vento e compreender a aerodinâmica (Russell & Norvig, 2013).



Fonte: (Russell & Norvig, 2013)

Figura 4 – Diagrama adaptado, sobre as áreas de estudo e definições sobre Inteligência Artificial

No início na década de 50, Alan Turing publicou o seu famoso artigo chamado "*Computing Machinery and Intelligency*" e atribui-se a ele as bases da IA até os dias de hoje. Contudo, o primeiro grande trabalho na área foi realizado por Warrem Macculloch e Walter Pitts (1943). Nesse trabalho eles se basearam em três fontes: o conhecimento fisiológico que já existia sobre o cérebro, a teoria da computação de Turing e a lógica proposicional de Russel e Whitehead. Foram esses pesquisadores que chegaram ao modelo de neurônio artificial caracterizado por "ligado" e "desligado", com o estímulo acontecendo a partir de outros

neurônios próximo, que simula a definição de estímulo da biologia e corresponde a definição de rede (Gomes, 2010).

De lá para cá, a história do campo de IA já teve momentos de sucesso, com ciclos de grande entusiasmo e otimismo, quedas, novas abordagens criativas e aperfeiçoamento sistemático das melhores estratégias. Um dessas, sem dúvida, foi a consolidação das Redes Neurais Artificiais (RNA), iniciada por Macculloch e Pitts. Apesar de existir diversas abordagens, no decorrer da história e nos últimos anos, o paradigma das RNAs, com o *deep learning*, pode ser considerado o padrão ouro dentro do campo da inteligência artificial e aprendizado de máquina (Alzubaidi et al., 2021). Este foi o paradigma "abraçado" pelas grandes empresas de tecnologia e que hoje baseia as ferramentas mais populares do mercado.

O avanço da IA que é visto atualmente só aconteceu, também, pela evolução dos componentes computacionais que proporcionaram um avanço na velocidade de processamento das máquinas. Esse fato permitiu que os computadores consigam processar volumes de informação nunca imaginados e, assim, permitir que as máquinas consigam ter tantos dados quanto o necessário para conseguir um resultado próximo ao de uma inteligência humana. Entram aqui, principalmente, a evolução das placas gráficas (*Graphics Processing Units* – GPU) que conseguem realizar mais operações em paralelo que os processadores principais dos computadores (*Central Processing Unit* – CPU) (Barata et al., 2021).

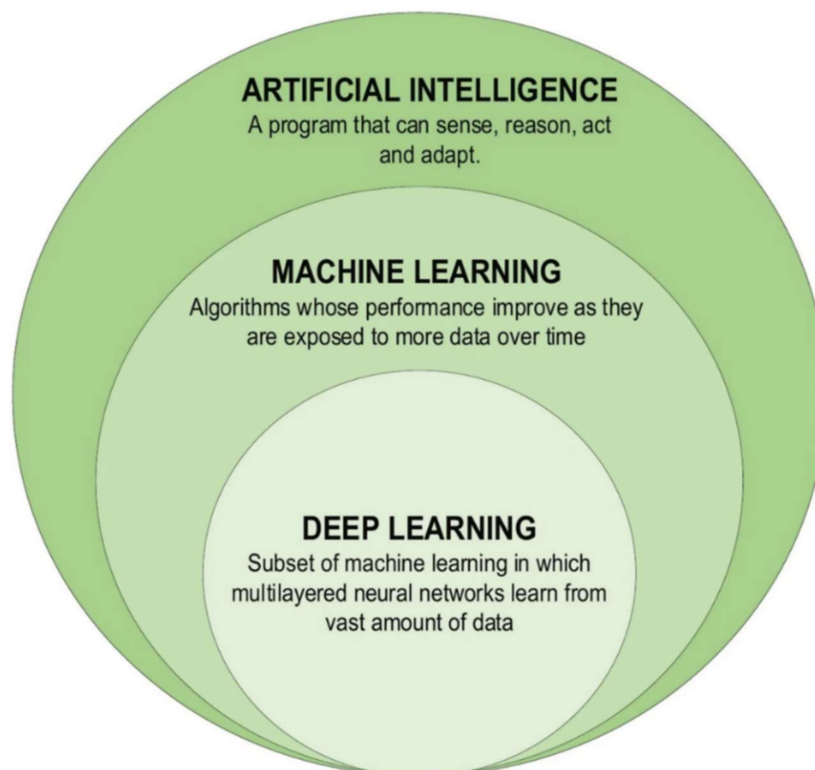
1.3.1 Aprendizado de máquina, redes neurais artificiais e *deep learning*

O principal método utilizado, atualmente, para a criação de uma IA perpassa a topologia de uma rede neural artificial (RNA) e a técnica de aprendizado de máquina (*machine learning* – ML). Conforme já mencionado, este é o modelo que se consagrou no mercado de tecnologia atualmente. Apesar de existirem algoritmos de ML que não utilizem RNAs (Chen et al., 2022) e vice-versa (Hancock & Khoshgoftaar, 2020), a maioria dos que existem hoje, utilizam a combinação dessas duas técnicas de forma basilar para a resolução de problemas. O fato de estarem tão intrinsecamente ligados faz com que seja difícil desassociar ML e RNAs e na maior parte das vezes falar sobre um tema é estar indiretamente falando sobre o outro. Mesmo os modelos mais recentes, como os LLM (*large language models*) ou as redes

generativas adversárias, e que ganham o mercado, são todos fundamentados por redes neurais e aprendizado de máquina.

O aprendizado de máquina é um campo de estudo que tem como premissa ensinar computadores, ou seja, colocar a máquina a aprender, a executar uma ação sem uma programação explícita que descreva seu comportamento. É uma forma de ensinar a partir do exemplo, onde estes exemplos nada mais são do que dados. Ensina a máquina a lidar com a informação de forma mais eficiente, detectando padrões a partir de modelos matemáticos (Mahesh, 2019). Nesta abordagem, o volume de dados pode e irá influenciar o resultado, onde, no geral, quantidades abundantes de informação produzem melhores respostas (IBM, n.d.). Não há apenas uma solução de ML, e sim, uma variedade de algoritmos que, a depender do problema, deverá ser usado.

> Cientistas de dados gostam de destacar que não existe um único tipo de algoritmo universal que seja o melhor para resolver um problema. O tipo de algoritmo empregado depende do tipo de problema que você deseja resolver, do número de variáveis, do tipo de modelo que melhor se adequaria a ele e assim por diante. (Mahesh, 2019, p. 381, tradução própria)



Fonte: (Alzubaidi et al., 2021, p. 7)

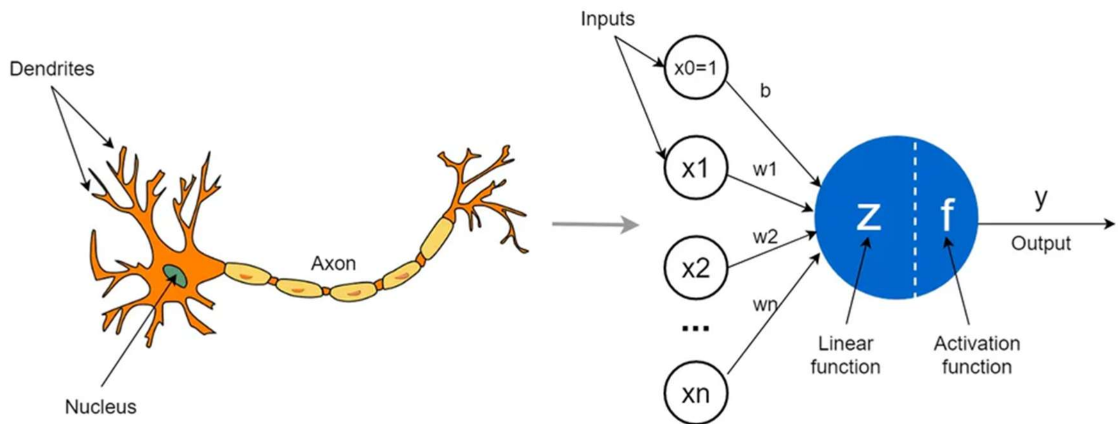
Figura 5 – *Deep learning* em perspectiva as grandes áreas de estudo da inteligência artificial

O aprendizado de máquina pode exigir algum pré-processamento, seja em classificar, organizar ou preparar os dados que serão utilizados no treinamento algorítmico. Essa etapa do processo, além de demandar muito esforço pode incluir viesses na medida que a seleção feita pode discriminar alguma informação relevante e que deixa de ser "percebida" pelo computador. O *deep learning* (DL) surge como um subconjunto (Figura 5) dentro do campo de ML que tem a capacidade de automatizar o aprendizado de conjuntos de características para diversas tarefas, ao contrário dos métodos convencionais de ML. O DL permite que o aprendizado e a classificação sejam realizados de forma integrada, em uma única etapa (Alzubaidi et al., 2021). Contudo é importante ressaltar que o DL pode ser feito, também, com dados pré-processados. A diferença crucial entre algoritmos de ML tradicionais e DL está diretamente ligado a quantidade de camadas utilizadas para o treinamento (Chen et al., 2022; IBM, n.d.), enquanto o primeiro normalmente possui poucas posições o segundo tem camadas suficientes que desafiam a abstração da compreensão humana.

Enquanto o aprendizado de máquina caracteriza a forma como a computação acontece, as redes neurais artificiais determinam a estrutura de como os algoritmos serão construídos (IBM, n.d.).

Como já mencionado, as RNAs têm esse nome porque foram inspiradas no comportamento dos neurônios humanos. Uma RNA "tem duas facetas elementares: a arquitetura e o algoritmo de aprendizagem. Essa divisão surge naturalmente pelo paradigma como a rede é treinada" (Rauber, 2005, p. 6). Diferente de uma programação convencional a RNA é treinada a partir de exemplos e o algoritmo de aprendizagem tem "liberdade" para definir o tipo de rede e adaptar os pesos para cada caminho que mapeia a tomada de decisão (Rauber, 2005).

O modelo de neurônio artificial de 1943, foi mais tarde aperfeiçoado por Frank Rosenblatt em 1958 e passou a ser conhecido como *perceptron* (Barata et al., 2021). Esta foi a gênesis deste tipo de arquitetura. Na Figura 6 tem-se de forma esquemática a ideia por trás do *perceptron,* comparando-o com um neurônio "real":

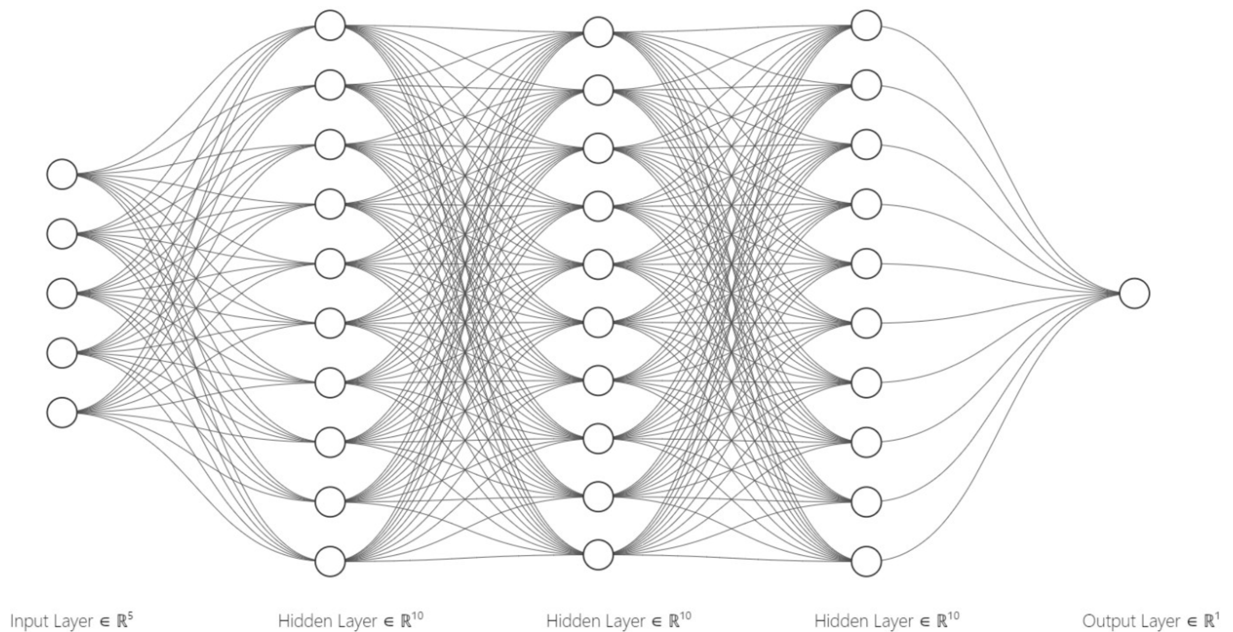


Fonte: (Solanki, 2024)

Figura 6 – *Perceptron* em comparação com um neurônio biológico

De forma geral, o *perceptron* (Figura 6) é um modelo matemático que recebe entradas numéricas, multiplica cada uma delas por pesos associados (que determinam a importância relativa de cada entrada), soma esses produtos (podendo incluir um viés, que é uma constante para ajustar a saída do modelo) e passa o resultado por uma função de ativação. A função de ativação, no *perceptron* clássico, é geralmente uma função de passo que decide se a saída será um valor como 0 ou 1 (classificação binária). Isso faz com que o *perceptron* filtre as entradas para produzir uma saída válida dentro de um conjunto pré-definido de respostas (Haykin, 2001). As RNAs são constituídas por camadas de *perceptrons* que computam suas entradas e alimentam a próxima camada com os seus resultados e assim sucessivamente, encadeando diversas operações até obter um resultado final.

A Figura 7, exemplifica a topologia de uma rede neural, onde cada círculo representa uma entrada/saída ou um *perceptron* e suas conexões com os demais elementos da rede.



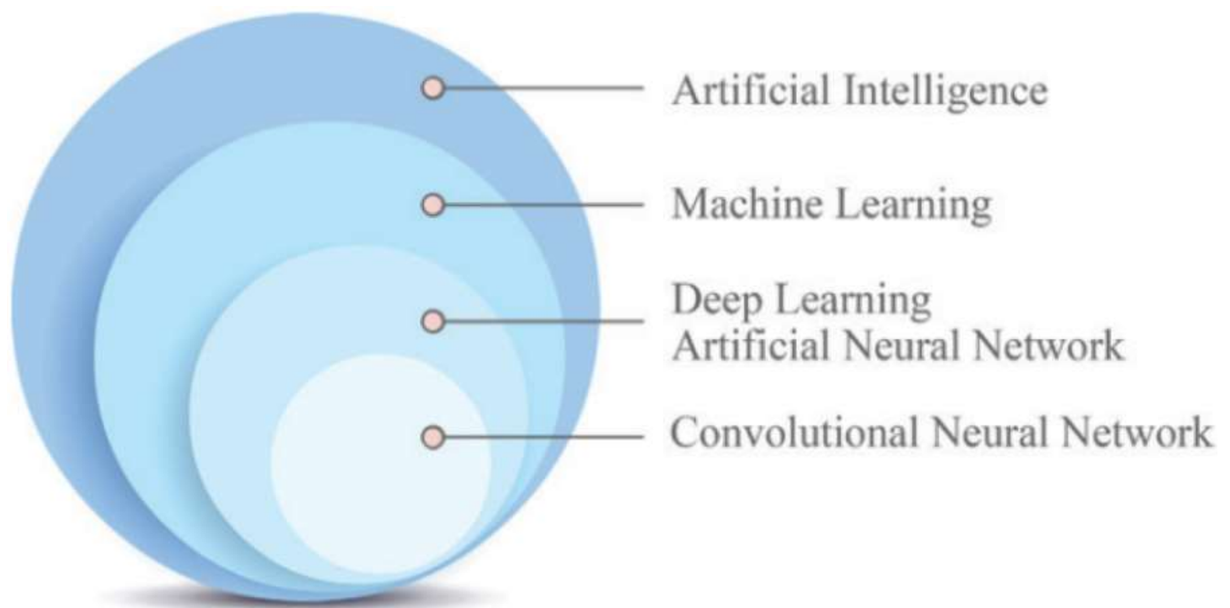
Fonte: https://alexlenail.me/NN-SVG/

Figura 7 – Representação de uma RNA. Imagem criada a partir do *website* mencionado na fonte.

No caso das redes neurais artificiais, a "aprendizagem" ocorre por meio do ajuste iterativo dos pesos das conexões entre os nós (neurônios). Esse processo utiliza algoritmos de otimização, como o gradiente descendente, e é guiado por uma função de perda que mede o quão distante a saída da rede está do valor esperado. Durante o treinamento, os pesos são recalculados de forma a minimizar essa perda, o que ocorre através da propagação do erro (*backpropagation*), que distribui os ajustes necessários pelos diferentes nós da rede.

1.3.2 Redes neurais convolucionais

As redes neurais convolucionais (convolutional neural network – CNN) são redes neurais de aprendizado profundo (Figura 8) aprimoradas, usadas comumente para tratar dados que estejam dispostos em forma de grade (matriz), como imagens e vídeos. Resolvem algumas questões de uma rede neural convencional: como o aumento excessivo de parâmetros, a identificação de padrões repetitivos dentro do conjunto de entradas e questões de espacialidade das repetições destes padrões (Ankile et al., 2020).

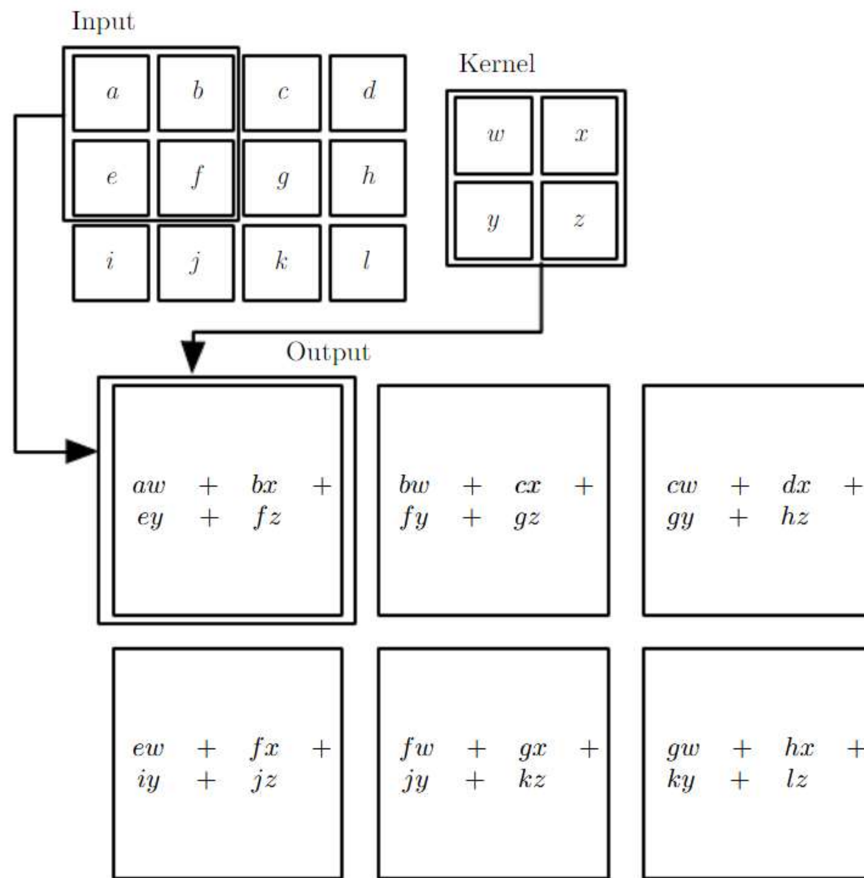


Fonte: (Ankile et al., 2020, p. 1)

Figura 8 – CNNs em perspectiva as grandes áreas de estudo da inteligência artificial

Este tipo de redes neural, diferentemente das redes neurais tradicionais (como as redes totalmente conectadas), não conectam todos os nós entre si. Elas utilizam operações de convolução para detectar padrões locais nos dados, como bordas, texturas e formas, especialmente em imagens. Esse mecanismo permite que os pesos (filtros) sejam compartilhados em diferentes regiões dos dados de entrada, reduzindo significativamente o número de parâmetros. Além disso, ao reutilizar esses filtros para detectar padrões em diferentes partes da entrada, a computação é otimizada, resultando em menor custo computacional em comparação às redes totalmente conectadas. São estas operações matemáticas chamadas convolução que dão nome a este tipo de topologia de rede (I. Goodfellow et al., 2016).

A Figura 9 ilustra de forma simplificada o processo de convolução. O filtro (*kernel*) vai percorrer a entrada, operando sobre uma parte apenas dos dados. A cada iteração o filtro é deslocado uma posição para direita e posteriormente para baixo. Neste caso o *kernel* não ultrapassa os limites da entrada e realiza dois movimentos para a direita e apenas um para baixo.



Fonte: (I. Goodfellow et al., 2016, p. 330)

Figura 9 – Esquema representativo de convoluções sendo operadas em uma entrada de duas dimensões

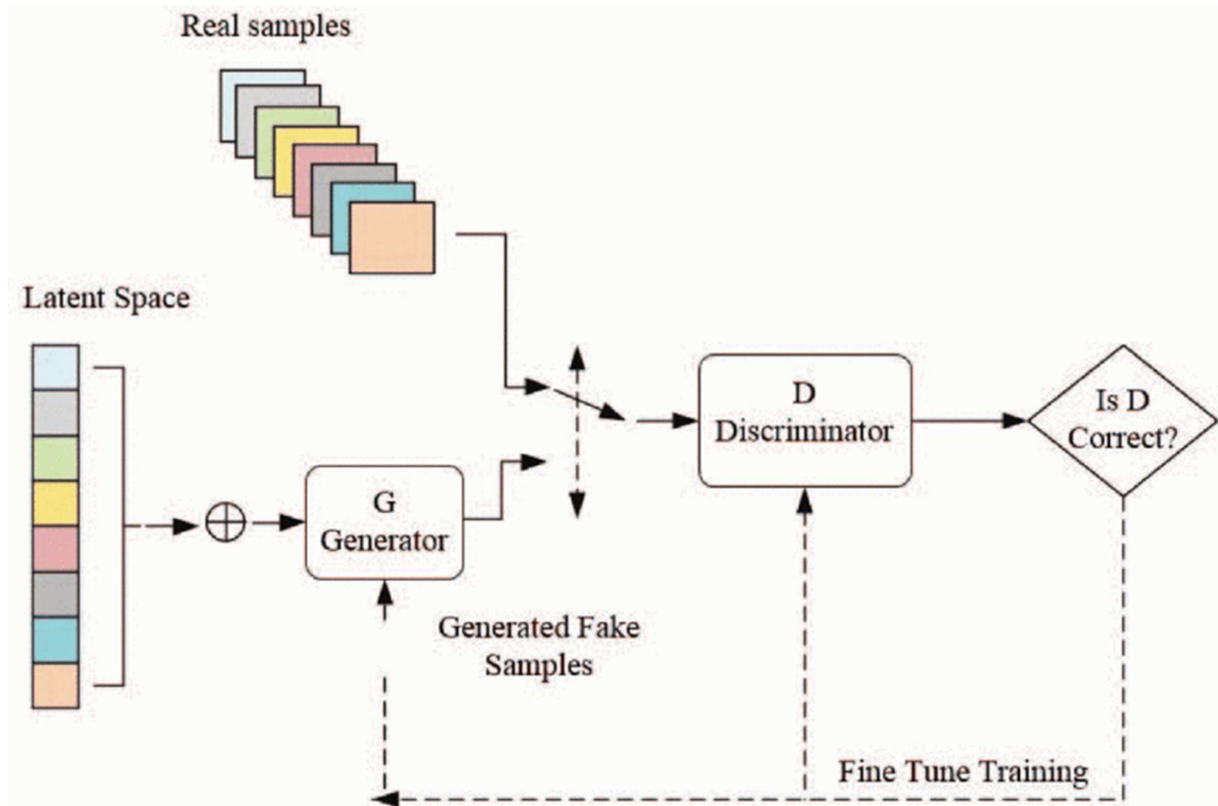
1.3.3 Redes adversárias generativas

As redes adversárias generativas (generative adversarial networks/nets – GAN) foram introduzidas em um artigo de 2014, feito por pesquisadores da Universidade de Montreal (Goodfellow et al., 2014). São estruturas de *deep learning* que foram pensadas para gerar informações artificiais que se assemelham de forma convincente aos dados do mundo real (Chakraborty et al., 2023).

As GANs utilizam um processo adversário, em que são treinados dois modelos simultaneamente. Um modelo gerador G, que captura a distribuição dos dados, e outro discriminador D, que estima a probabilidade de uma amostra ter vindo do conjunto de dados de treinamento em vez do modelo G. Ou seja, o objetivo de G é aprender e capturar a distribuição subjacente dos dados reais tanto quanto possível, gerando novas amostras de dados que se assemelhem a estas amostras. Por outro lado, D é um classificador binário cujo

objetivo é determinar se os dados de entrada são provenientes do conjunto de dados reais ou fictício, criado pelo gerador. O procedimento de treinamento para o gerador consiste em maximizar a probabilidade de o discriminador cometer um erro. Essa estrutura pode ser interpretada como um jogo “minimax” entre dois jogadores (Chakraborty et al., 2023; Gonog & Zhou, 2019).

A Figura 10 mostra de forma esquemática o conceito descrito.



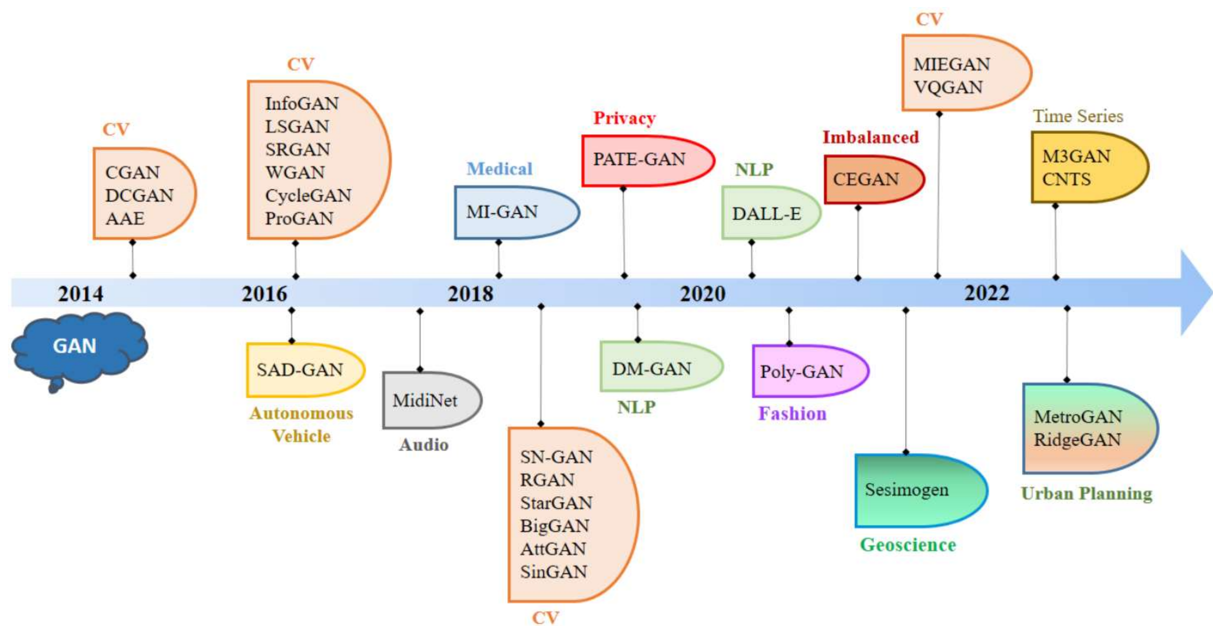
Fonte: (Gonog & Zhou, 2019, p. 505)

Figura 10 – A estrutura de uma GAN

Este modelo de GAN obteve muito sucesso no passar dos anos e em fevereiro de 2018, conquistou o primeiro lugar na “Lista das Dez Maiores Tecnologias Disruptivas Globais” emitida pela Massachusetts Science and Technology. Um dos campos em que foi muito explorado é o da geração de imagens. Apesar das GANs serem eficazes para gerar uma gama variada de tipos de dados, seus estudos iniciais, que popularizaram o tema, estão relacionados com a produção de imagens realistas (Chakraborty et al., 2023).

Depois do primeiro trabalho que projetou este tipo de abordagem inúmeras foram as aplicações e evoluções das GANs. Há, atualmente, uma quantidade significante de

variedades de GANs e que podem ser aplicadas a situações específicas como aumento de resolução de imagens, geração a partir de texto, preenchimento de imagens, aplicações em áreas da medicina, na área de veículos autônomos etc. A Figura 11 ilustra uma linha do tempo com a evolução e as diversas implementações que foram sendo criadas com o passar dos anos.



Fonte: (Chakraborty et al., 2023, p. 6)

Figura 11 – Linha do tempo com a evolução das GANs

As implementações das redes adversárias, em sua maioria, vão utilizar as arquiteturas de redes neurais, em particular as convolucionais (principalmente quando se tratar de criações visuais) e aproveitam, também, técnicas de *deep learning.*

Neste capítulo tratou-se de delimitar o conceito de módulo de um azulejo. Apresentou a importância da azulejaria na construção histórico-cultural de Portugal. Dissertou, também, sobre o tema da Inteligência Artificial, enfatizando as técnicas e estruturas que serão utilizadas neste projeto.

2 METODOLOGIA

Conforme descrito anteriormente, este projeto tem por objetivo o treinamento de uma GAN, para experimentar a possibilidade da criação de módulos de azulejos que utilizam das técnicas de rede neural, *deep learning* e o novo ferramental relacionado a inteligência artificial que temos disponível no momento. Para alcançar o exposto foi necessário seguir algumas etapas que serão descritas a seguir.

2.1 Coleta da amostra

Como primeiro passo, foi necessário coletar imagens para produção do *dataset,* que servirá como base de treinamento para o modelo de GAN escolhido. Nesta fase, a recolha de imagens se deu de forma aleatória sem nenhum critério específico. A tomada das fotografias sucedeu-se durante atividades do cotidiano e não foi determinadas rotas ou locais específicos para tal.

Baseou-se principalmente na região do Concelho do Porto, Matosinhos e de Vila Nova de Gaia, mas não exclusivamente. Há algumas recolhas feitas em outros concelhos, como Lisboa, Aveiro e Évora.

Foi utilizado, para a maioria das fotografias, um telemóvel Samgung Galaxy S10e[4]. Em um primeiro momento as fotos tiradas incluíam parte da fachada, mas no decorrer do processo, notou-se que a recolha seria mais eficaz se centrasse apenas no módulo, em um azulejo. A segunda abordagem só não pode ser cumprida quando a distância entre a fachada e o ponto de vista fosse demasiadamente grande. Nestes casos, é possível que tenha sido feita uma tomada de parte considerável da fachada e a utilização de zoom. A diferença de abordagem inicial e final pode ser visualizada nas Figura 12 e Figura 13, onde a primeira representa a abordagem inicial e a segunda a recolha a partir do módulo.

[4] https://www.gsmarena.com/samsung_galaxy_s10e-9537.php

Fonte: Imagem criada pelo autor

Figura 12 – Fotografia de fachada, primeira abordagem utilizada na recolha de imagens para o *dataset*

Fonte: Imagem criado pelo autor

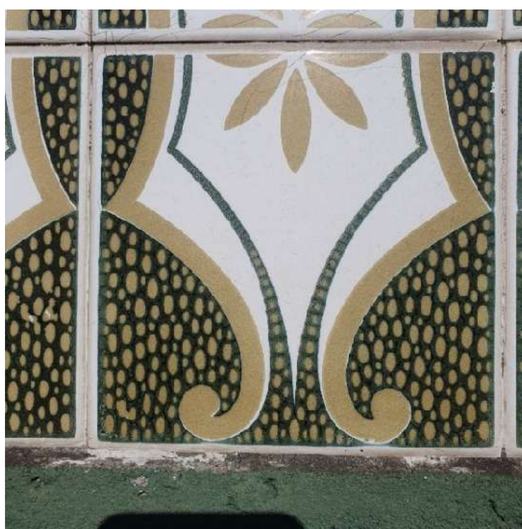
Figura 13 – Fotografia de fachada, abordagem final utilizada na recolha de imagens para o *dataset*

Houve contribuição de outras pessoas na recolha das fotos. Nestes casos os critérios descritos anteriormente não foram seguidos.

Nesta fase foram coletadas um total de 511 fotos que resultou em 522 módulos de azulejo. Como algumas fotografias possuíam parte de fachadas, foi possível recortar mais de um módulo em algumas das imagens. O processo de regularização das figuras está descrito na próxima sessão desta metodologia.

2.2 Preparação das imagens

Foi utilizada a ferramenta GIMP[5], em sua versão 2.10.38, para a regularização das imagens de azulejos. O processo de normalização consistiu em apenas duas modificações básicas: um recorte da imagem (Tools > Transform tools > Crop) para delimitá-la, conforme a borda de um módulo e um melhoramento do angulo de perspectiva do módulo, na tentativa de deixar a imagem "plana" (Tools > Transform tools > Perspective). Neste processo, usualmente, foi feito primeiro uma ação de "*crop*", na sequência uma ação de "*perspective*" e por fim mais um "*crop*". As Figura 14 e Figura 15 demonstram o estado inicial da fotografia e o resultado após tratamento.



Fonte: Imagem criado pelo autor

Figura 14 – Fotografia coletada, sem tratamento

[5] https://www.gimp.org

Fonte: Imagem criado pelo autor

Figura 15 – Módulo de azulejo após tratamento de fotografia coletada

Nenhum outro tratamento de imagem foi feito para a normalização. Questões de luminosidade, brilho, contraste, definição e outras imperfeições foram propositalmente mantidas para observar o comportamento do modelo ao ser treinado.

2.3 Criação do *dataset*

Ao realizar o treinamento da rede neural é esperado que as imagens tenham sempre o mesmo tamanho entre si. Outro ponto importante é reduzir a resolução das imagens, a fim de melhorar o tempo de processamento, uma vez que que será usado um computador “doméstico” padrão. Assim, para criar o banco de figuras, faz-se necessário converter as imagens originais, reduzindo o seu tamanho e corrigindo a proporção de pixels para cada fotografia que não contenham a altura e largura esperada para o treinamento. Neste procedimento perdem-se pixels, o que reduz a qualidade da imagem, mas é necessário para a experimentação.

Como algumas imagens não são quadradas, optou-se por redimensioná-las e criar uma pequena distorção, deixando as imagens com as mesmas dimensões em cada um de seus lados. Outra forma para transformar as fotos sem distorcê-las seria adicionar margens em cada módulo, contudo esse processo adicionaria ao treinamento da rede neural pixels não existentes, o que pode afetar negativamente o resultado.

Além disso, é possível aumentar a quantidade amostral do *dataset* a partir da rotação de cada fotografia em torno de todos os seus eixos (x, y e z). Como existem módulos de azulejos assimétricos, a partir dessas variações na posição da imagem, para as redes neurais, é como se fossem fornecidas novas imagens. É claro que alguns desses módulos são simétricos o que, nestes casos, vai impactar no treinamento aumentando a probabilidade para algumas das formas retratadas. Apesar disso, ampliar o *dataset* é importante considerando o número total de fotografias. Com esse processo, a quantidade final de imagens para o treinamento passa de 522 para 3132.

As rotações feitas em cada imagem seguiram as seguintes operações: rotação de (eixo z) 90, 180 e 270 graus e rotação, do tipo espelhamento, de forma horizontal (eixo x) e vertical (eixo y). Totalizando assim, 6 variações para cada imagem coletada e exposta na Figura 16.

Fonte: Imagem criada pelo autor

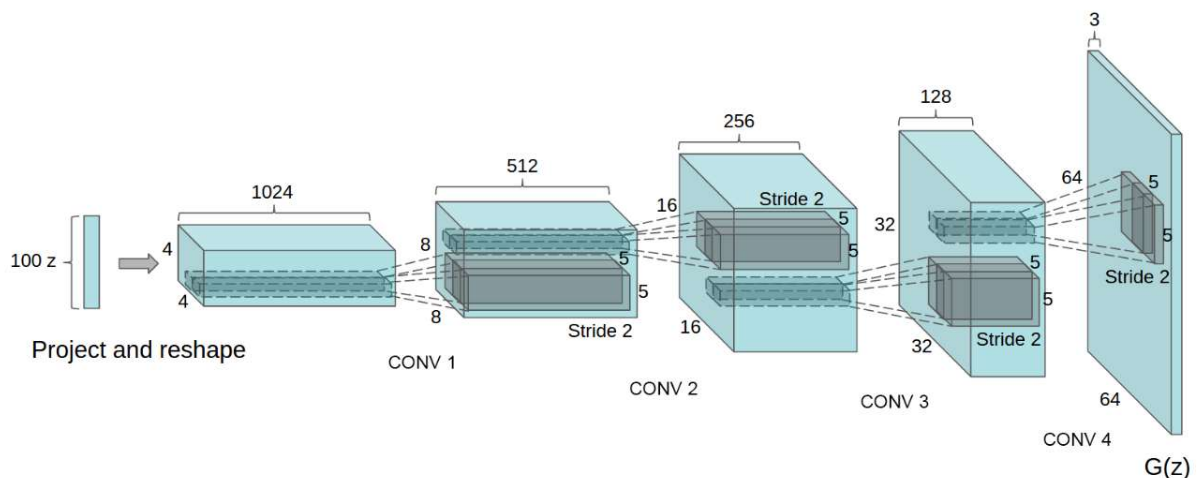
Figura 16 – Imagens rotacionadas para aumentar o *dataset*

Para todas as transformações de imagens feitas, foi usado a biblioteca opencv-python[6] em sua versão 4.11.0.86.

2.4 Estudos e testes de algoritmos

A arquitetura de GAN é a mais apropriada para este projeto pelo fato de ser uma implementação de rede neural para a geração de novos dados. Dentre as diversas implementações de GAN, a Deep Convolucional GAN (DCGAN) foi a escolhida para o projeto.

O primeiro ponto para esta escolha diz respeito a utilização do modelo de convolução. Como já apresentado, as redes neurais convolucionais são uma estrutura apropriada para o tratamento de imagens. A DCGAN combina as características de uma rede adversária e o processamento das fotografias com base na convolução, utiliza-se desta técnica para que o gerador construa uma imagem a partir de várias convoluções.



Fonte: (Radford et al., 2015, p. 6)

Figura 17 – Exemplo de camadas de convolução

Além disso, a DCGAN possui uma quantidade significativa de exemplos e base experimental disponível na internet e como é um algoritmo relativamente simples, necessita de menos poder computacional para fornecer resultados satisfatórios.

A Figura 17 ilustra esquematicamente uma DCGAN.

---

[6] https://docs.opencv.org/4.x/

2.5 Treinamento da GAN

Para o treinamento, a codificação da aplicação se dará utilizando Python[7], em sua versão 3.11.10, associado ao *framework* dedicado a modelos de *machine learning*: TensorFlow[8], na versão 2.19.0. Junto com o Tensor Flow há uma biblioteca chamada Keras[9] que é base para a implementação do modelo.

2.6 Implementação de API para "consumir" o modelo treinado

Por fim, para que seja possível "consumir" uma imagem gerada pelo modelo treinado neste projeto, será implementada um *endpoint* de *API REST*[10]. A implementação desta *API* será feita, também, em Python usando a biblioteca, ou *framework*, Flask[11] em sua versão 3.1.0.

---

[7] https://www.python.org
[8] https://www.tensorflow.org/
[9] https://keras.io/api/
[10] https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api
[11] https://flask.palletsprojects.com/en/stable/

3 DESENVOLVIMENTO

A construção de uma aplicação é influenciada pelo ambiente em que esta é desenvolvida, portanto, neste capítulo é especificada as configurações de *hardware* e *software* utilizadas, além daquelas já mencionadas na metodologia. Tem-se, também, o detalhamento do processo de desenvolvimento, com esclarecimentos sobre as decisões de implementação feitas e as comparações de duas formas de construção de uma DCGAN.

3.1 Configuração

O desenvolvimento da aplicação foi feito utilizando-se um computador "doméstico" com sistema operacional Windows 11 Pro, 64 bits, versão 24h2 e as seguintes características de *hardware*, mais relevantes, para o treinamento de uma IA:

- Processador AMD Ryzen 5 3600[12] de 6 núcleos (3,60 GHz);
- Placa de vídeo (GPU) Nvidia GeForce GTX 1660[13], com 6 gigabytes de memória;
- 32 megabytes de memória RAM.

As aplicações de IA, para um processamento mais eficiente, precisam utilizar as placas gráficas de um computador (GPU) e não o processador central da máquina (CPU). O uso de uma GPU reduz o tempo de cálculo das operações de treinamento dos modelos. Para que seja possível utilizar a GPU, a marca de placas gráficas, Nvidia[14], disponibiliza um kit de ferramentas chamado CUDA[15] para que os comandos e códigos escritos em diversas linguagens de programação possam indicar onde será feito o processamento de um código específico e direcioná-lo para a GPU e não a CPU. No caso, o *framework* escolhido para a implementação, TensorFlow, utiliza o CUDA para "acesso" à placa gráfica. Foi utilizada a versão 12.9 do CUDA, com a versão 576.28 do *driver* de placas gráficas da Nvidia, conforme apresentado na Figura 18.

---

[12] https://cpu-benchmark.org/pt-br/cpu/amd-ryzen-5-3600/
[13] https://www.techpowerup.com/gpu-specs/geforce-gtx-1660.c3365
[14] https://www.nvidia.com
[15] https://blog.nvidia.com.br/blog/o-que-e-cuda/

```
~/gan/gerador-azulejos git:(main)   (0.94s)
nvidia-smi
Mon Aug 11 11:20:42 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 575.51.03              Driver Version: 576.28         CUDA Version: 12.9     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce GTX 1660        On  |   00000000:07:00.0  On |                  N/A |
|  0%   45C    P8             14W /  130W |    2063MiB /   6144MiB |     15%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                        GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|    0   N/A  N/A              33      G   /Xwayland                             N/A      |
+-----------------------------------------------------------------------------------------+
```

Fonte: Imagem criada pelo autor

Figura 18 – Versão de driver Nvidia e CUDA

A instalação e configuração do *framework* de CUDA com o TensorFlow não é um processo simples. Como a evolução destas tecnologias acontece de forma acelerada, muitas vezes há divergências de versões e as instruções de instalações existentes são confusas. Para o projeto houve certa dificuldade nesta fase, o que atrasou a codificação da solução, mas que, eventualmente, foi superada.

O TensorFlow, a partir de sua versão 2.10, deixou de suportar GPUs nativamente no sistema operacional Windows[16]. Contudo, essa funcionalidade passou a ser possível através do Windows Subsystem for Linux (WSL). O WSL é uma aplicação que emula um sistema Linux dentro do ambiente Windows. Por esta razão, este projeto usou WSL2 em sua versão 2.3.26.0, com a distribuição Linux do sistema operacional Ubuntu 24.04, conforme Figura 19.

[16] https://www.tensorflow.org/install/pip?hl=pt-br#windows-native

```
~   (0.166s)
wsl --version

Versão do WSL: 2.3.26.0
Versão do kernel: 5.15.167.4-1
Versão do WSLg: 1.0.65
Versão do MSRDC: 1.2.5620
Versão do Direct3D: 1.611.1-81528511
Versão do DXCore: 10.0.26100.1-240331-1435.ge-release
Versão do Windows: 10.0.26100.4652
```

```
~/gan/gerador-azulejos git:(main)   (0.16s)
lsb_release -a

No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 24.04.2 LTS
Release:        24.04
Codename:       noble
```

Fonte: Imagem criada pelo autor

Figura 19 – Versão do WSL2 e distribuição Linux

Algumas ferramentas que foram usadas para o desenvolvimento do projeto, mas que não impactam nos resultados diretamente, foram:

- Git versão 2.43.0
- PyCharm 2025.2.0.1 (IDE Python)

3.2 Implementação

Neste tópico é descrito em detalhes as fases do desenvolvimento do *software* codificado. Apesar de o principal deste projeto ser o treinamento e visualização dos resultados, a preparação dos dados, ou seja, das imagens, exigiu que fosse criada uma funcionalidade para tratar essas informações. A seguir é feito o detalhamento de cada um desses usos da aplicação. Foi preciso conceber três funcionalidades centrais, cada uma delas projetada para atender a uma das etapas específica do fluxo de trabalho da aplicação e que foram intituladas de: REDIMENSIONAR, TREINAR e API.

Estruturalmente, o projeto consiste de um arquivo principal (*__main__.py*) que é "chamado" a partir de um terminal, ou *prompt*, com as funções acima descritas e parâmetros específicos para cada situação. A ideia foi reunir a aplicação em um único ponto de partida para que o projeto ficasse concentrado em apenas um programa, ao invés de criar três ou mais *scripts* Python diferentes. Toda a codificação feita está disponível no GitHub[17], no endereço eletrônico:

https://github.com/gustassifon/gerador-azulejos

[17] https://github.com/about

3.2.1 Estrutura da aplicação

```
~/gan/gerador-azulejos git:(main)   (0.152s)
tree --dirsfirst -I "*.png" -I "__pycache__/" -I __init__.py -I "2025*/"
.
├── config
│   ├── parametros_api.py
│   ├── parametros_dcgan_keras3.py
│   └── parametros_main.py
├── formatador
│   └── formatador_imagem.py
├── gan
│   ├── dcgan_keras3.py
│   └── dcgan.py
├── imagens
│   ├── imagens_redimensionadas
│   └── README.md
├── modelos
│   ├── 20250525113736_gerador_azulejos.h5
│   ├── 20250609113406_gerador_azulejos.h5
│   ├── 20250715114028_gerador_azulejos.h5
│   ├── 20250715120840_gerador_azulejos.h5
│   ├── 20250716112722_gerador_azulejos.h5
│   ├── 20250716120740_gerador_azulejos.h5
│   ├── 20250716135944_gerador_azulejos.h5
│   ├── 20250809154033_gerador_azulejos.h5
│   └── 20250809175927_gerador_azulejos.h5
├── referencias_uteis
│   └── speed_comparison.py
├── resultado
│   └── README.md
├── gerador_azulejo_api.py
├── __main__.py
├── README.md
├── requirements.txt
└── wsgi.py

9 directories, 23 files
```

Fonte: Imagem criada pelo autor

Figura 20 – Hierarquia de pastas para a aplicação: gerador-azulejos

A Figura 20, apresenta a “árvore” de pastas que constitui a aplicação. Como já foi mencionado, a aplicação é iniciada a partir do arquivo *__main__.py*. Esse nome foi escolhido de propósito, por uma razão específica: de acordo com as especificações do Python, é possível criar pacotes compactados que podem ser executados se contiverem um arquivo com esse nome. Esses pacotes são conhecidos como Python Zip Applications, um padrão definido na PEP 441[18].

Seguindo esse padrão, é possível criar um arquivo compactado, como por exemplo *gerador-azulejos.zip*, que pode ser executado diretamente pelo terminal, sem precisar descompactar. Essa maneira facilita a distribuição do programa, pois não é necessário enviar

---

[18] https://peps.python.org/pep-0441

uma pasta inteira, basta fornecer um único arquivo. Embora neste projeto não seja o método de distribuição escolhido, o *software* foi desenvolvido pensando na possibilidade de usar essa abordagem no futuro.

No arquivo principal ficam definidos os redireccionamentos de acordo com a função escolhida e uma função que prepara as imagens para o *dataset*. Contudo é no arquivo *formatador_imagem.py* dentro da pasta *formatador* (conforme ilustrado na Figura 20) que contém todo o núcleo da codificação para esta funcionalidade que será descrita melhor no próximo item deste trabalho.

Na hierarquia apresentada na Figura 20, a pasta *config* guarda os arquivos que fazem o tratamento dos parâmetros da aplicação quando esta é levantada a partir do *prompt* de comando. São os arquivos deste diretório que contém a codificação para validar e tratar as opções de entrada ao inicializar a aplicação. Apenas o arquivo *parametros_main.py* é utilizado, os demais foram criados para testes de execuções independentes da API e do treinamento de um dos modelos criados. Apesar de não serem utilizados no resultado, ficaram no projeto como referência.

A pasta *gan* possui duas codificações de uma mesma arquitetura de inteligência artificial para uma DCGAN. Em *dcgan.py* há uma implementação que utiliza uma codificação antiga em relação à que foi feita em *dcgan_keras3.py*. No tópico sobre a função TREINAR, maiores detalhes sobre essa codificação são explicados.

O diretório *imagens* é usado para guardar todas as imagens do projeto, tanto as processadas para compor o *dataset*, quanto as originais. Contudo esses dados não são enviados ao GitHub por causa do tamanho de cada arquivo. Alguns deles podem ser grandes demais para a versão gratuita do GitHub. Os arquivos originais estão disponibilizados para download através do endereço eletrônico:

https://ln5.sync.com/4.0/dl/e023c7610#mdfpm6y3-72ixrgpw-puze3ren-xhs2igiv

Os treinamentos que foram feitos durante os testes do projeto estão sob a pasta *modelos*. Os arquivos dentro deste diretório são identificados com data e horário em que foram criados, desta forma é possível identificar seus *logs* que ficam dentro da pasta *resultado*. Nesta última, existem pastas com data e hora de execução dos treinamentos realizados, mas que não estão listados na Figura 20. Contudo, essas pastas estão disponíveis no GitHub para consulta. A Figura 21 apresenta um exemplo da estrutura da pasta *resultado*.

```
.
├── 20250524212307
│   ├── imgs
│   ├── model
│   └── historico_execucao.txt
├── 20250525113736
│   ├── imgs
│   ├── model
│   └── historico_execucao.txt
├── 20250609113406
│   ├── imgs
│   ├── model
│   └── historico_execucao.txt
```

Fonte: Imagem criada pelo autor

Figura 21 – Exemplo de hierarquia da pasta *resultado*

Os arquivos *historico_execucao.txt* contém o passo-a-passo – *log* – das execuções feitas. Em um primeiro momento os *logs* não continham tanta informação, mas no decorrer do desenvolvimento foram sendo adicionadas mais dados relevantes ao histórico, como parâmetros usados para o treinamento e as camadas criadas para a rede neural que compõe os modelos gerador e discriminador da DCGAN.

Por fim, há na pasta *referencias_uteis* um *script* para validar a correta instalação do *framework* CUDA. Neste programa é feita uma implementação que compara uma rede neural sendo processada com a CPU e também com GPU para validar se de fato o ambiente Python está apto a realizar o acesso a placa gráfica corretamente. O arquivo *gerador_azulejo_api.py* é onde está a implementação da API REST usada para exibir as imagens aleatoriamente a partir dos modelos treinados. Maior pormenorização de sua codificação é feita no tópico API. Os demais arquivos que se apresentam na Figura 20, são arquivos de suporte seja para o correto funcionamento do Git, GitHub, PyCharm etc.

3.2.2 REDIMENSIONAR

A primeira funcionalidade, REDIMENSIONAR, consiste em um módulo de tratamento de imagens, cuja finalidade é padronizar as dimensões dos arquivos visuais e criar novos, baseado na imagem original, porém em posições diferentes – rotacionando e espelhando, esta técnica é chamada de *data augmentation* (literalmente, aumento dos dados). Assim, na pasta *formatador,* tem-se o arquivo *formatador_imagens.py,* que define a classe[19] FormatadorImagem. Esta é a classe que será usada tanto para o redimensionamento quanto para a ampliação dos dados do *dataset.*

A classe, FormatadorImagem, utiliza a biblioteca do opencv-python para as manipulações das imagens, seja para ler um arquivo, como para redimensionar, rotacionar e espelhar. A estrutura da classe pode ser observada na Figura 22. Ela possui atributos e métodos para que, quando utilizada, seu objeto contenha todas as imagens manipuladas, bem como a original, em uma única instanciação. Ao instanciar um objeto desta classe, o utilizador precisa informar um caminho para a imagem original e a intenção de novo tamanho de imagem, caso não determinado a classe utiliza o valor de 128 pixels como padrão. Na Figura 23 tem-se a definição do construtor da classe que ao ser iniciado já constrói todas as variações possíveis para o arquivo visual passado como parâmetro. Ou seja, basta construir um objeto da classe e executar o método salvar em seguida para gravar todas as imagens em disco. Cada método criado para classe define a rotação e seu espelhamento conforme descrito em seus nomes. São criados cinco arquivos manipulados, além do original redimensionado. Parte dos objetivos do redimensionamento já foram descritos no tópico “Criação do *dataset*” da Metodologia deste trabalho.

---

[19] https://docs.python.org/3/tutorial/classes.html

```
FormatadorImagem
    __init__(self, caminho_imagem, nova_altura=128, nova_largura=128)
    __str__(self)
    redimensionar(self, imagem=None, nova_altura=None, nova_largura=None)
    rotacionar_90(self, imagem=None)
    rotacionar_180(self, imagem=None)
    rotacionar_270(self, imagem=None)
    espelhar_horizontal(self, imagem=None)
    espelhar_vertical(self, imagem=None)
    salvar(self, diretorio)
    altura_original
    imagem_espelhada_horizontal
    imagem_espelhada_vertical
    imagem_original
    imagem_redimensionada
    imagem_rotacionada_180
    imagem_rotacionada_270
    imagem_rotacionada_90
    largura_original
    nome_imagem
    nova_altura
    nova_largura
```

Fonte: Imagem criada pelo autor

Figura 22 – Estrutura da classe FormatorImagem

```
class FormatadorImagem:  2 usages  Gustavo Leal
    def __init__(self, caminho_imagem: str, nova_altura: int = 128, nova_largura: int = 128):
        self.imagem_original = cv2.imread(caminho_imagem)
        self.nome_imagem = os.path.basename(caminho_imagem)
        self.altura_original = self.imagem_original.shape[0]
        self.largura_original = self.imagem_original.shape[1]
        self.nova_altura = nova_altura
        self.nova_largura = nova_largura
        self.imagem_redimensionada = self.redimensionar()
        self.imagem_rotacionada_90 = self.rotacionar_90()
        self.imagem_rotacionada_180 = self.rotacionar_180()
        self.imagem_rotacionada_270 = self.rotacionar_270()
        self.imagem_espelhada_horizontal = self.espelhar_horizontal()
        self.imagem_espelhada_vertical = self.espelhar_vertical()
```

Fonte: Imagem criada pelo autor

Figura 23 – Construtor da classe FormatadorImagem

Para usar a funcionalidade de REDIMENSIONAR é necessário chamar o programa indicando essa intenção e seus parâmetros. Há dois parâmetros obrigatórios: a indicação do diretório de origem – onde estão as imagens originais – e um diretório de destino – onde serão gravadas as novas imagens. Há ainda um parâmetro opcional que é o tamanho para o qual se pretende redimensionar a imagem. Como já indicado, se não for informado o valor padrão é de 128 pixels.

Este recurso da ferramenta foi pensado para sempre trabalhar com um conjunto e não com imagens individuais, por tanto, quando a aplicação é chamada ela espera uma pasta com uma lista de arquivos e processa todos eles. Abaixo a Figura 24 apresenta a descrição da aplicação e um exemplo de uso.

```
~/gan/gerador-azulejos git:(main)  (3.581s)
python __main__.py redimensionar -h

positional arguments:
  diretorio_origem   Indicar o diretório das imagens que serão usadas para compor o dataset.
  diretorio_destino  Indicar o diretório onde as novas imagens criadas para serem usadas pelo modelo serão salvas.

options:
  -h, --help         show this help message and exit
  -tam TAM           Indicar o tamanho da imagem, por padrão usa 128x128. As imagens são sempre quadradas. Parâmetro opcional

~/gan/gerador-azulejos git:(main)  (10.638s)
python __main__.py redimensionar imagens/originais_preparadas imagens/redimensionadas -tam 256
```

Fonte: Imagem criada pelo autor

Figura 24 – Exemplo de uso do recurso REDIMENSIONAR

3.2.3 TREINAR

A segunda funcionalidade corresponde à implementação para o treinamento de um modelo de inteligência artificial generativa, especificamente concebido para a produção de imagens que remetam esteticamente aos azulejos portugueses. Este é o recurso principal deste trabalho. O sistema recebe como entrada as imagens previamente tratadas e realiza o processo de aprendizado, ajustando parâmetros internos a fim de gerar novas composições visuais que preservem as características estilísticas do conjunto original.

Na pasta *gan*, tem-se dois arquivos: *dcgan.py* e *dcgan_keras3.py*. O primeiro tem implementado a classe DCGAN que representa uma codificação mais antiga, foi feita com recursos do *framework* TensorFlow e Keras de versões passadas. Não significa que essas implementações não funcionem, mas sim que o desenvolvimento de ambas as plataformas utilizadas sofreram alterações no passar dos anos e ganharam novas funcionalidades. O tutorial[20] em que a implementação foi baseada data de dezembro de 2021, quase quatro anos atrás.

Já o segundo arquivo, *dcgan_keras3.py* foi feito a partir de codificação mais atual, disponibilizado como tutorial[21] pela própria documentação do Keras. Este último foi revisado em 2023 e possui uma abordagem mais recente que utiliza a versão 3 do *framework*. Neste arquivo foi implementada a classe GAN e a auxiliar GANMonitor, além de alguns métodos auxiliares externos as classes. Para o *dcgan_keras3.py* tentou-se reproduzir a codificação o mais próximo do tutorial para comparar o resultado das duas implementações.

O recurso de treinamento poderá ser usado conforme exposta na Figura 25.

---

[20] https://github.com/jeffheaton/present/blob/master/youtube/gan/gans_scratch.ipynb
[21] https://keras.io/examples/generative/dcgan_overriding_train_step/

```
~/gan/gerador-azulejos git:(main)  2 files changed, 17 insertions(+), 8 deletions(-)  (9.837s)
python __main__.py treinar -h

positional arguments:
  diretorio_dataset     Indicar o diretório das imagens já redimensionadas e tratadas para o dataset.
  diretorio_resultado   Indicar o diretório onde serão salvas as imagens geradas pelas épocas e o log do treinamento.

options:
  -h, --help            show this help message and exit
  -epocas EPOCAS        Indicar a quantidade de épocas de treinameto, parâmetro opcional e por default utliza 100 épocas.
  -dcgan {PADRAO,KERAS3}
                        Indicar a implementação utilizada, se PADRAO ou KERAS3

~/gan/gerador-azulejos  main  2 · +17 -8  Conversations Ctrl Shift Y

python __main__.py treinar -epocas 50 -dcgan PADRAO imagens/imagens_redimensionadas resultado
```

Fonte: Imagem criada pelo autor

Figura 25 – Exemplo de uso do recurso TREINAR

3.2.3.1 Comparativo entre *dcgan.py* e *dcgan_keras3.py*

A primeira abordagem de implementação não utiliza os recursos da classe Model[22] do Keras, ou seja, a classe DCGAN criada não herda as propriedades da classe do *framework* que representa um modelo de IA "treinável". Aqui, a utilização de uma classe serviu para o propósito de agrupar a codificação em um mesmo objeto e conter em si mesma todas as propriedades e métodos necessários para a execução do treinamento. Apesar de existir sim a utilização da classe Model dentro da classe DCGAN, para criar os modelos do discriminador e do gerador, o objetivo não é exatamente o mesmo daquele implementado na *dcgan_keras3.py.* A segunda abordagem define a classe GAN herdando as propriedades e métodos da classe Model do Keras e redefine alguns de seus métodos, como o *compile()*[23] e *train_step()*[24], por exemplo. Na segunda implementação a classe GAN agrega o discriminador e gerador em um mesmo objeto e deixa a cargo dos métodos da própria classe Model, como o *fit()*[24], a "administração" do treinamento e compilação das redes neurais. Ou seja, na primeira implementação foi preciso escrever todas a sequência do treinamento das redes, enquanto na segunda, apesar da reescrita do método *train_step()*, parte da complexidade do treinamento fica por conta da classe Model.

[22] https://keras.io/api/models/model/
[23] https://keras.io/api/models/model_training_apis/
[24] https://keras.io/guides/custom_train_step_in_tensorflow/

Assim como a questão do treinamento, outro ponto de diferença entre as duas implementações, diz respeito a preparação do *dataset*. Na codificação que utilizou Keras 3, a função *keras.utils.image_dataset_from_directory()*[25] lê um diretório e trata, sem maiores codificações, as imagens existentes no local indicado. O objeto *dataset* resultante desta função já está pronto para ser usado no treinamento do modelo. É preciso apenas normalizá-lo para que os dados das imagens fiquem entre 0 e 1, já que nesta codificação, para o gerador, é usada uma função de ativação do tipo sigmóide. É feita a seguinte operação para obter o resultado de normalização:

```
dataset = dataset.map(lambda x: x / 255.0).
```

Para *dcgan.py* não foi usada a funcionalidade já implementada. Toda a preparação do objeto *dataset* foi codificado tal como a Figura 26 na função *__preparar_dataset()*. Para esta implementação foi preciso normalizar de acordo com a função de ativação do tipo tangente hiperbólica e não sigmóide, esta escolha foi baseada nos tutoriais utilizados e não por escolha do projeto. O trecho de código que realiza a normalização é determinado pelo seguinte trecho de código:

```
dataset = (dataset - 127.5) / 127.5
```

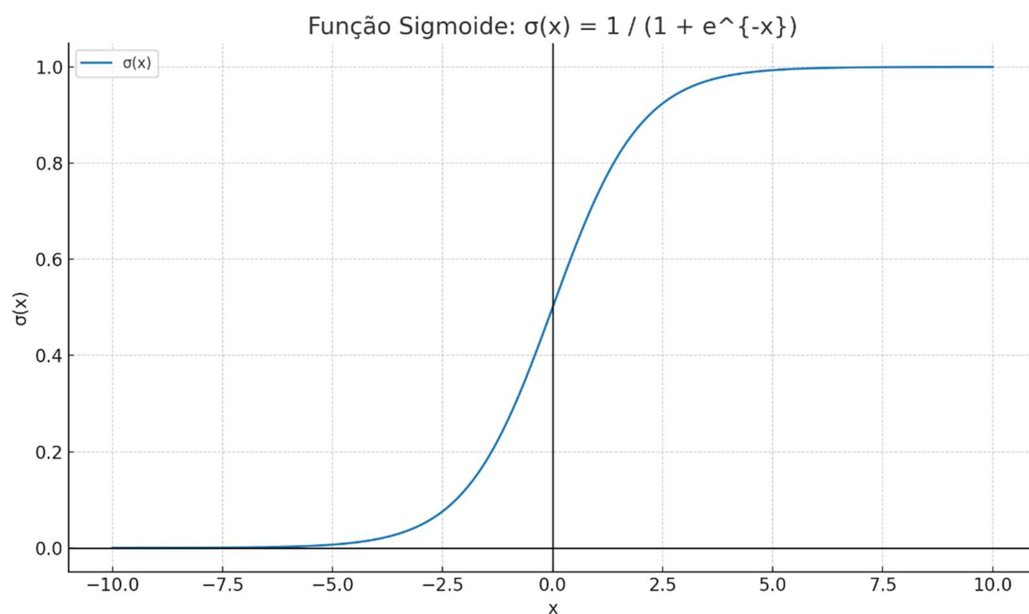
A diferença entre a função sigmóide e tangente hiperbólica é descrita nas Figura 27 e Figura 28. Após trabalhar as imagens e carregá-las em um *array,* posteriormente é convertido em um objeto utilizável pelo modelo a partir da função do TensorFlow *tf.data.Dataset.from_tensor_slices()*[26].

[25] https://keras.io/api/data_loading/image/
[26] https://www.tensorflow.org/api_docs/python/tf/data/Dataset#from_tensor_slices

```
class DCGAN:  2 usages  Gustavo Leal +1 *
    def __preparar_dataset(self):  1 usage  Gustavo Leal
        self.__logger('Preparando o dataset...')

        if self.caminho_imagens_dataset is not None and os.path.exists(self.caminho_imagens_dataset):
            dataset = []
            lista_arquivos = os.listdir(self.caminho_imagens_dataset)
            qtde_imagens = len(lista_arquivos)

            # Se não existir imagens dentro do diretório informado, não executa o processamento do dataset
            if qtde_imagens > 0:
                self.__logger('Carregando as imagens e convertendo em ndarray...')
                for arquivo in tqdm(lista_arquivos):
                    imagem = cv2.imread(os.path.join(self.caminho_imagens_dataset, arquivo))
                    imagem = np.asarray(imagem)
                    dataset.append(imagem)

                dataset = np.asarray(dataset).astype(np.float32)

                # Normaliza os dados para um conjunto de valores entre -1 e 1 para o uso da função "tanh". Por isso a
                # subtração do valor por 127,5 (metade de 255, valor máximo de uma cor RGB) e na sequência a divisão por
                # este mesmo valor
                self.__logger('Normaliza o dataset (-1, 1)')
                dataset = (dataset - 127.5) / 127.5

                self.__logger('Cria o dataset no formato do TensorFlow e atribui a propriedade do objeto')
                self.dataset = (
                    tf.data.Dataset.from_tensor_slices(dataset).shuffle(qtde_imagens).batch(self.tamanho_lote)
                )
                return True

        self.__logger("Não foi indicado um dataset para treinamento.")
        return False
```

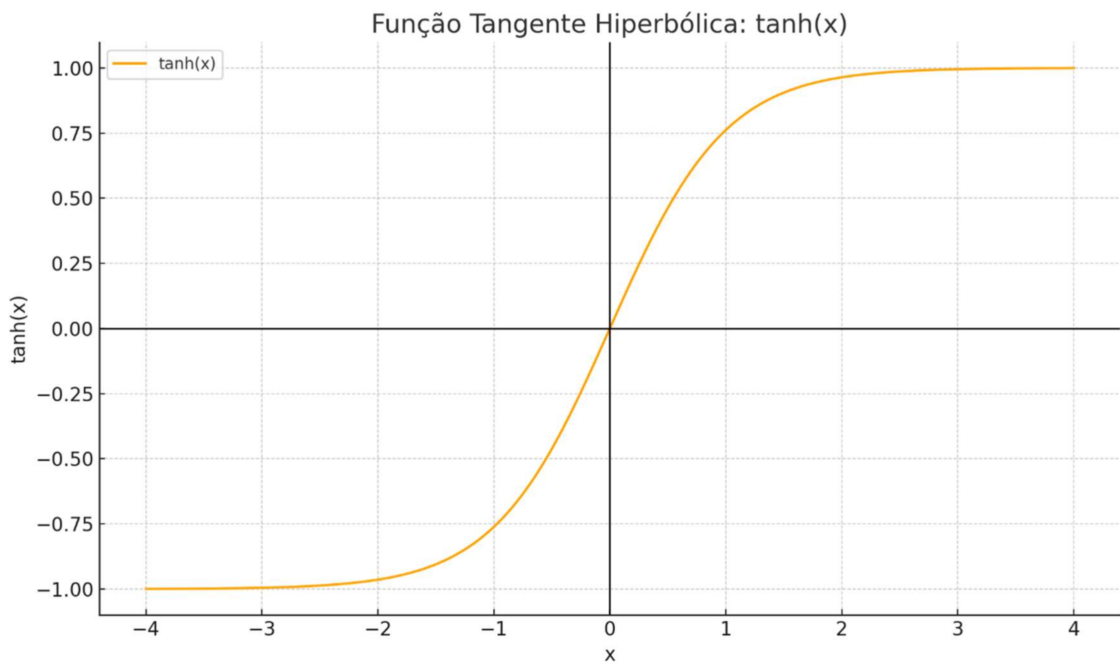
Fonte: Imagem criada pelo autor

Figura 26 – Função de preparação do *dataset* para classe DCGAN



Fonte: Prompt, “Gere para mim um gráfico de uma função sigmoide lado a lado com outro gráfico de uma função tangente hiperbólica”. Complementado por: “Refaça as imagens marcando o eixo Y nos gráficos” (Open AI, 2025)

Figura 27 – Gráfico da função sigmóide



Fonte: Prompt, “Gere para mim um gráfico de uma função sigmoide lado a lado com outro gráfico de uma função tangente hiperbólica”. Complementado por: “Refaça as imagens marcando o eixo Y nos gráficos” (Open AI, 2025)

Figura 28 – Gráfico de uma função tangente hiperbólica

Por último, uma divergência entre as duas implementações concerne a criação das estruturas dos modelos discriminador e gerador. Na classe DCGAN tentou-se uma abordagem dinâmica, em que o número de camadas é definido pelo tamanho da imagem, mas limitado a um tamanho mínimo e máximo de acordo com os filtros utilizados. Os filtros são utilizados para determinar o tamanho das matrizes que representam a imagem, assim para o gerador, ao final, precisamos ter um filtro de tamanho 3, representando as camadas R (*red*) G (*green*) B (b*lue*). Como a transformação da imagem pelo gerador acontece sempre dobrando seu tamanho, ou seja, de 4 x 4 depois temos 8 x 8, 16 x 16 e assim por diante, há uma repetição baseada em uma potência de 2. Usando o tamanho da imagem em uma função logarítmica na base 2 é possível encontrar o total de repetições necessárias para fazer esta transformação na imagem. Enquanto a imagem aumenta, o número de camadas é reduzido à metade, iniciado a partir do valor de 1024, mas finalizando em uma camada final de 3. A implementação dessa estrutura pode ser vista na Figura 29 na função __*constuir_gerador()*.

```
class DCGAN:  2 usages  Gustavo Leal +1 *

    def __construir_gerador(self):  1 usage  Gustavo Leal +1 *
        repeticoes = int(math.log2(self.tamanho_imagem)) - 2
        modelo = Sequential()
        modelo.add(layers.Input(shape=(self.dimensao_ruido,)))
        modelo.add(layers.Dense(4 * 4 * self.tamanho_filtro_maximo, use_bias=False))
        modelo.add(layers.BatchNormalization())
        modelo.add(layers.LeakyReLU())
        modelo.add(layers.Reshape((4, 4, self.tamanho_filtro_maximo)))

        for i in range(repeticoes):
            i += 1
            filtros = self.tamanho_filtro_maximo >> i
            filtros = max(filtros, self.tamanho_filtro_minimo)

            modelo.add(
                layers.Conv2DTranspose(
                    filters=filtros,
                    kernel_size=self.tamanho_kernel,
                    padding='same',
                    use_bias=False,
                    strides=self.tamanho_strides
                )
            )

            modelo.add(layers.BatchNormalization())
            modelo.add(layers.LeakyReLU(alpha=0.2))

        modelo.add(
            layers.Conv2DTranspose(
                filters=self.canais_imagem, kernel_size=self.tamanho_kernel, padding='same', use_bias=False, activation='tanh'
            )
        )
        modelo.name = 'gerador'

        return modelo
```

Fonte: Imagem criada pelo autor

Figura 29 – Função da classe DCGAN que realiza a construção do gerador

O discriminador é criado de forma análoga ao gerador, contudo ao invés de aumentar a imagem e diminuir os filtros, no discriminador é trabalha de forma oposta. Reduz-se a imagem à medida que se aumenta o filtro e ao fim tem-se uma camada de um único *array.* É como se a imagem que se inicia representada por uma matriz é reduzida a uma “linha”. Para o discriminador o resultado é 0 ou 1, ou seja, é ou não é uma imagem válida e por isso ela é reduzida dessa forma. Assim como o gerador, na DCGAN o número de camadas do discriminador foi feito de forma dinâmica, seguindo o mesmo raciocínio descrito anteriormente.

Já na implementação do arquivo *dcgan_keras3.py* as redes neurais foram definidas de forma manual, codificada no código sem dinamicidade.

3.2.3.2 Etapa de treinamento

Apesar das diferenças de codificação das duas abordagens, a etapa de treinamento tem como cerne a mesma concepção. O treinamento das duas redes, discriminadora e geradora, acontecem ao mesmo tempo e passam, de forma simplificada, pelo seguinte processo:

1) Criar ruído;
2) Utilizar o ruído para gerar imagens a partir da rede geradora;
3) Utilizar imagens geradas e as imagens reais do *dataset* para que a rede discriminadora faça uma predição;
4) As predições geradas são rotuladas como falsas (para as imagens do gerador) e verdadeiras (para as imagens do *dataset*);
5) É calculado a perda do discriminador com as predições rotuladas;
6) A predição do discriminador para as imagens geradas (falsas) são rotuladas como verdadeiras;
7) Esta predição é usada para calcular a perda da rede geradora;
8) Usa-se a o cálculo de perda de cada rede, respectivamente, para determinar os respectivos gradientes[27], os pesos e vieses são recalculados através do algoritmo de retropropagação;
9) Repete-se o processo um número N de vezes, chamado de épocas.

Mais uma diferença entre as duas implementações se dá no momento de determinação dos gradientes. Para realizar o cálculo de perda e manipulação dos pesos de uma rede é preciso utilizar uma estrutura de contexto do TensorFlow chamada GradientTape. Na implementação da classe DCGAN o cálculo das duas redes é feito em um mesmo contexto. Já no *dcgan_keras3.py* esse mesmo cálculo acontece em contextos GradientTape separados.

Algumas observações, em *dcgan.py* o rótulo 1 indica imagens reais, no *dcgan_keras3.py* é invertido, 0 indica imagens reais e 1 imagens falsas. No *dcgan_keras3.py* é utilizado um recurso de adicionar um deslocamento aleatório nos rótulos, a fim de minimizar a tendência que a rede neural pode ter de enviesar para uma tendência dos dados específica,

---

[27] O gradiente é o que determina, para as redes neurais, como os pesos e vieses irão ser manipulados para aumentar/diminuir o valor de perda calculado em próxima iteração do treinamento da rede. Este assunto foi citado neste trabalho, na Revisão de Literatura, no tópico sobre Inteligência Artificial.

deixando assim de generalizar e convergir para o padrão dos dados de sua base de informações. O trecho especificado é o que segue:

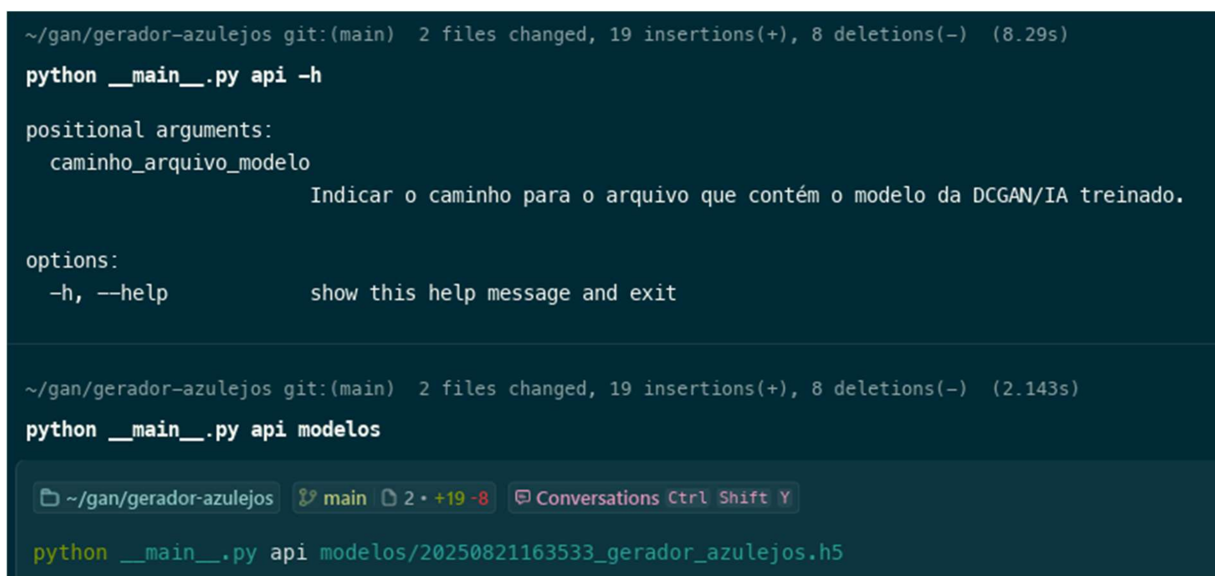
```
labels += 0.05 * tf.random.uniform(tf.shape(labels))
```

3.2.4 API

Por fim, a terceira funcionalidade corresponde à criação de uma API REST capaz de receber requisições HTTP e retornar uma imagem gerada a partir do modelo treinado. Essa interface possibilita a integração da solução a outros sistemas e garante a disponibilização das imagens de forma dinâmica, uma vez que cada requisição resulta na produção de uma nova imagem aleatória derivada do modelo generativo treinado.

O arquivo *gerador_azulejo_api.py* contém a implementação da API REST.

Para colocar a API em funcionamento é preciso chamar a aplicação passando alguns parâmetros para o programa conforme Figura 30.



Fonte: Imagem criada pelo autor

Figura 30 – Exemplo de uso do recurso API

Em concordância com o que já foi exposto na Metodologia deste trabalho, a API foi desenvolvida usando o *framework* Flask. O benefício de utilizar o Flask é a facilidade com que

é criado um *endpoint* REST. A parte específica da implementação que trata as requisições HTTP é muito simplificado com a utilização dessa ferramenta. A Figura 31 apresenta o trecho de código que faz o trabalho relacionado ao protocolo HTTP, o resto da implementação do arquivo *gerador_azulejo_api.py*, é todo feito para carregar um modelo de IA, recuperar uma imagem e convertê-la em um arquivo PNG[28].

```
73    app = Flask(__name__)
74
75    @app.route( rule: '/', methods=['GET'])  Gustavo Leal +1
76    def gerar_azulejo():
77        return send_file(recuperar_imagem_modelo(caminho_modelo), mimetype='image/png')
78
79    return app
80
```

Fonte: Imagem criada pelo autor

Figura 31 – Trecho de código que usa o *framework* Flask e trata todas as questões relacionadas ao protocolo HTTP

A API foi projetada para funcionar com arquivos de modelos treinados e salvos através da função Model.save()[29] do Keras. Afinal, para realizar a carga da inteligência artificial será usado, também, o framework do TensorFlow para fazer o "*load*" do arquivo. Usualmente esses arquivos recebem a extensão "h5", mas é possível usar uma extensão própria do Keras, intitulada "keras". Para este projeto poderá ser informado uma pasta com diversos arquivos de modelos ou apenas um arquivo. Se a primeira opção for feita, o programa escolhe aleatoriamente um arquivo dentro da pasta para utilizar e gerar a imagem, se não, usa o modelo indicado e a cada *request* feito à API gera uma nova imagem. A Figura 32 mostra a parte do código que realiza essa operação.

---

[28] https://www.adobe.com/pt/creativecloud/file-types/image/raster/png-file.html
[29] https://keras.io/api/models/model_saving_apis/model_saving_and_loading/

```
18
19      def recuperar_imagem_modelo(caminho_arquivo: str):  Gustavo Leal +1 *
20          caminho_arquivo = Path(caminho_arquivo)
21
22          if caminho_arquivo.is_dir(): # Significa que é uma pasta com vários modelos
23              lista_arquivos = os.listdir(caminho_arquivo)
24              nome_arquivo = random.choice(lista_arquivos)
25              caminho_arquivo = caminho_arquivo.joinpath(nome_arquivo)
26
27          gerador = keras.models.load_model(str(caminho_arquivo))
```

Fonte: Imagem criada pelo autor

Figura 32 – Trecho de código que recupera um arquivo aleatório de uma pasta ou utiliza o arquivo passado como parâmetro

O restante da codificação do arquivo da API refere-se à geração da imagem e conversão do resultado. Ao gerar uma gravura usando o Keras o resultado não é uma imagem em formato PNG. O resultado do modelo é um Tensor[30], que se aproxima de uma matriz. Antes de retornar a figura via requisição HTTP, é preciso convertê-la. O restante da implementação trata desta questão, a Figura 33 apresenta as operações matemáticas necessárias para isso. Pelo fato de utilizar duas implementações de DCGAN, uma usar função de ativação de tangente hiperbólica e a outra sigmóide, é preciso identificar qual o modelo sendo utilizado para realizar a operação correta no Tensor. Foi usado o nome da primeira camada do modelo para realizar essa identificação.

[30] https://www.tensorflow.org/guide/tensor

```python
imagem_gerada = gerador(ruido)
imagem_gerada = imagem_gerada.numpy().astype("float32")

# A implementação do dcgan.py retorna com o nome "dense" e assim é possível filtrar o modelo que usa a tanh
if gerador.layers[0].get_config()['name'] == 'dense':
    imagem_gerada = imagem_gerada + 127.5
    imagem_gerada = imagem_gerada * 127.5
# O outro modelo é retornado com o nome "dense_1" e é usado a função sigmóide.
else:
    imagem_gerada *= 255

imagem_gerada = imagem_gerada.astype("uint8")

# Aumenta a imagem para 256x256 para melhorar a resposta da API
imagem = cv2.resize(imagem_gerada[0], dsize: (256, 256), interpolation=cv2.INTER_CUBIC)
```

Fonte: Imagem criada pelo autor

Figura 33 – Trecho de código que apresenta a conversão de um Tensor em uma imagem PNG

Ao final há um pequeno trecho de código que adiciona um texto na imagem para identificar o modelo e converte em PNG a matriz trabalhada. É inserido na figura a data e hora do treinamento da IA, dessa forma é possível identificar qual foi o treinamento que deu origem à gravura retornada pela API. A Figura 34 apresenta o trecho de código com a implementação.

```python
# Detectar tamanho da imagem e ajustar fonte e posição
altura, largura = imagem.shape[:2]

# Adicionar texto na imagem com a data do modelo para identificar qual está sendo o modelo usado
cv2.putText(
    imagem,
    caminho_arquivo.name.split('_')[0],
    org: (0, altura - 2),
    cv2.FONT_HERSHEY_PLAIN,
    fontScale: 0.8,
    color: (0, 0, 0),
    thickness: 1,
    cv2.LINE_AA
)
resultado, buffer = cv2.imencode( ext: '.png', imagem)
```

Fonte: Imagem criada pelo autor

Figura 34 – Trecho de código onde é feita a inclusão de um texto na imagem para identificar o modelo gerador

4 RESULTADOS

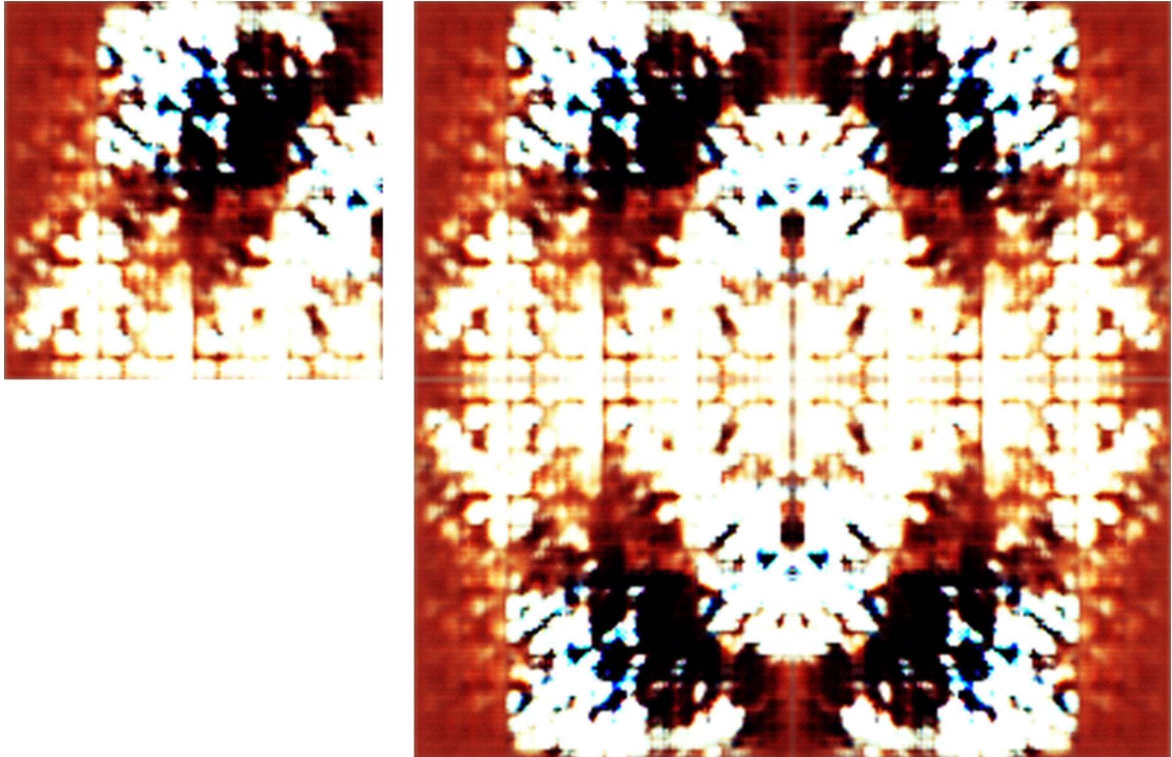
Durante o curso do projeto foram feitos diversos treinamentos para validar a implementação da inteligência artificial. Como já mencionado, em um primeiro momento havia apenas uma implementação – *dcgan.py* – e posteriormente foi construído um segundo modelo – *dcgan_keras3.py* – para comparar os resultados.

Apesar de os resultados não terem atingido plenamente a semelhança com um módulo de azulejo português, o trabalho conseguiu desenvolver uma DGAN que gera imagens com potencial para originar novos padrões.

4.1 Análise geral

As imagens geradas são abstratas e, no geral, são módulos passíveis de uma construção de padrões. A Figura 35 exemplifica isso. Ela apresenta um resultado obtido de um modelo treinado em 09/06/2025 (20250609113406)[31], feito a partir da primeira implementação da DCGAN e treinado por 100 épocas. À esquerda da gravura é apresentado o módulo e a direita tem-se uma composição feita a partir da rotação do módulo para compor um possível resultado.

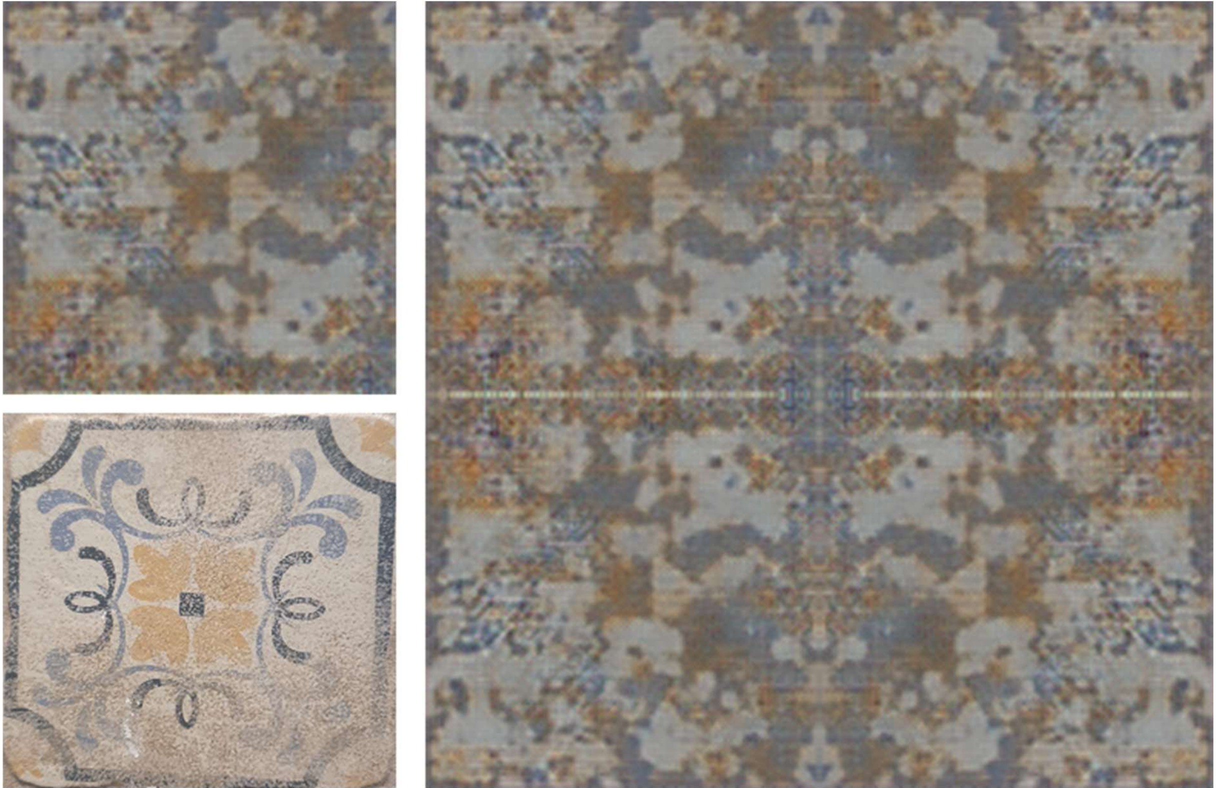
[31] Numeração usada para identificar cada treinamento de forma única



Fonte: Imagem criada pelo autor

Figura 35 – Exemplo de módulo gerado por modelo treinado (20250609113406)

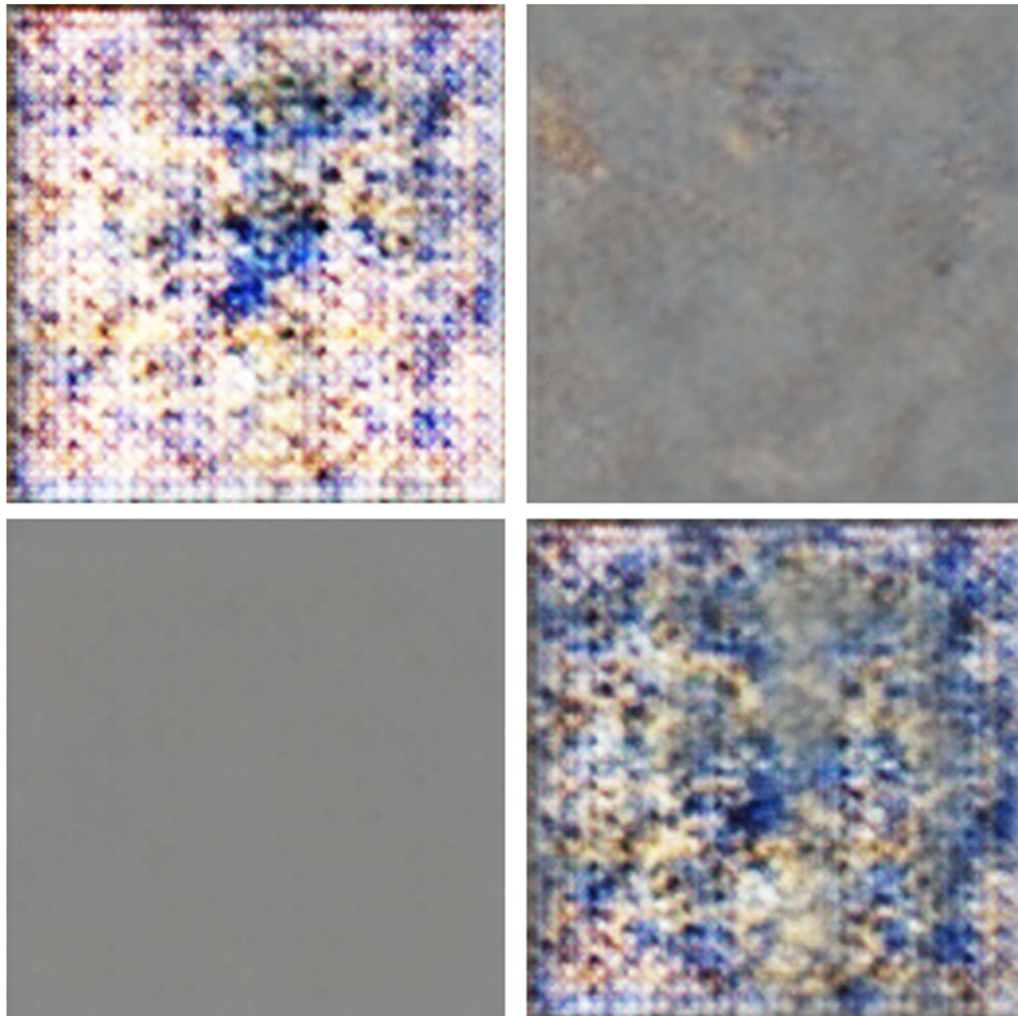
O exemplo da Figura 35, apresenta uma imagem que dificilmente seria associada a um azulejo. Nas fotografias que compuseram o *dataset*, não há uma foto que poderia ser referenciada para chegar neste resultado. Contudo, outro treinamento, feito em 16/07/2025 (20250716112722), também de 100 épocas, o resultado é um pouco diferente. Apesar de ainda se ter uma imagem abstrata sem definições claras, é possível notar alguma proximidade a um azulejo. A Figura 36 apresenta esse exemplo de treinamento. À esquerda tem-se o módulo gerado pelo modelo e abaixo do módulo uma das fotos utilizadas no *dataset* para fins de comparação. À direita tem a composição de um padrão a partir do módulo gerado. Este módulo tem cores mais próximas de alguns tipos de azulejos do *dataset*, apresenta textura que se assemelha a uma parede e é possível vislumbrar a formação de uma espécie de "flor" em seu centro. O módulo real, apresentado no canto inferior esquerdo possui cores que se aproximam da criação e um desenho que remete ao que foi gerado pela inteligência artificial.



Fonte: Imagem criada pelo autor

Figura 36 – Exemplo de módulo gerado por modelo treinado comparado com uma fotografia do *dataset* (20250716112722)

Alguns outros treinamentos tendem a levar o resultado para uma imagem monocromática. A Figura 37 apresenta o resultado de treinamento feito, também, em 16/07/2025 (20250716135944) de 100 épocas e que resultou em um comportamento que, no geral, retorna uma imagem cinza, apesar de em alguns momentos conseguir devolver imagens diversas. Todas as imagens foram retornadas pelo mesmo modelo treinado e consistentemente são retornadas imagens completamente cinzas ou com uma alguma textura.



Fonte: Imagem criada pelo autor

Figura 37 – Exemplos de módulos gerados em teste de treinamento (20250716135944)

É importante ressaltar que cada geração é feita com uma entrada de dados diferente, ou seja, para criar cada uma dessas imagens, o modelo de inteligência artificial recebe um "ruído" que variou para cada geração. Cada novo "ruído" é completamente aleatório em relação ao anterior, podendo se repetir. Porém, percebe-se no treinamento apresentado na Figura 37 a tendência, independente de qual for o "ruído", de ter-se uma imagem cinzenta.

Os treinamentos, mesmo contendo os mesmos parâmetros, podem diferenciar entre si, mas nenhum teste gerou algo próximo à azulejaria.

A variação dos resultados pode acontecer pela forma como as redes neurais são implementadas e treinadas (diz respeito as implementações internas dos *frameworks* e algoritmos consolidados para realizar os treinamentos). Há sempre uma aleatoriedade nos treinamentos, sejam nos caminhos internos que a rede percorre, ou nos lotes de imagens para

cada época de treinamento. Além disso é possível alterar esses parâmetros para aumentar ou diminuir essas aleatoriedades e influenciar nos resultados. O que pode ser constatado é que a mudança nos fatores da rede não influenciou para se obter um resultado mais próximo da realidade.

As variações nos parâmetros da DCGAN sejam nos tamanhos de filtros, do *kernel,* lote ou qualquer outro utilizado nas camadas de normalização e das funções de ativação são diversos e neste projeto trabalhou-se com alguns deles, contudo, como já constatado, nenhum deles apresentou mudanças significativas nas imagens que diz respeito a nitidez e semelhança com a azulejaria. Por esse motivo, não será detalhado todos as variações de parâmetros utilizadas, apenas aquelas que influenciaram na estabilidade da rede.

4.2 Estabilidade da rede neural

O que se pode perceber com as modificações dos parâmetros, foi a melhora na estabilidade do treinamento. É possível constatar isso nos arquivos de *log* gerados por cada treinamento e que regista o resultado das perdas, a cada época, das duas redes: discriminadora e geradora. É esperado que o valor da função de perda diminua a cada época. Nos primeiros treinamentos temos que os valores das funções de perda são bastante divergentes e variam muito, ora aumentando demais e voltando a diminuir, além de apresentar saltos enormes entre as épocas. Com o refinamento dos atributos foi possível estabilizar melhor os números da função de perda.

A Figura 38 ilustra sobre a estabilidade de treinamento. Na parte de cima da imagem temos o resultado das funções de perda do treinamento que deu origem ao módulo da Figura 35. Nota-se que na sequência do treinamento temos saldos para a perda da rede geradora que sai de 11 para 17, uma variação grande e que devia estar diminuindo e não aumentando. Sua rede discriminadora manifesta da mesma forma, porém com saltos menores entre épocas. Já a parte de baixo da figura temos o resultado do treinamento do modelo exemplificado na Figura 36, é notável a diferença. A perda do gerador varia muito pouco e tende a diminuir até o fim do treinamento. O mesmo aspecto é possível perceber em relação à rede discriminadora.

20250609113406

```
33  2025-06-09 11:40:41 -> Epoca 28 finalizada em 12.53 segundos. PERDA GERADOR: 11.591462135314941 >> PERDA DISCRIMINADOR: 0.27205854654312134
34  2025-06-09 11:40:56 -> Epoca 29 finalizada em 15.36 segundos. PERDA GERADOR: 11.696053504943848 >> PERDA DISCRIMINADOR: 0.2681112587451935
35  2025-06-09 11:41:09 -> Epoca 30 finalizada em 12.77 segundos. PERDA GERADOR: 11.704628944396973 >> PERDA DISCRIMINADOR: 0.2612975239753723
36  2025-06-09 11:41:25 -> Epoca 31 finalizada em 15.65 segundos. PERDA GERADOR: 11.828568458557129 >> PERDA DISCRIMINADOR: 0.25954243540763855
37  2025-06-09 11:41:38 -> Epoca 32 finalizada em 12.88 segundos. PERDA GERADOR: 11.881709098815918 >> PERDA DISCRIMINADOR: 0.256544321775436
38  2025-06-09 11:41:51 -> Epoca 33 finalizada em 12.61 segundos. PERDA GERADOR: 12.539179801940918 >> PERDA DISCRIMINADOR: 0.31658461689949036
39  2025-06-09 11:42:06 -> Epoca 34 finalizada em 15.70 segundos. PERDA GERADOR: 13.19170093536377 >> PERDA DISCRIMINADOR: 0.3448216021060936
40  2025-06-09 11:42:19 -> Epoca 35 finalizada em 12.91 segundos. PERDA GERADOR: 15.244467735290527 >> PERDA DISCRIMINADOR: 0.35075441002845764
41  2025-06-09 11:42:32 -> Epoca 36 finalizada em 12.67 segundos. PERDA GERADOR: 15.880849838256836 >> PERDA DISCRIMINADOR: 0.35273656249046326
42  2025-06-09 11:42:48 -> Epoca 37 finalizada em 12.82 segundos. PERDA GERADOR: 15.98486042022705 >> PERDA DISCRIMINADOR: 0.348666995763787
43  2025-06-09 11:43:01 -> Epoca 38 finalizada em 13.06 segundos. PERDA GERADOR: 16.08196258544922 >> PERDA DISCRIMINADOR: 0.34416642785072327
44  2025-06-09 11:43:17 -> Epoca 39 finalizada em 15.99 segundos. PERDA GERADOR: 16.140302658081055 >> PERDA DISCRIMINADOR: 0.33850255608558655
45  2025-06-09 11:43:30 -> Epoca 40 finalizada em 12.95 segundos. PERDA GERADOR: 16.051971435546875 >> PERDA DISCRIMINADOR: 0.3337625563144684
46  2025-06-09 11:43:45 -> Epoca 41 finalizada em 15.59 segundos. PERDA GERADOR: 16.201276779174805 >> PERDA DISCRIMINADOR: 0.331457972526550
47  2025-06-09 11:43:58 -> Epoca 42 finalizada em 12.65 segundos. PERDA GERADOR: 16.24922752380371 >> PERDA DISCRIMINADOR: 0.3286492824554443
48  2025-06-09 11:44:11 -> Epoca 43 finalizada em 12.93 segundos. PERDA GERADOR: 16.174531936645508 >> PERDA DISCRIMINADOR: 0.322929680347442
49  2025-06-09 11:44:27 -> Epoca 44 finalizada em 15.77 segundos. PERDA GERADOR: 16.600269317626953 >> PERDA DISCRIMINADOR: 0.340777426958084
50  2025-06-09 11:44:40 -> Epoca 45 finalizada em 12.95 segundos. PERDA GERADOR: 16.691129684448242 >> PERDA DISCRIMINADOR: 0.33701974153518677
51  2025-06-09 11:44:56 -> Epoca 46 finalizada em 15.92 segundos. PERDA GERADOR: 17.273447036743164 >> PERDA DISCRIMINADOR: 0.34380993247032166
52  2025-06-09 11:45:09 -> Epoca 47 finalizada em 12.94 segundos. PERDA GERADOR: 17.402658462524414 >> PERDA DISCRIMINADOR: 0.34417712688446045
53  2025-06-09 11:45:22 -> Epoca 48 finalizada em 13.10 segundos. PERDA GERADOR: 17.371883392333984 >> PERDA DISCRIMINADOR: 0.34090861678123474
54  2025-06-09 11:45:38 -> Epoca 49 finalizada em 16.34 segundos. PERDA GERADOR: 17.244226455688477 >> PERDA DISCRIMINADOR: 0.33535343408584595
55  2025-06-09 11:45:52 -> Epoca 50 finalizada em 13.50 segundos. PERDA GERADOR: 17.254159927368164 >> PERDA DISCRIMINADOR: 0.331488996744155
```

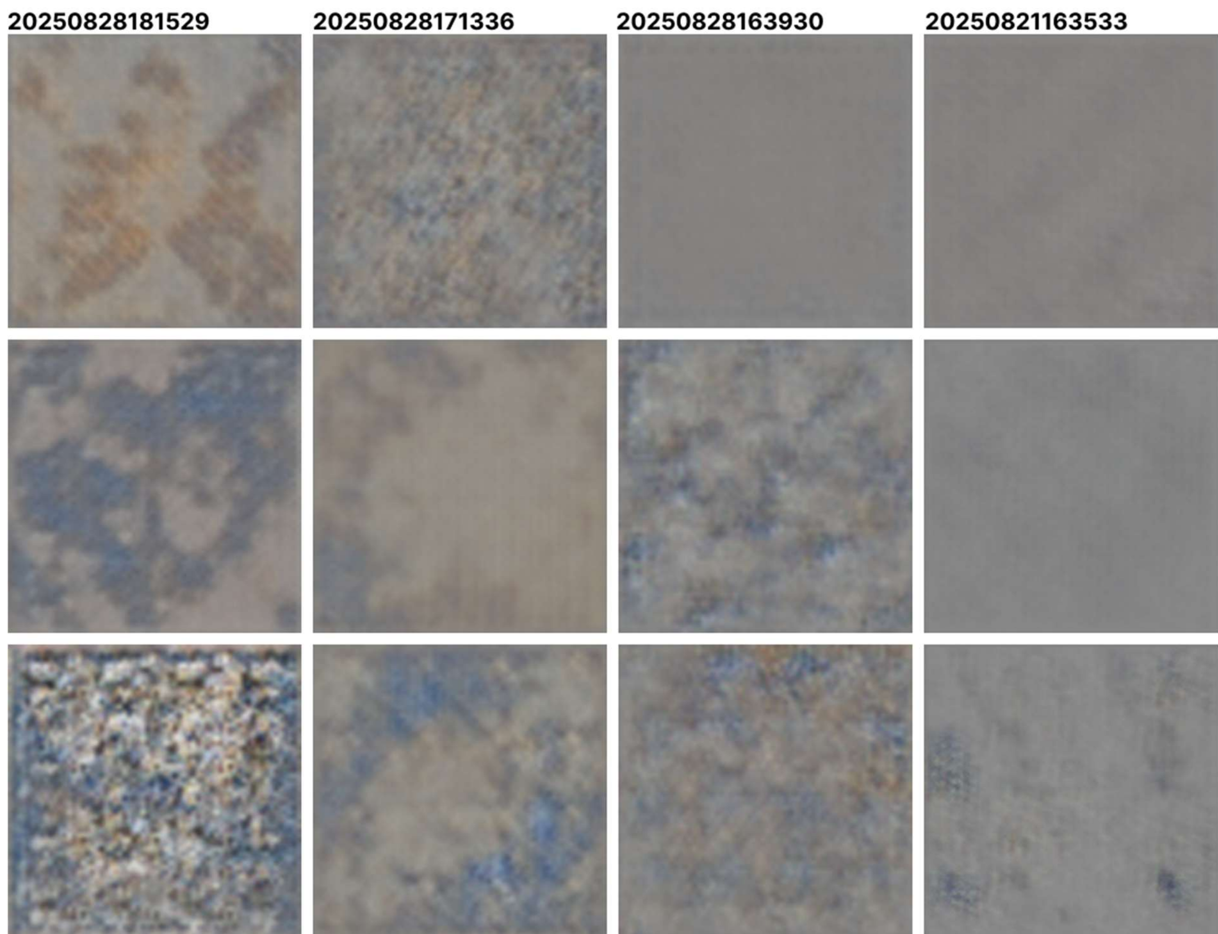
20250716112722

```
33  2025-07-16 11:32:16 -> Epoca 26 finalizada em 9.64 segundos. PERDA GERADOR: 1.4164140224456787 >> PERDA DISCRIMINADOR: 0.9323277473449707
34  2025-07-16 11:32:25 -> Epoca 27 finalizada em 9.62 segundos. PERDA GERADOR: 1.4082905054092407 >> PERDA DISCRIMINADOR: 0.9287759065628052
35  2025-07-16 11:32:38 -> Epoca 28 finalizada em 12.63 segundos. PERDA GERADOR: 1.407169222831726 >> PERDA DISCRIMINADOR: 0.9251105785369873
36  2025-07-16 11:32:48 -> Epoca 29 finalizada em 9.82 segundos. PERDA GERADOR: 1.3988717794418335 >> PERDA DISCRIMINADOR: 0.9265521764755249
37  2025-07-16 11:32:57 -> Epoca 30 finalizada em 9.70 segundos. PERDA GERADOR: 1.3925186395645142 >> PERDA DISCRIMINADOR: 0.9243875741958618
38  2025-07-16 11:33:10 -> Epoca 31 finalizada em 12.53 segundos. PERDA GERADOR: 1.3851187229156494 >> PERDA DISCRIMINADOR: 0.9243858456611633
39  2025-07-16 11:33:21 -> Epoca 32 finalizada em 10.57 segundos. PERDA GERADOR: 1.3751804828643799 >> PERDA DISCRIMINADOR: 0.9189302921295166
40  2025-07-16 11:33:30 -> Epoca 33 finalizada em 9.56 segundos. PERDA GERADOR: 1.3615479469299316 >> PERDA DISCRIMINADOR: 0.9216629266738892
41  2025-07-16 11:33:40 -> Epoca 34 finalizada em 9.85 segundos. PERDA GERADOR: 1.352121114730835 >> PERDA DISCRIMINADOR: 0.921660363674163
42  2025-07-16 11:33:53 -> Epoca 35 finalizada em 12.83 segundos. PERDA GERADOR: 1.3458720445632935 >> PERDA DISCRIMINADOR: 0.9188987016677856
43  2025-07-16 11:34:03 -> Epoca 36 finalizada em 9.86 segundos. PERDA GERADOR: 1.3405991792678833 >> PERDA DISCRIMINADOR: 0.9167277812957764
44  2025-07-16 11:34:13 -> Epoca 37 finalizada em 9.73 segundos. PERDA GERADOR: 1.3339251279830933 >> PERDA DISCRIMINADOR: 0.9144049286842346
45  2025-07-16 11:34:26 -> Epoca 38 finalizada em 12.78 segundos. PERDA GERADOR: 1.3237569332122803 >> PERDA DISCRIMINADOR: 0.9146428108215332
46  2025-07-16 11:34:35 -> Epoca 39 finalizada em 9.62 segundos. PERDA GERADOR: 1.3169593811035156 >> PERDA DISCRIMINADOR: 0.9116066694259644
47  2025-07-16 11:34:45 -> Epoca 40 finalizada em 9.70 segundos. PERDA GERADOR: 1.309334635734558 >> PERDA DISCRIMINADOR: 0.9100713729858398
48  2025-07-16 11:34:55 -> Epoca 41 finalizada em 10.19 segundos. PERDA GERADOR: 1.3028333187103271 >> PERDA DISCRIMINADOR: 0.9097495079040527
49  2025-07-16 11:35:06 -> Epoca 42 finalizada em 10.90 segundos. PERDA GERADOR: 1.2945833206176758 >> PERDA DISCRIMINADOR: 0.9106588959693909
50  2025-07-16 11:35:20 -> Epoca 43 finalizada em 13.57 segundos. PERDA GERADOR: 1.2865464687347412 >> PERDA DISCRIMINADOR: 0.9104419946670532
51  2025-07-16 11:35:30 -> Epoca 44 finalizada em 10.06 segundos. PERDA GERADOR: 1.280705451965332 >> PERDA DISCRIMINADOR: 0.9097228050231934
52  2025-07-16 11:35:40 -> Epoca 45 finalizada em 9.82 segundos. PERDA GERADOR: 1.2738001346588135 >> PERDA DISCRIMINADOR: 0.9081095457077026
53  2025-07-16 11:35:51 -> Epoca 46 finalizada em 11.83 segundos. PERDA GERADOR: 1.2671153545379639 >> PERDA DISCRIMINADOR: 0.9075570106506348
54  2025-07-16 11:36:02 -> Epoca 47 finalizada em 9.91 segundos. PERDA GERADOR: 1.2626006603240967 >> PERDA DISCRIMINADOR: 0.9054355025291443
55  2025-07-16 11:36:12 -> Epoca 48 finalizada em 10.23 segundos. PERDA GERADOR: 1.2548452615737915 >> PERDA DISCRIMINADOR: 0.9043969511985779
```

Fonte: Imagem criada pelo autor

Figura 38 – Comparação entre os resultados da função de perda de dois treinamentos

Apesar de se perceber melhora com a variação dos atributos na questão de cálculo das redes, novamente, não se conseguiu melhoras significativas em relação aos aspectos gráficos produzidos e todos os resultados geraram imagens aleatórias e abstratas. A estabilização da rede tendeu a resultados semelhantes que resultaram em imagens acinzentadas, mas que apresentam melhores texturas e uma predisposição para o azul (característica presente na azulejaria portuguesa). A Figura 39 mostra um comparativo entre modelos feitos com pequenas variações nos atributos usados, porém, com resultados muito semelhantes.



Fonte: Imagem criada pelo autor

Figura 39 – Comparativo entre os resultados de modelos estabilizados

A maior variação do resultado e sua estabilidade se apresentou quando é alterado os valores da função de otimização das redes. É na função de otimização que é definida a "velocidade" de aprendizado das redes, por isso é esperado que tanto a rede discriminadora, quanto a geradora, possuam optimizadores com valores próximos. Abaixo exemplo de função de otimização.

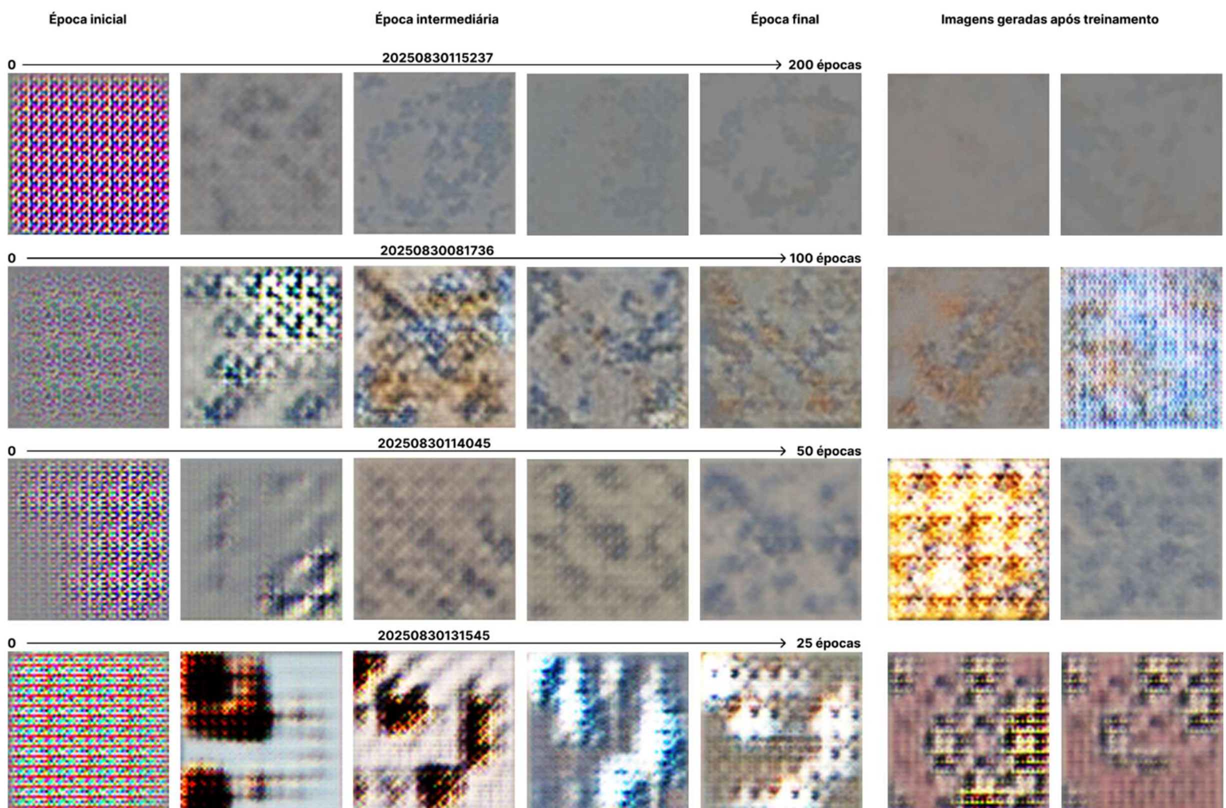
```
keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5)
```

Alterar os atributos das camadas das redes, como os parâmetros de *Dropout*, *BatchNoramalization*, *LeakReLU* etc, não pareceu criar grandes impactos nos resultados para este projeto, seja visualmente ou na questão de estabilidade.

Em testes, para se obter a estabilidade, a função de otimização pareceu se comportar melhor com o parâmetro de *learning_rate* tendo valores entre 0,001 à 0,0002 e o *beta_1* com valores inferiores a 0,5 até 0,1. O valor padrão para a função para o *beta_1* é de 0,9 e esse

valor também apresentou instabilidade para as redes. Os exemplos da Figura 39 tiveram todos a mesma configuração para o optimizador (*learning_rate* = 0,0002, *beta_1* = 0,5). Já o resultado da Figura 35 teve como optimizador um *learning_rate* de 0,001 e o *beta_1* padrão (0,9), a Figura 38, mostra como o parâmetro de *beta_1* influência no resultado, implicando em estabilidade quando acertado.

4.3 Influência da quantidade de épocas



Fonte: Imagem criada pelo autor

Figura 40 – Comparativo de treinamento relativo a quantidade de épocas para a implementação *dcgan.py*

A Figura 40 apresenta um comparativo de treinamentos baseados na quantidade de épocas de treinamento. Na gravura temos a evolução por épocas das imagens geradas pela DCGAN a medida que evolui em seu treinamento. Está identificado a quantidade de épocas para cada linha da imagem, bem como a referência para o *log* do treinamento. Todos os parâmetros foram mantidos variando apenas o "tempo" de treinamento.

É possível afirmar que todos os treinamentos, de certa forma, caminham para resultados que fixam os padrões de cores. No treinamento de 25 épocas vê-se uma tendência

de pixels vermelhos e amarelos e as demais em tons de cinza com azul. Os treinamentos de época 50 e 100 tem os melhores resultados, pois apresentam imagens menos “pixeladas” e maior variabilidade, já que, apesar de retornarem padrões de cinza, ainda conseguem gerar imagens diferentes. Ao contrário do treinamento de 200 épocas que possui nitidez, mas pouquíssima variabilidade.

Estes comportamentos são esperados e estão descritos na literatura que tratam de GAN/DCGAN (I. J. Goodfellow et al., 2014; Salimans et al., 2016). Um desses problemas é o que foi definido como *mode collapse,* que é o fato do gerador produzir amostras que se repetem uma vez que ele aprende que aquele padrão consegue “enganar” a rede discriminadora e assim repete sempre o seu resultado. Parece ser o caso nos exemplos de muitas épocas apresentado.

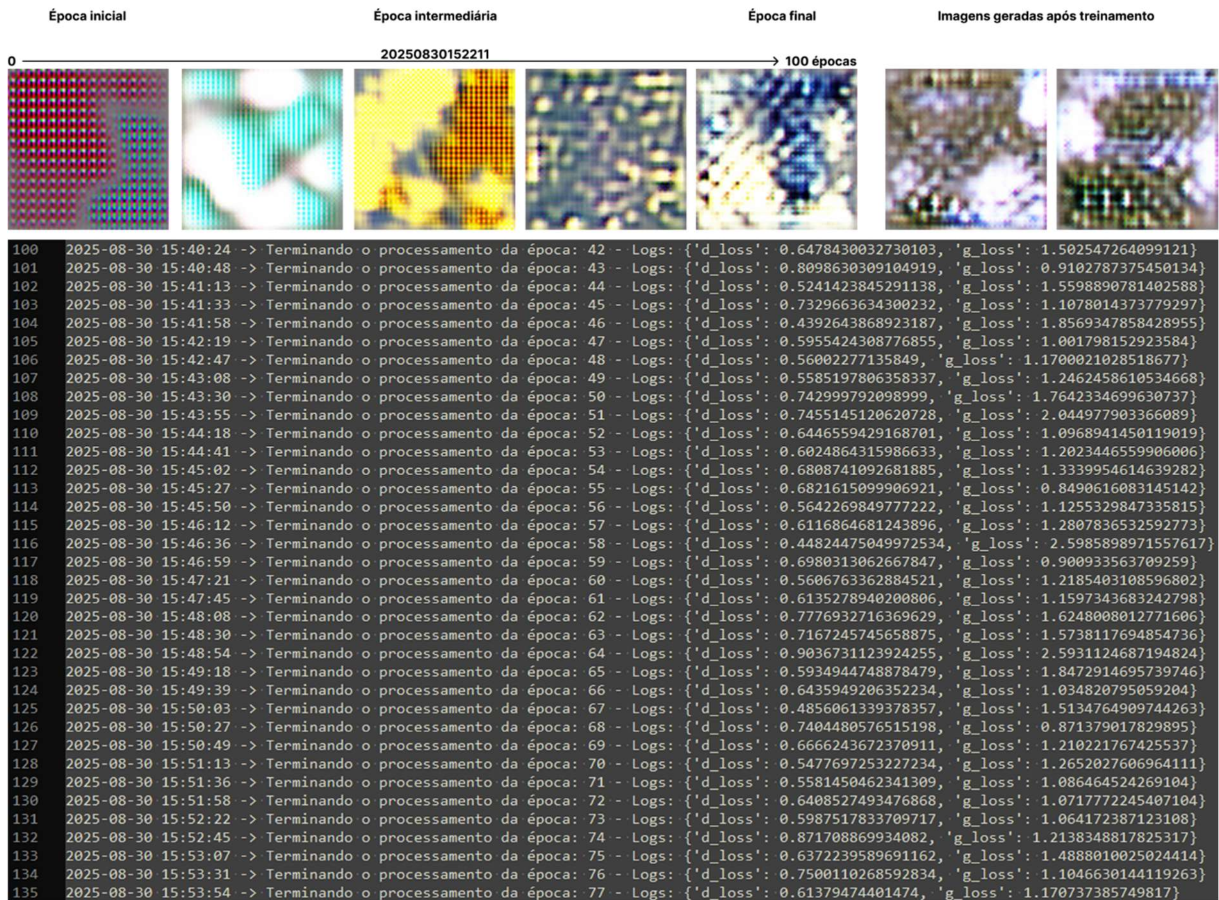
4.4 Comparativo empírico entre *dcgan.py* e *dcgan_keras3.py*

Toda a análise apresentada até aqui diz respeito a implementação original da classe DCGAN e que se encontra no arquivo *dcgan.py*. Na tentativa de validar a codificação, foi confeccionada uma segunda DCGAN, para fins comparativos. Esse novo teste tenta verificar a possibilidade de melhorar a qualidade das imagens com base em outra abordagem de implementação.

A classe GAN do arquivo *dcgan_keras3.py* foi criada a partir de exemplo[32] da documentação do próprio *framework* e sem maiores modificações, apenas o necessário para que fosse trocado os *datasets* e a forma como seria documentado os treinamentos.

A seguir a Figura 41 apresenta o treinamento feito com Keras 3. No topo é indicado as imagens geradas por épocas em um treinamento de um total de 100. Abaixo das gravuras há um *log* com os cálculos de perda das duas redes. O termo “g_loss” identifica a perda da rede geradora e “d_loss” da discriminadora. É visível que a implementação utilizando a *dcgan_keras3.py* não gerou um treinamento estável. A variação nos números do cálculo de perda é completamente aleatória e o resultado pode ser visto nas imagens. Cada época tem um padrão de cor diferente e no final a nitidez da imagem é baixa.

[32] https://keras.io/examples/generative/dcgan_overriding_train_step

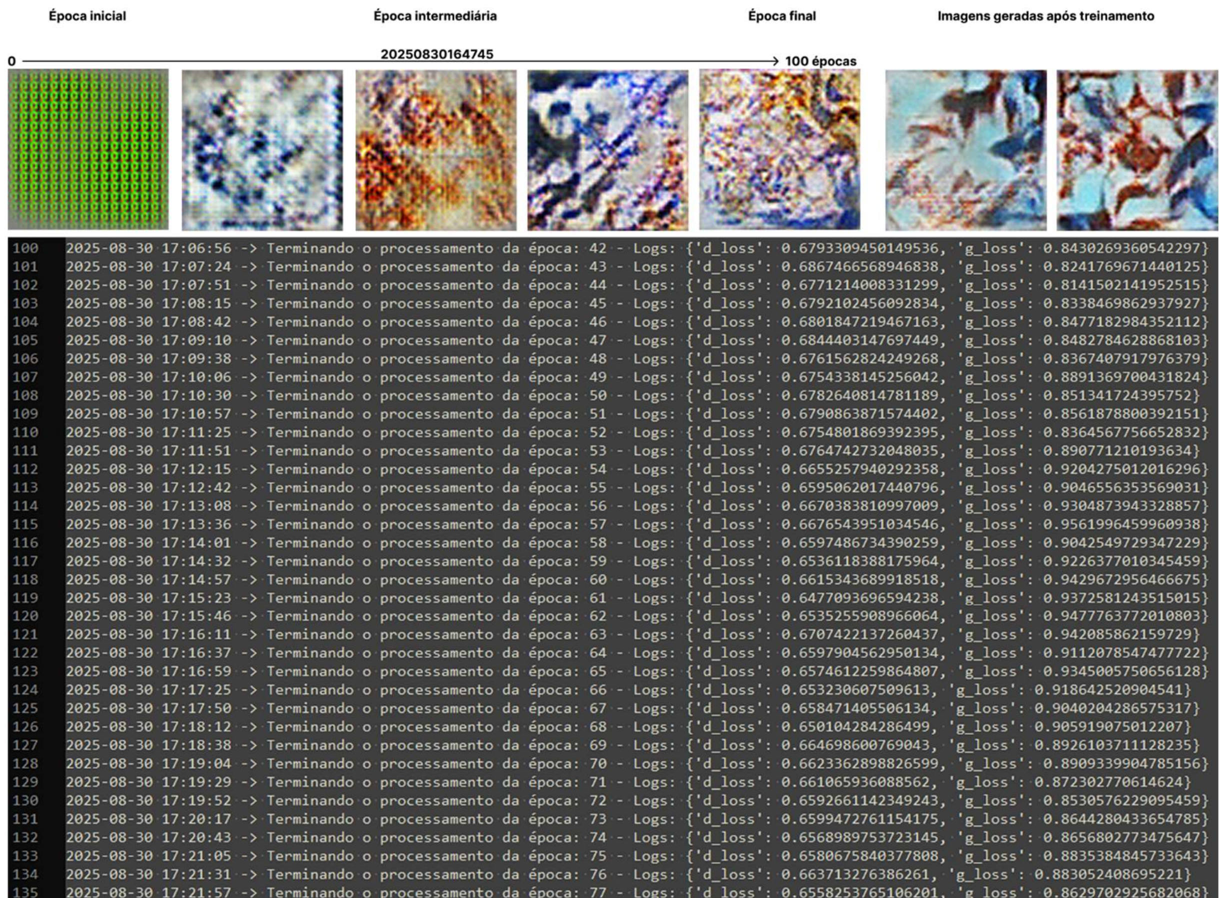



```
100  2025-08-30 15:40:24 -> Terminando o processamento da época: 42 - Logs: {'d_loss': 0.6478430032730103, 'g_loss': 1.502547264099121}
101  2025-08-30 15:40:48 -> Terminando o processamento da época: 43 - Logs: {'d_loss': 0.8098630309104919, 'g_loss': 0.9102787375450134}
102  2025-08-30 15:41:13 -> Terminando o processamento da época: 44 - Logs: {'d_loss': 0.5241423845291138, 'g_loss': 1.5598890781402588}
103  2025-08-30 15:41:33 -> Terminando o processamento da época: 45 - Logs: {'d_loss': 0.7329663634300232, 'g_loss': 1.1078014373779297}
104  2025-08-30 15:41:58 -> Terminando o processamento da época: 46 - Logs: {'d_loss': 0.4392643868923187, 'g_loss': 1.8569347858428955}
105  2025-08-30 15:42:19 -> Terminando o processamento da época: 47 - Logs: {'d_loss': 0.5955424308776855, 'g_loss': 1.001798152923584}
106  2025-08-30 15:42:47 -> Terminando o processamento da época: 48 - Logs: {'d_loss': 0.56002277135849, 'g_loss': 1.1700021028518677}
107  2025-08-30 15:43:08 -> Terminando o processamento da época: 49 - Logs: {'d_loss': 0.5585197806358337, 'g_loss': 1.2462458610534668}
108  2025-08-30 15:43:30 -> Terminando o processamento da época: 50 - Logs: {'d_loss': 0.742999792098999, 'g_loss': 1.7642334699630737}
109  2025-08-30 15:43:55 -> Terminando o processamento da época: 51 - Logs: {'d_loss': 0.7455145120620728, 'g_loss': 2.044977903366089}
110  2025-08-30 15:44:18 -> Terminando o processamento da época: 52 - Logs: {'d_loss': 0.6446559429168701, 'g_loss': 1.0968941450119019}
111  2025-08-30 15:44:41 -> Terminando o processamento da época: 53 - Logs: {'d_loss': 0.6024864315986633, 'g_loss': 1.2023446559906006}
112  2025-08-30 15:45:02 -> Terminando o processamento da época: 54 - Logs: {'d_loss': 0.6808741092681885, 'g_loss': 1.3339954614639282}
113  2025-08-30 15:45:27 -> Terminando o processamento da época: 55 - Logs: {'d_loss': 0.6821615099906921, 'g_loss': 0.8490616083145142}
114  2025-08-30 15:45:50 -> Terminando o processamento da época: 56 - Logs: {'d_loss': 0.5642269849777222, 'g_loss': 1.1255329847335815}
115  2025-08-30 15:46:12 -> Terminando o processamento da época: 57 - Logs: {'d_loss': 0.6116864681243896, 'g_loss': 1.2807836532592773}
116  2025-08-30 15:46:36 -> Terminando o processamento da época: 58 - Logs: {'d_loss': 0.44824475049972534, 'g_loss': 2.5985898971557617}
117  2025-08-30 15:46:59 -> Terminando o processamento da época: 59 - Logs: {'d_loss': 0.6980313062667847, 'g_loss': 0.900933563709259}
118  2025-08-30 15:47:21 -> Terminando o processamento da época: 60 - Logs: {'d_loss': 0.5606763362884521, 'g_loss': 1.2185403108596802}
119  2025-08-30 15:47:45 -> Terminando o processamento da época: 61 - Logs: {'d_loss': 0.6135278940200806, 'g_loss': 1.1597343683242798}
120  2025-08-30 15:48:08 -> Terminando o processamento da época: 62 - Logs: {'d_loss': 0.7776932716369629, 'g_loss': 1.6248008012771606}
121  2025-08-30 15:48:30 -> Terminando o processamento da época: 63 - Logs: {'d_loss': 0.7167245745658875, 'g_loss': 1.5738117694854736}
122  2025-08-30 15:48:54 -> Terminando o processamento da época: 64 - Logs: {'d_loss': 0.9036731123924255, 'g_loss': 2.5931124687194824}
123  2025-08-30 15:49:18 -> Terminando o processamento da época: 65 - Logs: {'d_loss': 0.5934944748878479, 'g_loss': 1.8472914695739746}
124  2025-08-30 15:49:39 -> Terminando o processamento da época: 66 - Logs: {'d_loss': 0.6435949206352234, 'g_loss': 1.034820795059204}
125  2025-08-30 15:50:03 -> Terminando o processamento da época: 67 - Logs: {'d_loss': 0.4856061339378357, 'g_loss': 1.5134764909744263}
126  2025-08-30 15:50:27 -> Terminando o processamento da época: 68 - Logs: {'d_loss': 0.7404480576515198, 'g_loss': 0.871379017829895}
127  2025-08-30 15:50:49 -> Terminando o processamento da época: 69 - Logs: {'d_loss': 0.6666243672370911, 'g_loss': 1.210221767425537}
128  2025-08-30 15:51:13 -> Terminando o processamento da época: 70 - Logs: {'d_loss': 0.5477697253227234, 'g_loss': 1.2652027606964111}
129  2025-08-30 15:51:36 -> Terminando o processamento da época: 71 - Logs: {'d_loss': 0.5581450462341309, 'g_loss': 1.086464524269104}
130  2025-08-30 15:51:58 -> Terminando o processamento da época: 72 - Logs: {'d_loss': 0.6408527493476868, 'g_loss': 1.0717772245407104}
131  2025-08-30 15:52:22 -> Terminando o processamento da época: 73 - Logs: {'d_loss': 0.5987517833709717, 'g_loss': 1.064172387123108}
132  2025-08-30 15:52:45 -> Terminando o processamento da época: 74 - Logs: {'d_loss': 0.871708869934082, 'g_loss': 1.2138348817825317}
133  2025-08-30 15:53:07 -> Terminando o processamento da época: 75 - Logs: {'d_loss': 0.6372239589691162, 'g_loss': 1.4888010025024414}
134  2025-08-30 15:53:31 -> Terminando o processamento da época: 76 - Logs: {'d_loss': 0.7500110268592834, 'g_loss': 1.1046630144119263}
135  2025-08-30 15:53:54 -> Terminando o processamento da época: 77 - Logs: {'d_loss': 0.61379474401474, 'g_loss': 1.170737385749817}
```

Fonte: Imagem criada pelo autor

Figura 41 – Treinamento feito com Keras 3, exemplo de imagens e resultado de perdas do discriminador e gerador por épocas

A função de otimização usada para essa implementação não define o parâmetro *beta_1*, já mencionado anteriormente. A fim de comparação foi feito mais um treinamento com a mesma otimização utilizada na classe DCGAN, ou seja, usando o *learnig_rate* igual a 0,0002 e *beta_1* com o valor de 0,5. Posteriormente será feito uma comparação das duas implementações com todos os parâmetros definidos exatamente igual: número de camadas, filtros, *kernel etc.*



Fonte: Imagem criada pelo autor

Figura 42 - Treinamento feito com Keras 3, exemplo de imagens e resultado de perdas do discriminador e gerador por épocas. Parâmetro beta_1 = 0,5.

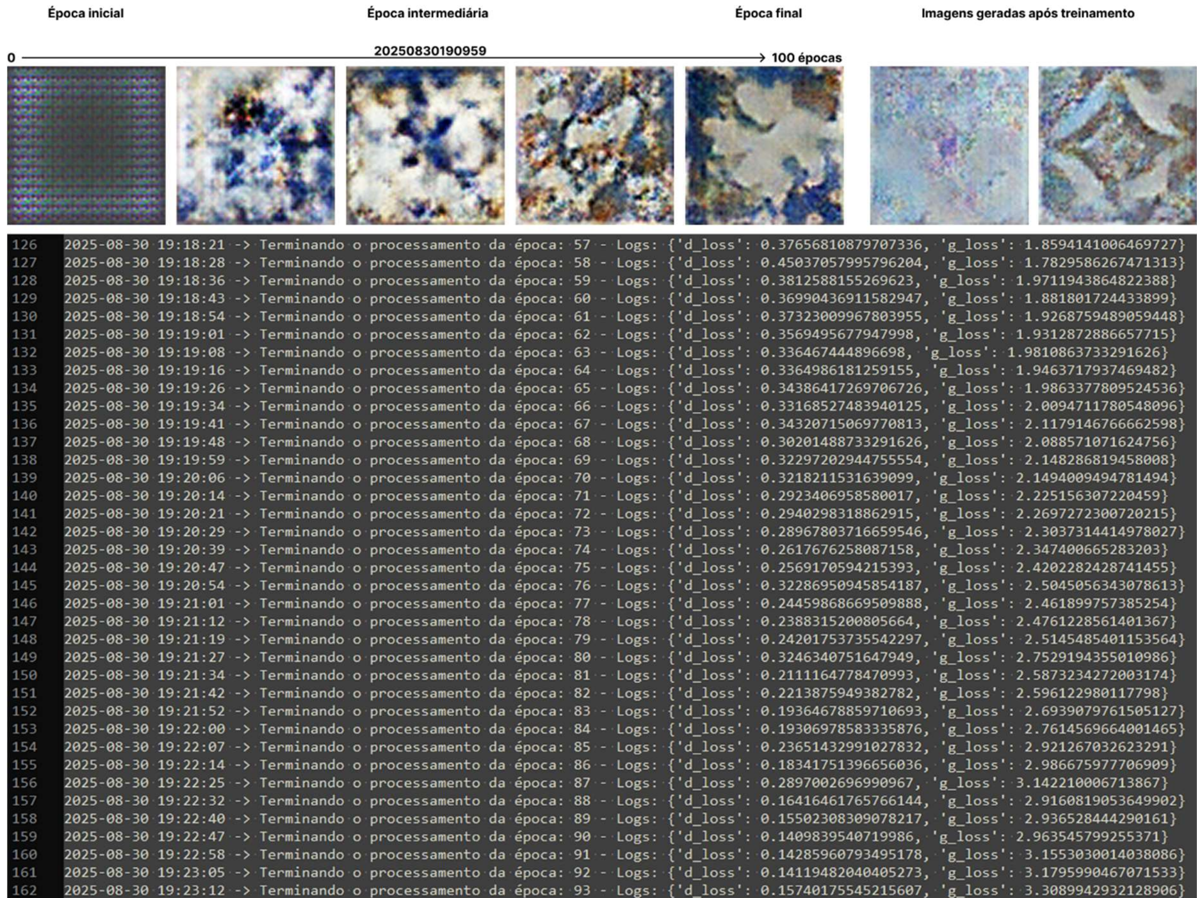
Após firmar os parâmetros do optimizador, fica visível a diferença no treinamento. A Figura 42 apresenta o treinamento igual ao da Figura 41, mas com o optimizador configurado conforme o exposto.

```
keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5)
```

O valor do cálculo da função de perda é estabilizado para ambas as redes e segue a tendência de diminuição para o discriminador. As imagens tornam-se mais nítidas e convergem para algo mais próximo de um azulejo, apesar de ainda representar uma imagem abstrata.

Se mantivermos a arquitetura igual nas duas implementações (*dcgan.py* e *dcgan_keras3.py*) os resultados modificam-se um pouco. Quando observada o cálculo da função de perda para o gerador, com a nova arquitetura, houve uma grande divergência ao

final do treinamento para 100 épocas. A Figura 43 apresenta os resultados. Nas imagens geradas pelo treinamento é possível perceber uma aproximação com texturas que remetem a azulejaria portuguesa, apesar de abstratas. O principal problema aqui, se comparamos com o resultado da Figura 42, é apenas o fato da instabilidade ao final do treinamento da rede geradora. Acontecem saltos grandes que variam de 1,85 a 3,30. Contudo em questões gráficas, ambos os treinamentos resultaram de forma imaterial e com alguma clareza.




```
126 2025-08-30 19:18:21 -> Terminando o processamento da época: 57 - Logs: {'d_loss': 0.37656810879707336, 'g_loss': 1.8594141006469727}
127 2025-08-30 19:18:28 -> Terminando o processamento da época: 58 - Logs: {'d_loss': 0.45037057995796204, 'g_loss': 1.7829586267471313}
128 2025-08-30 19:18:36 -> Terminando o processamento da época: 59 - Logs: {'d_loss': 0.3812588155269623, 'g_loss': 1.9711943864822388}
129 2025-08-30 19:18:43 -> Terminando o processamento da época: 60 - Logs: {'d_loss': 0.36990436911582947, 'g_loss': 1.881801724433899}
130 2025-08-30 19:18:54 -> Terminando o processamento da época: 61 - Logs: {'d_loss': 0.37323009967803955, 'g_loss': 1.9268759489059448}
131 2025-08-30 19:19:01 -> Terminando o processamento da época: 62 - Logs: {'d_loss': 0.3569495677947998, 'g_loss': 1.9312872886657715}
132 2025-08-30 19:19:08 -> Terminando o processamento da época: 63 - Logs: {'d_loss': 0.336467444896698, 'g_loss': 1.9810863733291626}
133 2025-08-30 19:19:16 -> Terminando o processamento da época: 64 - Logs: {'d_loss': 0.3364986181259155, 'g_loss': 1.9463717937469482}
134 2025-08-30 19:19:26 -> Terminando o processamento da época: 65 - Logs: {'d_loss': 0.34386417269706726, 'g_loss': 1.9863377809524536}
135 2025-08-30 19:19:34 -> Terminando o processamento da época: 66 - Logs: {'d_loss': 0.33168527483940125, 'g_loss': 2.0094711780548096}
136 2025-08-30 19:19:41 -> Terminando o processamento da época: 67 - Logs: {'d_loss': 0.34320715069770813, 'g_loss': 2.1179146766662598}
137 2025-08-30 19:19:48 -> Terminando o processamento da época: 68 - Logs: {'d_loss': 0.30201488733291626, 'g_loss': 2.08857071624756}
138 2025-08-30 19:19:59 -> Terminando o processamento da época: 69 - Logs: {'d_loss': 0.32297202944755554, 'g_loss': 2.148286819458008}
139 2025-08-30 19:20:06 -> Terminando o processamento da época: 70 - Logs: {'d_loss': 0.3218211531639099, 'g_loss': 2.1494009494781494}
140 2025-08-30 19:20:14 -> Terminando o processamento da época: 71 - Logs: {'d_loss': 0.2923406958580017, 'g_loss': 2.225156307220459}
141 2025-08-30 19:20:21 -> Terminando o processamento da época: 72 - Logs: {'d_loss': 0.2940298318862915, 'g_loss': 2.2697272300720215}
142 2025-08-30 19:20:29 -> Terminando o processamento da época: 73 - Logs: {'d_loss': 0.28967803716659546, 'g_loss': 2.3037314414978027}
143 2025-08-30 19:20:39 -> Terminando o processamento da época: 74 - Logs: {'d_loss': 0.2617676258087158, 'g_loss': 2.347400665283203}
144 2025-08-30 19:20:47 -> Terminando o processamento da época: 75 - Logs: {'d_loss': 0.2569170594215393, 'g_loss': 2.4202282428741455}
145 2025-08-30 19:20:54 -> Terminando o processamento da época: 76 - Logs: {'d_loss': 0.32286950945854187, 'g_loss': 2.5045056343078613}
146 2025-08-30 19:21:01 -> Terminando o processamento da época: 77 - Logs: {'d_loss': 0.24459868669509888, 'g_loss': 2.461899757385254}
147 2025-08-30 19:21:12 -> Terminando o processamento da época: 78 - Logs: {'d_loss': 0.2388315200805664, 'g_loss': 2.4761228561401367}
148 2025-08-30 19:21:19 -> Terminando o processamento da época: 79 - Logs: {'d_loss': 0.24201753735542297, 'g_loss': 2.5145485401153564}
149 2025-08-30 19:21:27 -> Terminando o processamento da época: 80 - Logs: {'d_loss': 0.3246340751647949, 'g_loss': 2.7529194355010986}
150 2025-08-30 19:21:34 -> Terminando o processamento da época: 81 - Logs: {'d_loss': 0.2111164778470993, 'g_loss': 2.5873234272003174}
151 2025-08-30 19:21:42 -> Terminando o processamento da época: 82 - Logs: {'d_loss': 0.2213875949382782, 'g_loss': 2.596122980117798}
152 2025-08-30 19:21:52 -> Terminando o processamento da época: 83 - Logs: {'d_loss': 0.19364678859710693, 'g_loss': 2.6939079761505127}
153 2025-08-30 19:22:00 -> Terminando o processamento da época: 84 - Logs: {'d_loss': 0.19306978583335876, 'g_loss': 2.7614569664001465}
154 2025-08-30 19:22:07 -> Terminando o processamento da época: 85 - Logs: {'d_loss': 0.23651432991027832, 'g_loss': 2.921267032623291}
155 2025-08-30 19:22:14 -> Terminando o processamento da época: 86 - Logs: {'d_loss': 0.18341751396656036, 'g_loss': 2.986675977706909}
156 2025-08-30 19:22:25 -> Terminando o processamento da época: 87 - Logs: {'d_loss': 0.2897002696990967, 'g_loss': 3.142210006713867}
157 2025-08-30 19:22:32 -> Terminando o processamento da época: 88 - Logs: {'d_loss': 0.16416461765766144, 'g_loss': 2.9160819053649902}
158 2025-08-30 19:22:40 -> Terminando o processamento da época: 89 - Logs: {'d_loss': 0.15502308309078217, 'g_loss': 2.936528444290161}
159 2025-08-30 19:22:47 -> Terminando o processamento da época: 90 - Logs: {'d_loss': 0.1409839540719986, 'g_loss': 2.963545799255371}
160 2025-08-30 19:22:58 -> Terminando o processamento da época: 91 - Logs: {'d_loss': 0.14285960793495178, 'g_loss': 3.1553030014038086}
161 2025-08-30 19:23:05 -> Terminando o processamento da época: 92 - Logs: {'d_loss': 0.14119482040405273, 'g_loss': 3.1795990467071533}
162 2025-08-30 19:23:12 -> Terminando o processamento da época: 93 - Logs: {'d_loss': 0.15740175545215607, 'g_loss': 3.3089942932128906}
```

Fonte: Imagem criada pelo autor

Figura 43 – Treinamento feito com Keras 3, exemplo de imagens e resultado de perdas do discriminador e gerador por épocas. Parâmetros iguais ao do treinamento feito com a classe do arquivo *dcgan.py*.

5 CONCLUSÃO

Este projeto foi feito para compreensão e experimentação em cima de um tema atual e que tem pautado tanto a academia quanto o debate público. A Inteligência Artificial é assunto recorrente em diversas esferas e é importante pois impacta as vidas das pessoas como poucos outros temas. A tentativa do trabalho é de desmistificar a IA e “brincar” de forma a validá-la, sem entrar em questões matemáticas e computacionais aprofundadas, como ferramenta de cunho artístico dentro do campo de Medias Digitais Interativas.

Observa-se que o resultado não correspondeu integralmente ao esperado, mas trouxe exemplos interessantes. Os treinamentos não geraram módulos de azulejos iguais aos reais, apesar de em alguns momentos sermos capazes de enxergar texturas e formas que remetem vagamente as geometrias das fotos que compõe o *dataset*. Apesar da subjetividade das gravuras criadas pela máquina, há nas gerações cores e formas que podem ser utilizadas artisticamente em contextos interativos e para criar padrões.

Considerando as implementações feitas, percebe-se que a *dcgan_keras3.py* trouxe cores mais vivas e suavidade na forma e, portanto, tem-se um melhor resultado. Contudo, nenhum dos treinamentos conseguiu fidelidade ao real. A efetividade que é vendida para as IAs não foi confirmada neste trabalho.

Se compararmos o treinamento de faces de seres humanos, exemplo bastante comum que pode ser visto em trabalhos académicos e outras fontes de internet, tem-se que a imagem de rostos tem um padrão bem definido. Há o contorno dos olhos, da boca, do nariz etc. Neste projeto não há isso, as fotos dos azulejos são diversas. Pode ter algo que pareça com uma flor, traços retos e geométricos, cores diversas, traços arredondados e desenhos completamente diferentes entre si. Considerando que uma rede neural é uma representação matemática para um padrão “universal” de uma amostra de dados, talvez seja por isso que o resultado não tenha conseguido chegar mais próximo da realidade. Soma-se ao exposto, questões de *hardware*, volume de dados de entrada etc.

Apesar do resultado, as capacidades de treinamento de uma inteligência artificial não foram esgotadas. Há outros algoritmos de GAN’s que poderiam ser aplicados para o proposto neste trabalho, técnicas para mitigação de falhas, (Salimans et al., 2016) métodos para classificar os dados de entrada ou a possibilidade de treinar redes específicas para padrões de

azulejos específicos, entre outras perspectivas que podem ser aproveitadas em trabalhos futuros e que tratem sobre o mesmo tema.

REFERÊNCIAS BIBLIOGRÁFICAS

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, *8*(1). https://doi.org/10.1186/s40537-021-00444-8

Ankile, L. L., Heggland, M. F., & Krange, K. (2020). *Deep Convolutional Neural Networks: A survey of the foundations, selected improvements, and some current applications*. http://arxiv.org/abs/2011.12960

Areán-García, N. (2009, September). Breve Histórico da Península Ibérica. *Revista Philologus*, 25–48.

Arntz, M., Gregory, T., & Zierahn, U. (2017). Revisiting the risk of automation. *Economics Letters*, *159*, 157–160. https://doi.org/10.1016/j.econlet.2017.07.001

Barata, G. S. G., Nascimento, I. T. do, Santos, F. P. M. dos, Santos, A. C. de S. G. dos, & Santos, G. N. (2021). A IMPORTÂNCIA DOS ACELERADORES DE HARDWARE EM PROJETOS QUE USAM INTELIGÊNCIA ARTIFICIAL. *Revista Eletrônica Perspectivas Da Ciência e Tecnologia - ISSN: 1984-5693*, *13*. https://doi.org/10.22407/1984-5693.2021.v13.p.27-34

Chakraborty, T., S, U. R. K., Naik, S. M., Panja, M., & Manvitha, B. (2023). *Ten Years of Generative Adversarial Nets (GANs): A survey of the state-of-the-art*. http://arxiv.org/abs/2308.16316

Chen, Z., Wang, T., Cai, H., Mondal, S. K., & Sahoo, J. P. (2022). BLB-gcForest: A High-Performance Distributed Deep Forest With Adaptive Sub-Forest Splitting. *IEEE Transactions on Parallel and Distributed Systems*, *33*(11), 3141–3152. https://doi.org/10.1109/TPDS.2021.3133544

Conceição, V. S., Nunes, E. M., & Rocha, A. M. (2020). O Reconhecimento Facial como uma das Vertentes da Inteligência Artificial (IA): um estudo de prospecção tecnológica. *Cadernos de Prospecção*, *13*(3), 745. https://doi.org/10.9771/cp.v13i3.32818

Gomes, D. dos S. (2010, August). Inteligência Artificial: Conceitos e Aplicações. *Revista Olhar Científico – Faculdades Associadas de Ariquemes – V. 01, n.2*, 234–246.

Gonog, L., & Zhou, Y. (2019). A Review: Generative Adversarial Networks. *14th IEEE Conference on Industrial Electronics and Applications (ICIEA 2019): 19-21 June 2019, Xi'an, China*, 505–510.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Convolutional Networks. In *Deep Learning* (pp. 326–366). MIT Press. https://www.deeplearningbook.org

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). *Generative Adversarial Networks*. http://arxiv.org/abs/1406.2661

Hancock, J. T., & Khoshgoftaar, T. M. (2020). Survey on categorical data for neural networks. *Journal of Big Data*, *7*(1). https://doi.org/10.1186/s40537-020-00305-w

Haykin, S. (2001). *Redes Neurais: Princípios e práticas* (2nd ed.).

Howley, D. (2023, February 22). There's no going back on A.I.: "The genie is out of the bottle." *Yahoo Finance Tech*. https://finance.yahoo.com/news/theres-no-going-back-on-ai-the-genie-is-out-of-the-bottle-193456914.html

IBM. (n.d.). *O que é um algoritmo de aprendizado de máquina?* Retrieved January 19, 2025, from https://www.ibm.com/br-pt/topics/machine-learning-algorithms

Legg, S., & Hutter, M. (2007). Universal intelligence: A definition of machine intelligence. *Minds and Machines*, *17*(4), 391–444. https://doi.org/10.1007/s11023-007-9079-x

Levinbook, M. (2008). *Design de superfície: técnicas e processos em estamparia têxtil para produção industrial* [Universidade Anhembi Morumbi]. https://livros01.livrosgratis.com.br/cp129496.pdf

Mahesh, B. (2019). Machine Learning Algorithms - A Review. *International Journal of Science and Research (IJSR)*, *9*(1), 381–386. https://doi.org/10.21275/art20203995

Maia Filho, M. S., & Junquilho, T. A. (2018). Projeto Victor: perspectivas de aplicação da inteligência artificial ao direito. *Revista de Direitos e Garantias Fundamentais*, *19*(3), 218–237. https://doi.org/10.18759/rdgf.v19i3.1587

Nicolelis, M. (2023, June 12). Inteligência artificial: tudo que você precisa saber - Miguel Nicolelis - Programa 20 Minutos [Broadcast]. In *Youtube - Opera Mundi*. https://www.youtube.com/live/pb4b4_MlNwo

Nunes, S. M. B. R. da G. (2014). *Azulejos de Padrão e Relevo — Uma Proposta Infográfica* [Universidade de Lisboa]. https://repositorio.ul.pt/handle/10451/11513

Open AI. (2025). *ChatGPT (Versão 5)*. https://chatgpt.com

Pergamon Museum. (n.d.). *Leão andante da Avenida Processional*. Retrieved January 18, 2025, from https://www.smb.museum/en/museums-institutions/pergamonmuseum/exhibitions/detail/ancient-near-eastern-cultures/

Priberam. (n.d.). Superfície. In *Priberam*. Retrieved January 18, 2025, from https://dicionario.priberam.org/superf%C3%ADcie

Radford, A., Metz, L., & Chintala, S. (2015). *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. http://arxiv.org/abs/1511.06434

Raley, R., & Rhee, J. (2023). Critical AI: A Field in Formation. *American Literature*, *95*(2), 185–204. https://doi.org/10.1215/00029831-10575021

Rauber, T. W. (2005). *Redes Neurais Artificiais*. https://www.researchgate.net/publication/228686464

Ribeiro, R. A., & Matos, M. L. de. (2022). Inteligência artificial forte como sujeito de direito e a ética por trás de seu desenvolvimento. In *Open Science Research IX* (pp. 908–926). Editora Científica Digital. https://doi.org/10.37885/221211308

Ridler, A. (2018). *Myriad (Tulips)*. http://annaridler.com/myriad-tulips

Ridler, A. (2019). *Mosaic Virus*. https://annaridler.com/mosaic-virus

Russell, S. J., & Norvig, P. (2013). *Inteligência Artificial* (3rd ed.). Elsevier.

Ruthschilling, E. A. (2002). *Design de superfície: prática e aprendizagem mediadas pela tecnologia digital* [Universidade Federal do Rio Grande do Sul]. https://lume.ufrgs.br/handle/10183/131159

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., & Chen, X. (2016). *Improved Techniques for Training GANs*. http://arxiv.org/abs/1606.03498

Schwartz, A. R. D. (2008). *Design de superfície: por uma visão projetual geométrica e tridimensional*. Universidade Estadual Paulista.

Solà, M. C. (2022). The Age of Intelligent Reproduction and Machine Learning Creativity. In R. Kelomees, V. Guljajeva, & O. Laas (Eds.), *The meaning of creativity in the age of AI* (pp. 59–70). Estonian Academy of Arts.

Solanki, U. (2024, December 31). *Understanding the Perceptron: A Foundation for Machine Learning Concepts*. Lucent Innovation. https://www.lucentinnovation.com/blogs/technology-posts/understanding-the-perceptron

Souza, M. M. De. (2019). *Azulejaria portuguesa: conexões entre moda, design de superfície e estamparia têxtil* [Universidade de São Paulo]. https://www.teses.usp.br/teses/disponiveis/100/100133/tde-17062019-114845/publico/DISSERTACAOMATHEUSSOUZA.pdf

Teixeira, P. M. (2023, February 1). O ChatGPT e os desafios às universidades. *Público*. https://www.publico.pt/2023/02/01/opiniao/opiniao/chatgpt-desafios-universidades-2037314