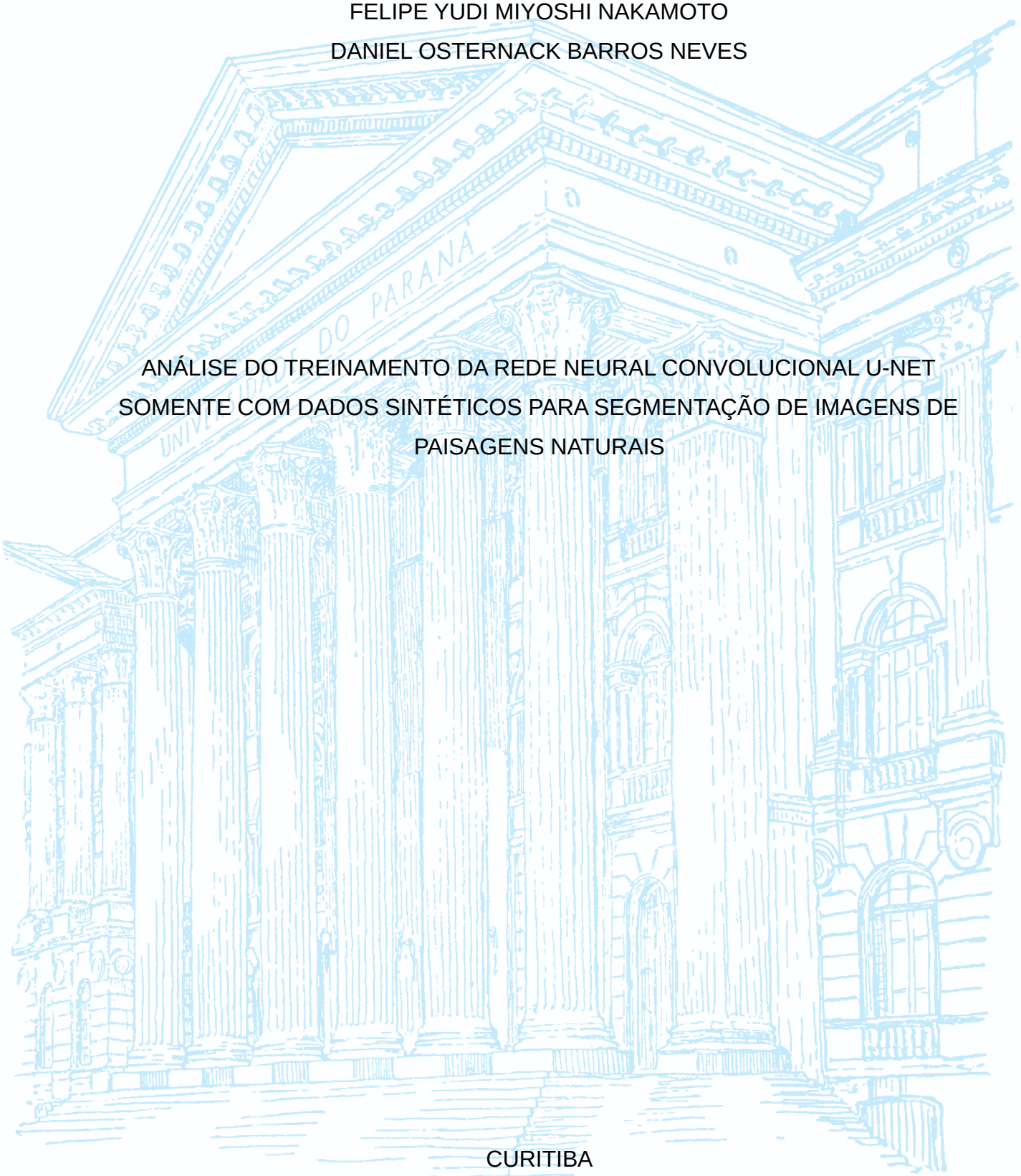
UNIVERSIDADE FEDERAL DO PARANÁ

FELIPE YUDI MIYOSHI NAKAMOTO

DANIEL OSTERNACK BARROS NEVES



ANÁLISE DO TREINAMENTO DA REDE NEURAL CONVOLUCIONAL U-NET SOMENTE COM DADOS SINTÉTICOS PARA SEGMENTAÇÃO DE IMAGENS DE PAISAGENS NATURAIS

CURITIBA

2021

FELIPE YUDI MIYOSHI NAKAMOTO
DANIEL OSTERNACK BARROS NEVES

ANÁLISE DO TREINAMENTO DA REDE NEURAL CONVOLUCIONAL U-NET SOMENTE COM DADOS SINTÉTICOS PARA SEGMENTAÇÃO DE IMAGENS DE PAISAGENS NATURAIS

Trabalho apresentado como requisito parcial para a conclusão do Curso de Bacharelado em Ciência da Computação, setor de Ciências Exatas da Universidade Federal do Paraná.

Orientador: Prof. Dr. Lucas Ferrari de Oliveira

CURITIBA
2021

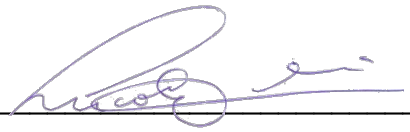
TERMO DE APROVAÇÃO

DANIEL OSTERNACK BARROS NEVES

FELIPE YUDI MIYOSHI NAKAMOTO

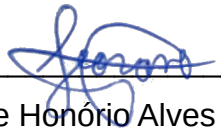
ANÁLISE DO TREINAMENTO DA REDE NEURAL CONVOLUCIONAL U-NET SOMENTE COM DADOS SINTÉTICOS PARA SEGMENTAÇÃO DE IMAGENS DE PAISAGENS NATURAIS

TCC apresentado ao Curso de Graduação em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação.



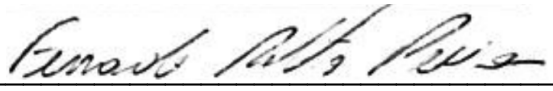
Prof. Dr. Lucas Ferrari de Oliveira

Orientador – Departamento de Informática, UFPR



Msc. Jeovane Honório Alves

Doutorando PPGInf, UFPR



Msc. Fernando Roberto Pereira

Professor IFSC Câmpus Canoinhas

Curitiba, 20 de agosto de 2021.

AGRADECIMENTOS

Gostaríamos de agradecer ao nosso orientador, Professor Dr. Lucas Ferrari de Oliveira, pelo direcionamento, colaboração e paciência durante a construção desta pesquisa.

Às nossas famílias pelo apoio e incentivo, mesmo nas ocasiões mais difíceis.

Aos colegas de curso, pelo companheirismo e convivência durante todo o período da graduação.

Aos colaboradores e professores do Departamento de Informática, que foram essenciais para o nosso processo de formação.

À todos aqueles que contribuíram, de alguma forma, para a realização deste trabalho.

RESUMO

A utilização de dados sintéticos é uma alternativa que vem se tornando cada vez mais comum para treinamento de algoritmos de *machine learning*. Fatores como a dificuldade na coleta, escassez de amostras reais, direitos autorais e leis de proteção à privacidade como a LGPD e GDPR motivaram novas pesquisas nesse segmento. No estado da arte não é incomum encontrar trabalhos que utilizam os dados sintéticos como opção para aumentar *datasets* pré-existentes. Contudo, entendemos que ainda existe espaço para explorar a viabilidade de abordagens com bases totalmente sintéticas, sem amostras reais. Neste trabalho, temos como objetivo avaliar o desempenho de um modelo de rede neural convolucional treinado exclusivamente com dados artificiais no reconhecimento de elementos da natureza em paisagens reais. Para isso, implementamos uma arquitetura U-Net que recebe como entrada figuras de cenários criados sinteticamente e máscaras de segmentação semântica que identificam as classes existentes. Dessa forma, após o treinamento, a rede passa a ser capaz de segmentar paisagens que de fato existem na natureza. Com os resultados alcançados concluímos que, mesmo com um *dataset* relativamente pequeno, o algoritmo teve um desempenho satisfatório, obtendo uma acurácia média de 0,8 e IoU médio de 0,673 entre 10 execuções.

Palavras-chave: Inteligência Artificial. Aprendizagem Profunda. Processamento de Imagens. Rede Neural Convolucional. Paisagem. Segmentação. Dados Sintéticos. Redes Adversárias Generativas.

ABSTRACT

The usage of synthetic data has become increasingly common in the training of machine learning algorithms. Factors such as gathering difficulty, lack of real data, copyright and privacy protection laws such as LGPD and GDPR have motivated new research in this segment. It is not infrequent to find state-of-the-art papers using synthetic data as an option to expand pre-existing datasets. However, we understand that there is still space to explore the feasibility of approaches that rely solely on synthetic databases, using no real data. In this work, we aim to assess the performance of a convolutional neural network model trained exclusively with artificial data in the recognition of natural elements in real sceneries. For that, we implement a U-net architecture that receives as input synthetically-created landscape images and their corresponding semantic segmentation masks that identify the existing classes. After training, the network becomes capable of segmenting landscapes that truly exist in nature. With the achieved results we conclude that, even with a relatively small dataset, the algorithm had a satisfactory performance, obtaining a mean accuracy of 0,8 and mean IoU of 0,673 among 10 runs.

Keywords: Artificial Intelligence. Deep Learning. Digital Image Processing. Convolutional Neural Network. Landscape. Segmentation. Synthetic Data. Generative Adversarial Networks.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS OU SIGLAS

ANNs	- *Artificial Neural Networks*
CNN	- *Convolutional Neural Network*
CPU	- *Central Process Unit*
GAN	- *Generative Adversarial Network*
GDPR	- *General Data Protection Regulation*
GPU	- *Graphics Processing Unit*
IA	- Inteligência Artificial
IDE	- *Integrated Development Environment*
IoU	- *Intersection Over Union*
JPG	- *Joint Photographic Expert Group*
JSON	- *Javascript Object Notation*
LGPD	- Lei Geral da Proteção de Dados
PNG	- *Portable Network Graphics*
RAM	- *Random Access Memory*

SUMÁRIO

1 INTRODUÇÃO

Diante de um período dominado pela presença de algoritmos de *machine learning* nos mais diversos segmentos da sociedade, os dados se tornaram uma matéria-prima essencial, sendo até mesmo considerados “o novo petróleo na era moderna da inteligência artificial” (ANDREWS, 2021). Nesse contexto, em métodos de aprendizagem profunda, uma das tarefas necessárias para treinar uma rede neural artificial é a obtenção de um *dataset* que engloba o cenário desejado. Considerando aplicações convencionais, como classificação de dígitos manuscritos, essa atividade pode ser trivial, visto que existem inúmeras bases públicas disponíveis na internet, como por exemplo o MNIST[1] (LECUN *et al.*, 1998), para a aplicação citada. Entretanto, como apontado por Nikolenko (2019), boa parte dos problemas dessa era moderna da inteligência artificial estão relacionados com dados insuficientes, seja pela quantidade precária de amostras nos *datasets* ou pela dificuldade do processo manual de anotação em tarefas de aprendizagem supervisionada.

Uma das abordagens para resolver este problema está na utilização de dados sintéticos, que, de acordo com Andrews (2021), são informações anotadas geradas a partir de processos computacionais para servirem como alternativa aos dados do mundo real. Por serem rotuladas, as amostras sintéticas são menos custosas, já que dispensam a necessidade da anotação manual, que demanda tempo e dinheiro. Além disso, outro benefício proporcionado por essa alternativa é a quebra de obstáculos relacionados à privacidade e proteção de dados, que estão cada vez mais rigorosos com a ascenção da área de segurança da informação.

Nesse contexto, o trabalho apresentará um modelo de rede neural convolucional treinado, apenas com imagens geradas por outra rede convolucional, para verificação do reconhecimento de imagens reais. No caso, utilizaremos imagens de paisagens naturais geradas sinteticamente na plataforma GauGAN para segmentar fotos de paisagens verídicas, que estão presentes no mundo real.

1 http://yann.lecun.com/exdb/mnist/

1.1 JUSTIFICATIVA

De maneira geral, a aptidão de um modelo de inteligência artificial está diretamente ligada a diversidade e qualidade do *dataset* utilizado no treinamento. Dependendo do problema, a quantidade de elementos necessários pode passar da casa de milhões, tornando inviável a obtenção de dados reais. Além disso, com o crescimento da proteção de privacidade pessoal por leis tais como a LGPD, a obtenção de dados e sua utilização para treinamento de redes neurais tornam-se tarefas cada vez mais árduas e burocráticas.

Dados sintéticos apresentam-se, portanto, como uma alternativa plausível para tal situação, sendo capazes de reproduzir o conteúdo presente no mundo real de modo mais acessível.

1.2 OBJETIVOS

Nesta seção serão apresentados os objetivos que desejamos alcançar com o presente trabalho. O objetivo geral resume, de forma ampla, a ação principal que será abordada. Na sequência, os objetivos específicos detalham os passos necessários para atingirmos os resultados desejados.

1.2.1 Objetivo geral

O presente trabalho tem como objetivo geral avaliar o desempenho de uma rede neural convolucional na classificação de elementos da natureza em paisagens reais, quando treinada apenas com imagens sintéticas.

1.2.2 Objetivos específicos

Como objetivos específicos, temos:

a) Gerar um *dataset* de treino com imagens artificiais através do software GauGAN, que transforma mapas de segmentação em imagens sintéticas realistas;

b) Realizar o treinamento da rede neural convolucional U-Net com os dados obtidos por meio do GauGAN;
c) Elaborar uma base de dados para testes com imagens reais, criando máscaras de segmentação para estas;
d) Verificar, com os resultados obtidos, se é factível a realização do treinamento da rede neural com dados sintéticos.

1.3 ESTRUTURA DE CAPÍTULOS

No Capítulo 1 são tratados os tópicos de introdução do trabalho, com a contextualização do problema de pesquisa, justificativa, que motivou a realização do mesmo, e os objetivos alcançados.

O Capítulo 2 compreende a fundamentação teórica utilizada para o desenvolvimento do trabalho, bem como, a descrição das ferramentas no contexto da aplicação.

O Capítulo 3 mostra a metodologia adotada durante o desenvolvimento, abordando as ferramentas utilizadas, restrições estabelecidas para a criação do *dataset* e a estrutura do algoritmo.

No Capítulo 4 apresentamos os resultados obtidos com as execuções de treinamento do modelo. Além disso, é feita uma análise dos possíveis motivos que levaram aos valores obtidos.

Por fim, no Capítulo 5, são esclarecidas as considerações finais sobre o trabalho, citando recomendações para possíveis abordagens futuras.

2 REVISÃO DE LITERATURA

Este capítulo aborda os assuntos utilizados como embasamento teórico para a produção desta pesquisa. A partir da compreensão dos temas que estão dispostos nas seções a seguir, foi possível desenvolver e refinar o modelo que fornece os resultados que definimos em nossos objetivos.

2.1 APRENDIZADO DE MÁQUINA

Machine Learning ou Aprendizagem de Máquina é um campo da Inteligência Artificial que se baseia na adaptação dinâmica do processo decisório de um programa autônomo a partir de mudanças no ambiente no qual está inserido (HURWITZ; KIRSCH, 2018). Ao contrário da abordagem clássica, em que decisões são fixas, tomadas a partir de condições explicitamente codificadas na aplicação, na aprendizagem de máquina o programa desenvolve suas escolhas conforme novas informações lhe são disponibilizadas.

O conceito base da aprendizagem de máquina é a detecção de padrões significativos entre dados de entrada, que não poderiam ser facilmente identificados e codificados manualmente por um programador, seja por mera complexidade ou total inexequibilidade (SHALEV-SHWARTZ; BEN-DAVID, 2014). Há quatro principais divisões da área (MAHESH, 2020):

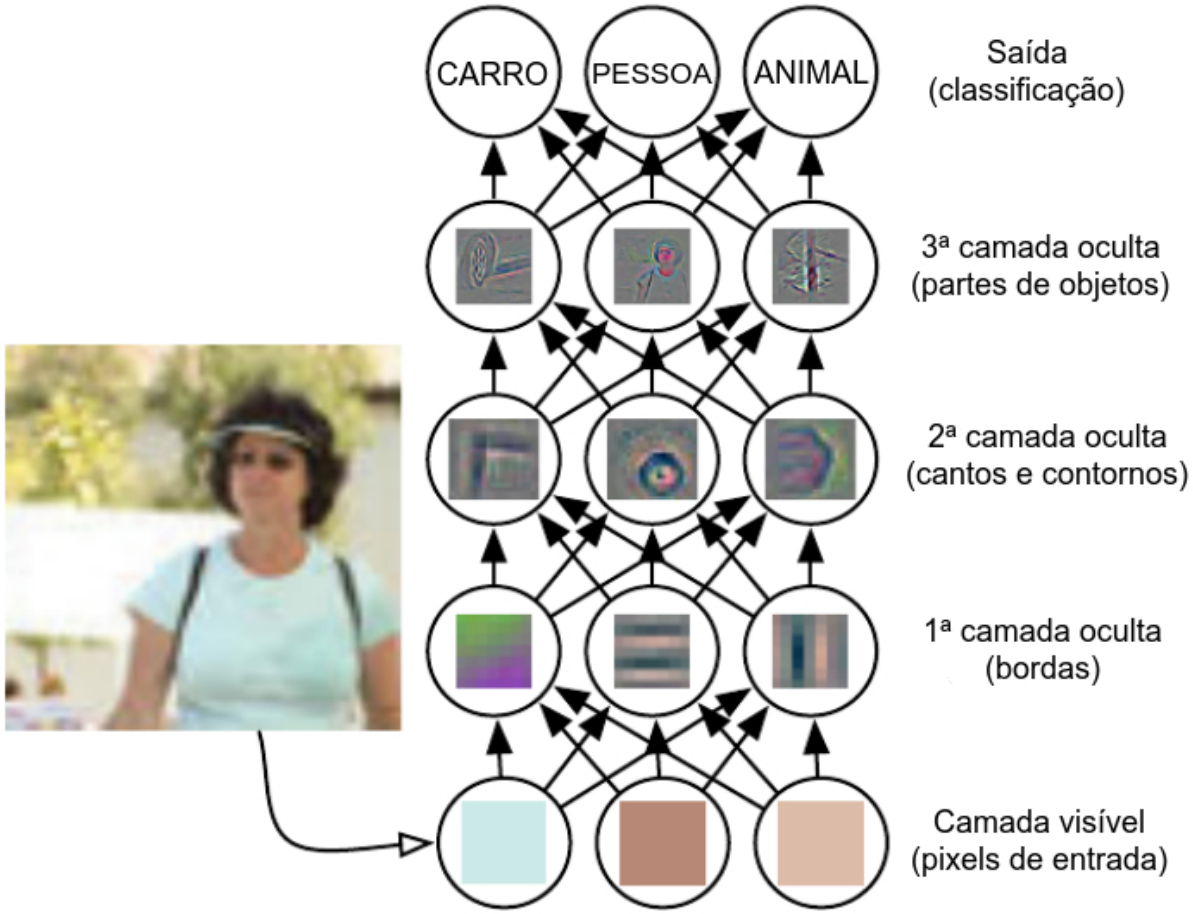
- Aprendizagem supervisionada: Consiste em encontrar uma função de mapeamento de determinadas entradas a determinadas saídas, com base em um conjunto de dados de entrada manualmente rotulados (conjunto de pares em formato lógico equivalente a “dado de entrada => saída esperada”)
- Aprendizagem não-supervisionada: Consiste no aprendizado de padrões entre dados de entrada não-rotulados. Seus resultados são primariamente análises estruturais dos dados, de forma que tais algoritmos são usados majoritariamente para *clustering* (agrupamento de dados com características semelhantes) e redução de características.

- Aprendizagem semi-supervisionada: Uma combinação dos dois métodos anteriores, para situações em que apenas parte da base de dados está rotulada. Um exemplo é o *self-training*, em que a IA, após ser treinada com dados rotulados, adiciona dados rotulados por si mesma à base de treinamento.
- Aprendizagem por reforço: Diferentemente das outras abordagens, esta técnica aprende por tentativa e erro, interagindo com o ambiente ao invés de receber uma base de dados de entrada (HURWITZ; KIRSCH, 2018). A cada decisão não-satisfatória, o programa recebe uma “punição”, e a cada decisão adequada, uma “recompensa”.
-

Ainda dentro do campo de aprendizagem de máquina, temos um método mais específico de aprendizado que consiste na identificação e criação de padrões complexos entre os dados de entrada a partir de partes mais simples. Essa estratégia é conhecida como *Deep Learning* ou Aprendizagem Profunda, sendo seu nome derivado das diversas camadas de extração de características que aumentam de complexidade a cada nível. É aplicada principalmente em problemas de natureza trivial para humanos, resolvidos de forma intuitiva e automática, como reconhecimento facial ou interpretação de fala, mas que são desafiadores para computadores por serem complexos de definir formalmente (GOODFELLOW *et al*., 2016).

A estrutura base de um algoritmo de aprendizagem profunda é composta por uma camada de entrada e uma de saída, com um número variável de camadas intermediárias, denominadas “*hidden layers”* (camadas ocultas). A cada camada, são aplicadas transformações sobre as saídas da camada anterior, de forma a identificar determinadas características. Tais características são combinadas na camada seguinte, resultando em conceitos mais específicos. A última camada combina estas informações e apresenta um resultado de alto nível, seja este uma descrição da entrada, uma classificação, uma pontuação ou predição (FIGURA 1).

FIGURA 1 – REPRESENTAÇÃO VISUAL DA ESTRUTURA DE UMA IA DE DEEP LEARNING



FONTE: Adaptado de Zeiler e Fergus (2014) citado por Goodfellow *et al*. (2016).

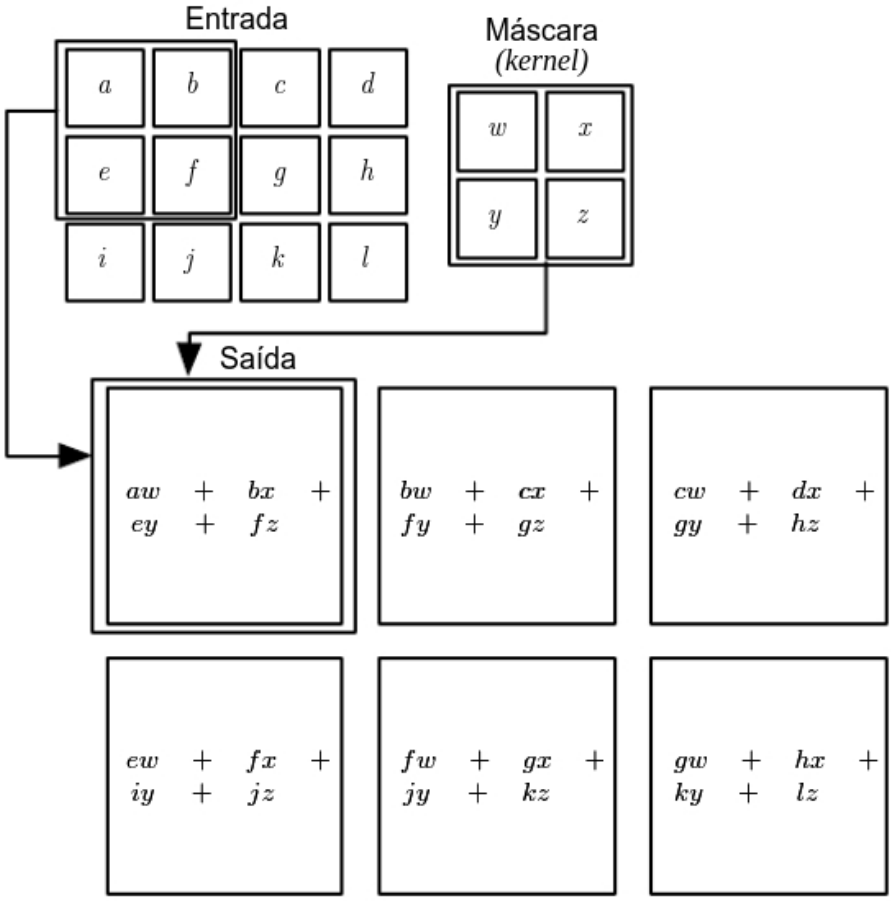
Entre as aplicações da aprendizagem profunda, uma das mais amplamente utilizadas atualmente é a de Redes Neurais Artificiais. Baseadas no conceito de neurônios biológicos e aprendizado entre seres vivos, as ANNs abstraem o funcionamento do cérebro humano, utilizando uma rede de nodos interligados (equivalentes a neurônios), cada qual com determinado “peso” utilizado para a o cálculo da saída a partir das características de entrada. Conforme exemplos são apresentados à rede, os pesos são modificados de forma a minimizar o erro na saída. O mais antigo exemplo de uma rede neural artificial com ajuste automático de pesos é o *perceptron* (ROSENBLATT, 1958, 1962 citado por GOODFELLOW *et al*., 2016).

2.1.1 Redes Neurais Convolucionais

Dentre as redes neurais artificiais, existe ainda o conceito de Redes Neurais Convolucionais, especializadas na extração de informação a partir de dados com estrutura matricial, como imagens (GOODFELLOW *et al*., 2016). Popularizadas por Yann LeCun entre as décadas de 80 e 90 (LECUN, 1989, citado por GOODFELLOW *et al*., 2016), as CNNs se aproveitam da ideia de “filtros” do campo de processamento de imagens (denominados “*kernels*”) para extração de características a partir de dados crus de entrada, destacando atributos como linhas verticais, horizontais, bordas e outros padrões. Em níveis mais profundos de uma rede convolucional, estes são combinados para formar padrões mais complexos, como olhos ou bocas, no contexto de reconhecimento facial, por exemplo.

O conceito base das redes convolucionais é a aplicação de operações matemáticas denominadas convoluções sobre a entrada, em que cada posição da matriz inicial é multiplicada pela posição equivalente em uma matriz de pesos intermediária (o *kernel* ou máscara), os produtos são somados e inseridos em uma matriz de saída. A máscara então desliza por toda a matriz de entrada, repetindo a operação, até que todas as posições da matriz de saída sejam preenchidas, conforme representado pela Figura 2. A matriz resultante do processo é chamada de *feature map* ou mapa de características (LECUN *et al*., 1998).

FIGURA 2 – REPRESENTAÇÃO VISUAL DA OPERAÇÃO DE CONVOLUÇÃO



FONTE: Adaptado de Goodfellow *et al*. (2016).

Além das operações de convolução, o uso de camadas de *pooling* e *fully connected layers* também é comum entre as CNNs. O primeiro utiliza uma ideia similar à máscara previamente citada, tomando como entrada uma seção da matriz inicial e produzindo como saída um número posicionado correspondentemente em uma matriz de saída. Alguns exemplos desta operação são o *max-pooling*, que seleciona o maior número entre os existentes na matriz de entrada, e o *average-pooling*, que calcula a média destes valores. Tomando como exemplo a figura 2 e a operação *max-pooling* com tamanho 2x2, o número resultante da célula indicada como saída seria o maior entre “a”, “b”, “e” e “f”.

Já o conceito de *fully connected layers,* ou camadas totalmente conectadas, consiste em camadas de neurônios em que cada um é conectado a todos os neurônios da camada anterior, por meio da aplicação de uma função de pesos. Tal estratégia também é conhecida como *Multilayer Perceptron* e é frequentemente utilizada nas últimas camadas de CNNs, de forma a mapear as características extraídas a saídas que equivalem às informações processadas, como classes (GOODFELLOW *et al*., 2016).

O aprendizado destas redes ocorre da mesma forma que previamente explicado para as redes neurais artificiais: são aplicadas funções de adequação para cada valor das matrizes de filtro (máscaras), de forma que o erro na saída seja reduzido. Esta operação é denominada de *back-propagation* (GOODFELLOW *et al*., 2016).

2.1.1.1 U-Net

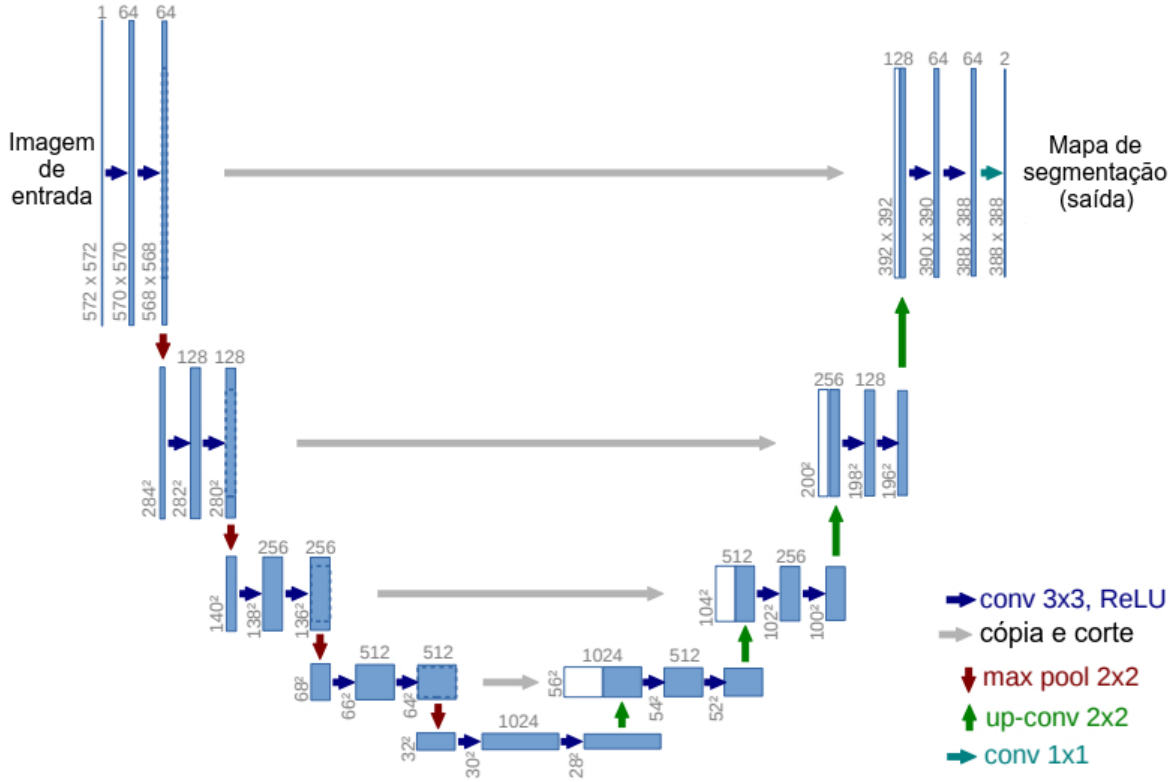
A U-Net é uma rede neural convolucional apresentada por Ronneberger *et al.* (2015) para segmentação de imagens biomédicas, criada com o intuito de melhorar a localização espacial (classificação por pixel) do processo. É uma rede neural totalmente convolucional (*fully convolutional network*), em que as camadas totalmente conectadas são substituídas por operações de convolução 1x1, e seu conceito base é o de “suplementar uma rede usual de contração com camadas sucessivas, em que os operadores de *pooling* são substituídos por *upsampling*” (RONNEBERGER *et al.,* 2015). Além disso, mapas de características resultantes de convoluções e *poolings* anteriores são combinados com os resultados do *upsampling*, aumentando a precisão da localização.

A arquitetura da U-Net consiste em um caminho de redução e outro de expansão da imagem de entrada, denominados *downstack* (ou *downsampling path*) e *upstack* (ou *upsampling path*). Neste primeiro, são aplicadas duas convoluções 3x3 seguidas a cada nível da rede, e uma função de ativação ReLU. Entre cada nível também é executada, em um primeiro momento, uma redução da imagem de entrada por meio de operações de *max-pooling* 2x2 com *stride* 2 (número de células que a máscara se move conforme desliza sobre a matriz de entrada). A arquitetura

apresentada por Ronneberger *et al.* (2015) é composta por 5 níveis que envolvem estas operações, tendo o menor a resolução de 32x32.

Na segunda metade da rede, as operações de *max-pool* são então substituídas por operações de *upsampling*, que aumentam a resolução da matriz resultante, combinando canais de características. A matriz resultante de cada *max-pool* do *downsampling path* é cortada e concatenada à matriz resultante da operação de *up-convolution* (equivalente a *upsampling*) no mesmo nível. Com isso, obtém-se uma estrutura semelhante a um “U” (Figura 3), da qual deriva-se o nome da rede.

FIGURA 3 – ARQUITETURA DA REDE U-NET



FONTE: Adaptado de Ronneberger *et al*. (2015).

2.1.2 Redes Adversárias Generativas

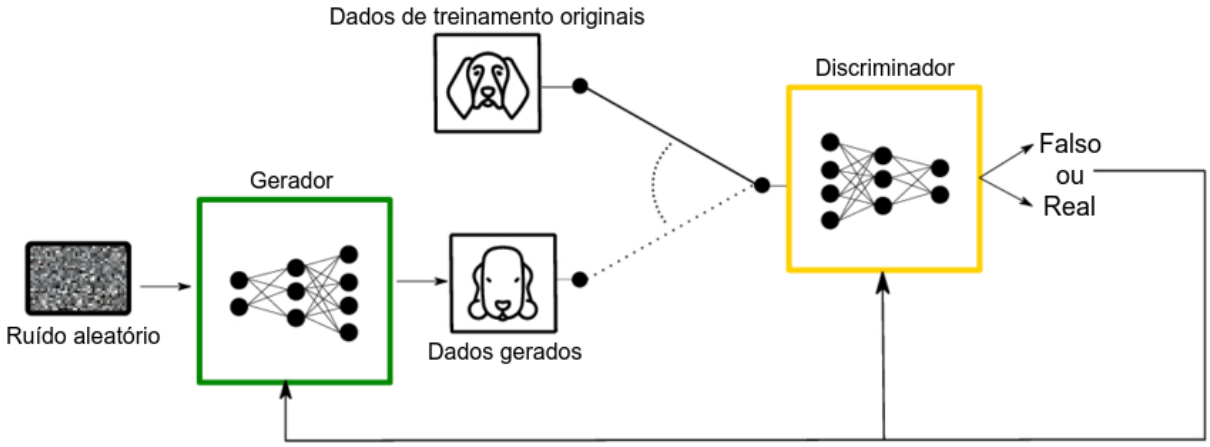
As Redes Adversárias Generativas são uma arquitetura para treinamento de modelos generativos introduzidas por Goodfellow *et al*. (2014).

De acordo com os autores, a ideia principal é que duas redes neurais sejam treinadas em um contexto competitivo, seguindo a essência de um jogo soma zero, onde o ganho de um indivíduo representa perda para o outro.

Uma das redes que compõem a GAN é denominada de Generativa. Como o nome sugere, ela é responsável por produzir novos exemplos sintéticos no domínio do problema. A entrada da rede generativa é um vetor aleatório, que ao passar pelo modelo é transformado na saída artificial, que pode ser uma imagem, áudio, vídeo, entre outros. A segunda rede, conhecida como Discriminativa, tem a tarefa de identificar se os dados de saída criados na rede generativa são verdadeiros ou falsos. Para isso, inicialmente o modelo é treinado com dados reais para aprender o que de fato é uma entrada verdadeira e então poder recusar os dados que não fazem parte do conjunto.

A Figura 4 ilustra o fluxo de eventos na GAN, que caracteriza a disputa entre o Gerador e o Discriminador. Nesse cenário, conforme a classificação (real ou falso) emitida pela rede discriminativa, um dos modelos deve ser atualizado. Quando uma imagem falsa é considerada verdadeira, o discriminador precisa atualizar seus parâmetros com o objetivo de maximizar a probabilidade da classificação correta, enquanto o gerador é recompensado pela capacidade de enganar o adversário. Em contrapartida, caso as imagens falsas estejam sendo detectadas, é necessário atualizar o gerador para que este passe a criar exemplos mais próximos a realidade.

FIGURA 4 – ARQUITETURA DE REDE ADVERSÁRIA GENERATIVA



FONTE: Adaptado de Barrios *et al*. (2019).

Como explicado por Goodfellow *et al*. (2014), essa situação pode ser interpretada de maneira análoga a um falsificador de dinheiro tentando enganar a polícia. No caso, a rede generativa seria o falsificador, que precisa aprimorar suas técnicas para criar a moeda falsa. Por outro lado, a rede discriminativa (polícia), tem o objetivo de identificar se o dinheiro é legítimo. Essa competição continua até que o falsificador passe a produzir saídas indistinguíveis às originais.

2.1.3 Data Augmentation

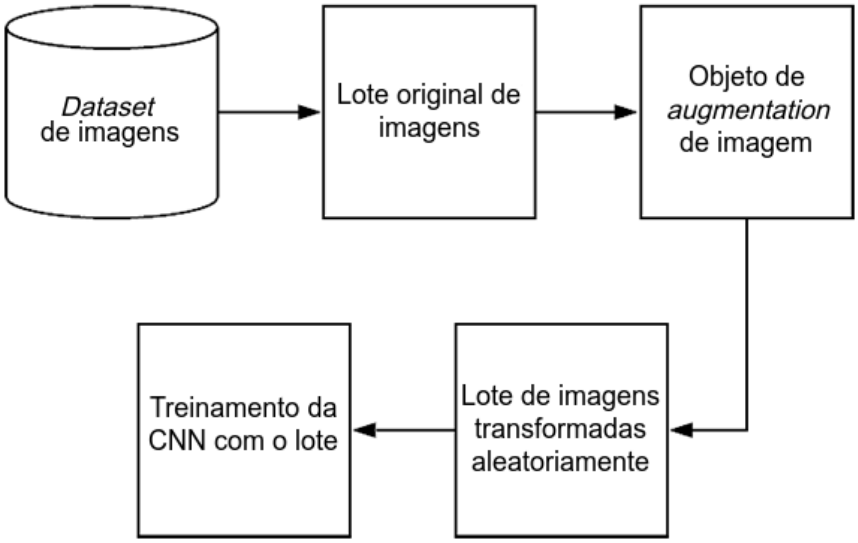
Os dados de entrada utilizados em modelos de aprendizado de máquina consistem em representações de um cenário por meio de um conjunto limitado de amostras. Com o objetivo de expandir o espaço de variações e melhorar a representação da realidade na base de dados, o *data augmentation* é utilizado para a criação de novos dados a partir de pequenas mudanças em amostras pré-existentes (ROSEBROCK, 2019).

Essa técnica auxilia na redução do problema de *overfitting*, quando o modelo se ajusta excessivamente aos casos específicos apresentados no treinamento e perde precisão para situações novas, que não foram vistas anteriormente (DVORNIK *et al*., 2018, p. 1). Em circunstâncias nas quais coletar e classificar novos dados exigem um esforço impraticável, o mecanismo também atua como facilitador, promovendo o aumento na base de dados sem custos adicionais.

Dentre as maneiras de se aplicar o *data augmentation*, as mais populares são conhecidas como *offline augmentation* e *augmentation on-the-fly.* De acordo com Rosebrock (2019), a primeira opção ocorre quando os novos dados são adicionados no conjunto pré-existente, ou seja, a quantidade de elementos no *dataset* aumenta com a inclusão das amostras modificadas. Essa alternativa é interessante para situações onde o conjunto de entrada original é pequeno, pois possibilita que os dados sejam multiplicados inúmeras vezes.

O augmentation on-the-fly é recomendado para conjuntos de dados maiores, onde não é possível aumentá-los por limitações de memória e/ou disco (ROSEBROCK, 2019). Como ilustrado na Figura 5, essa variação realiza as transformações durante o processo de treinamento, onde cada *batch* recebe amostras aleatórias com *augmentation*, sem alterar o *dataset* original. Portanto, o modelo passa a ser treinado apenas com o resultado do *data augmentation.*

FIGURA 5 – AUGMENTATION ON-THE-FLY PARA *DATASET* DE IMAGENS



FONTE: Adaptado de Rosebrock (2019).

2.1.3.1 Aplicação em imagens

No contexto de redes neurais convolucionais que utilizam imagens como dados de entrada, existe uma grande variedade de técnicas de manipulação que podem ser aplicadas como *data augmentation*. Embora seja possível utilizá-las em diversas ocasiões, para atingir melhores resultados é necessário escolher os filtros adequados com base no problema a ser atacado. Como exemplificado por Dvornik *et al*. (2018, p. 1), aplicar *augmentations* impróprios, como *flip* horizontal em um *dataset* com imagens de dígitos numéricos, pode subverter a eficiência da técnica, já que o resultado obtido não fará parte do espaço de soluções do problema.

Para as situações que interferem no posicionamento da imagem, as categorias mais comuns são:

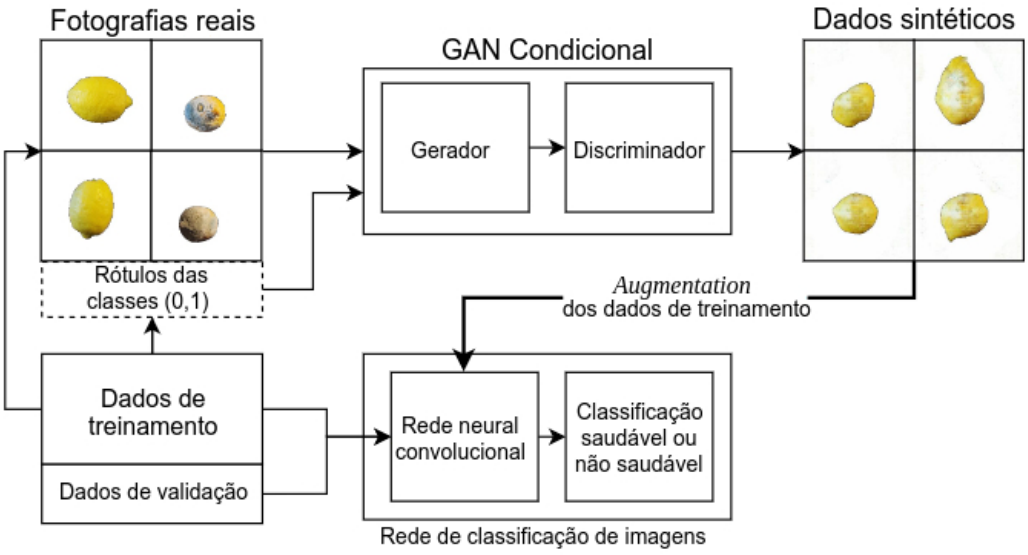
- Redimensionamento: Consiste em alterar a resolução (largura e altura) da amostra original
- Recorte: Seleciona somente uma porção da imagem original, cortando informações desnecessárias

- *Flip:* Vira a imagem no eixo vertical e/ou horizontal
- Rotação: Gira a imagem no sentido horário ou anti-horário, alterando a angulação

Além disso, em determinadas aplicações, técnicas como adição de ruídos e alteração nas características de cor da figura, como controle de brilho, contraste, saturação e matiz, podem ser apropriadas.

Outra possibilidade para a geração de amostras sintéticas é o uso de GANs Condicionais. Como mencionado na seção anterior, as Redes Adversárias Generativas formam uma arquitetura que possibilita a criação de novos dados a partir de um vetor de entrada aleatório. As GANs Condicionais são uma extensão dessa arquitetura, onde o Gerador e Discriminador também recebem informações adicionais na entrada (MIRZA e OSINDERO, 2014, p. 2). Para o caso de *augmentation*, essas informações poderiam ser imagens originais ou rótulos. Na Figura 6, de Bird *et al*. (2021, p. 5), é demonstrada uma visão geral do processo de *augmentation* com uma GAN Condicional, que auxilia no modelo treinado para classificar imagens de frutas podres e próprias para o consumo.

FIGURA 6 – VISÃO GERAL DE MODELO QUE UTILIZA GAN CONDICIONAL PARA DATA AUGMENTATION



FONTE: Adaptado de Bird *et al*. (2021, p. 5).

3 METODOLOGIA

O capítulo atual tem o propósito de detalhar os processos realizados durante a execução dessa pesquisa. Os tópicos a seguir irão explicar as escolhas e abordagens adotadas para alcançar o objetivo geral.

3.1 FERRAMENTAS

Nesta seção serão apresentadas as ferramentas que nos auxiliaram durante a construção do trabalho.

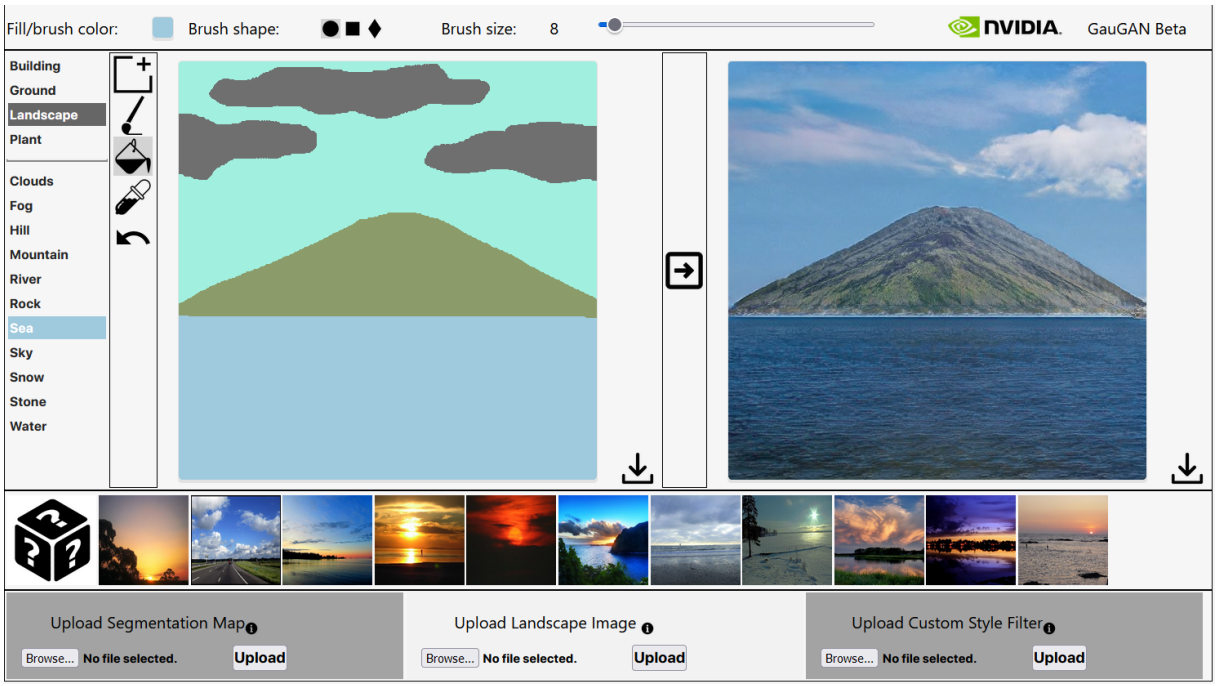
3.1.1 GauGAN

Com a apresentação do artigo de Park *et al*. (2019) no evento *Conference on Computer Vision and Pattern Recognition*, o GauGAN surgiu como uma ferramenta que utiliza redes GANs para transformar desenhos simples em imagens extremamente realistas.

Nomeado em homenagem ao pintor pós-impressionista Paul Gauguin, o software conta com uma interface para a edição dos desenhos, onde é possível colorir o *canvas* por meio de pincéis que representam construções e elementos da natureza. Assim como outros editores de imagens simples, é possível ajustar o tamanho e formato do pincel, utilizar a opção “balde de tinta” para preencher uma região e o “conta-gotas” para copiar a cor presente em uma determinada posição da imagem.

A tecnologia possibilita que o usuário escolha entre diversos estilos para a imagem de saída. Na Figura 7, estas possibilidades estão dispostas logo abaixo do desenho e da imagem resultante.

FIGURA 7 – INTERFACE DO GAUGAN

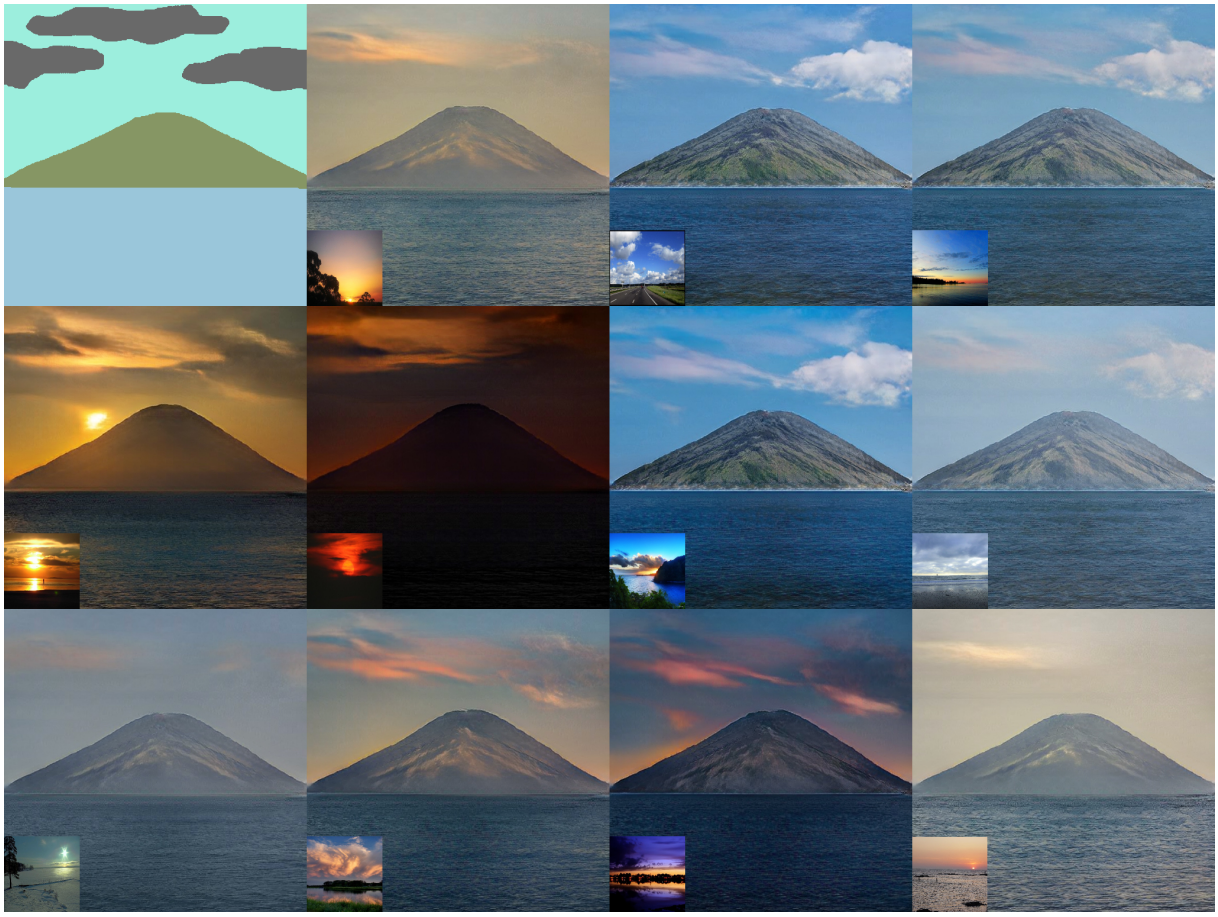


FONTE: Os autores (2021)[2].

Ao selecionar um novo estilo, a imagem de saída é atualizada, alterando características como luminosidade, posição do sol, estação do ano e clima, se aproximando mais do cenário desejado. A Figura 8 apresenta uma série de imagens que são criadas a partir de uma única entrada. A alteração das características de luminosidade, posição do sol e outras pode ser vista nas imagens criadas pelo GauGAN

2 Disponível em: <http://nvidia-research-mingyuliu.com/gaugan/>. Acesso em: 10 maio 2021

FIGURA 8 – SAÍDAS MODIFICADAS PELOS ESTILOS



FONTE: Os autores (2021).

Após finalizar a criação da imagem sintética, o usuário pode exportá-la em formato JPG na resolução de 512x512 *pixels*. A máscara de segmentação pode ser obtida na mesma resolução, com a extensão PNG.

3.1.2 Pixel Annotation Tool

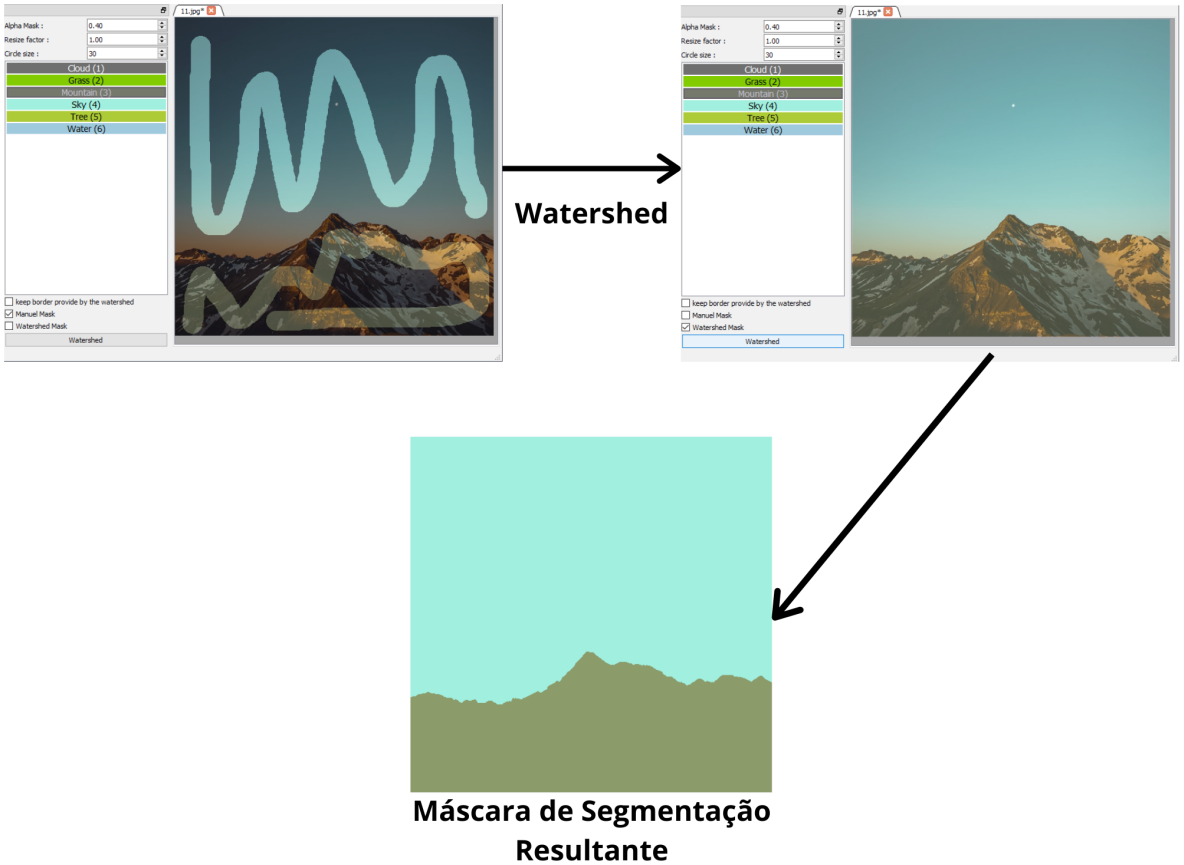
Em algoritmos de aprendizado de máquina supervisionados, uma das tarefas necessárias para a criação de um *dataset* é a rotulação. Neste processo, é feita a classificação manual dos dados (imagens, áudios, vídeos ou textos) que serão utilizados durante os treinamentos e validação.

Conforme Breheret (2017), o Pixel Annotation Tool é uma ferramenta que permite anotar imagens manualmente de forma ágil. Nela, o usuário utiliza pincéis

para segmentar a imagem com os rótulos desejados. As marcações criadas não precisam ser minuciosas, já que o algoritmo *watershed marked* do OpenCV é utilizado para essa tarefa, expandindo o rótulo aplicado em certo ponto da imagem para seu entorno, com base nos contornos identificados na imagem. Caso necessário, ainda é possível refinar qualquer segmentação feita erroneamente pelo algoritmo, até que a rotulação esteja satisfatória.

A Figura 9 demonstra o funcionamento do Pixel Annotation Tool. No exemplo, é realizada a marcação dos rótulos Montanha e Céu.

FIGURA 9 - FUNCIONAMENTO DO PIXEL ANNOTATION TOOL



FONTE: Os Autores (2021).

Por padrão, o software é carregado com rótulos apropriados para aplicações de trânsito, como veículos, placas de sinalização, estrada, calçada, etc. Contudo, é possível sobrescrever essa configuração através de um arquivo no formato JSON.

Selecionando a opção “*Open config file*” no menu “*Tool*”, o usuário pode importar o arquivo de configuração apropriado para sua aplicação, que deve estar no seguinte formato (FIGURA 10):

FIGURA 10 - EXEMPLO CONFIGURAÇÃO PARA O PIXEL ANNOTATION TOOL

```
{
    "labels": {
        "rotulo_1": {
            "categorie": "nome da categoria",
            "color": [
                0,
                100,
                0
            ],
            "id": 1,
            "id_categorie": 1,
            "name": "nome do rótulo 1"
        },
        "rotulo_2": {
            "categorie": "nome da categoria 2",
            "color": [
                100,
                0,
                0
            ],
            "id": 2,
            "id_categorie": 2,
            "name": "nome do rótulo 2"
        }
    }
}
```

FONTE: Os Autores (2021).

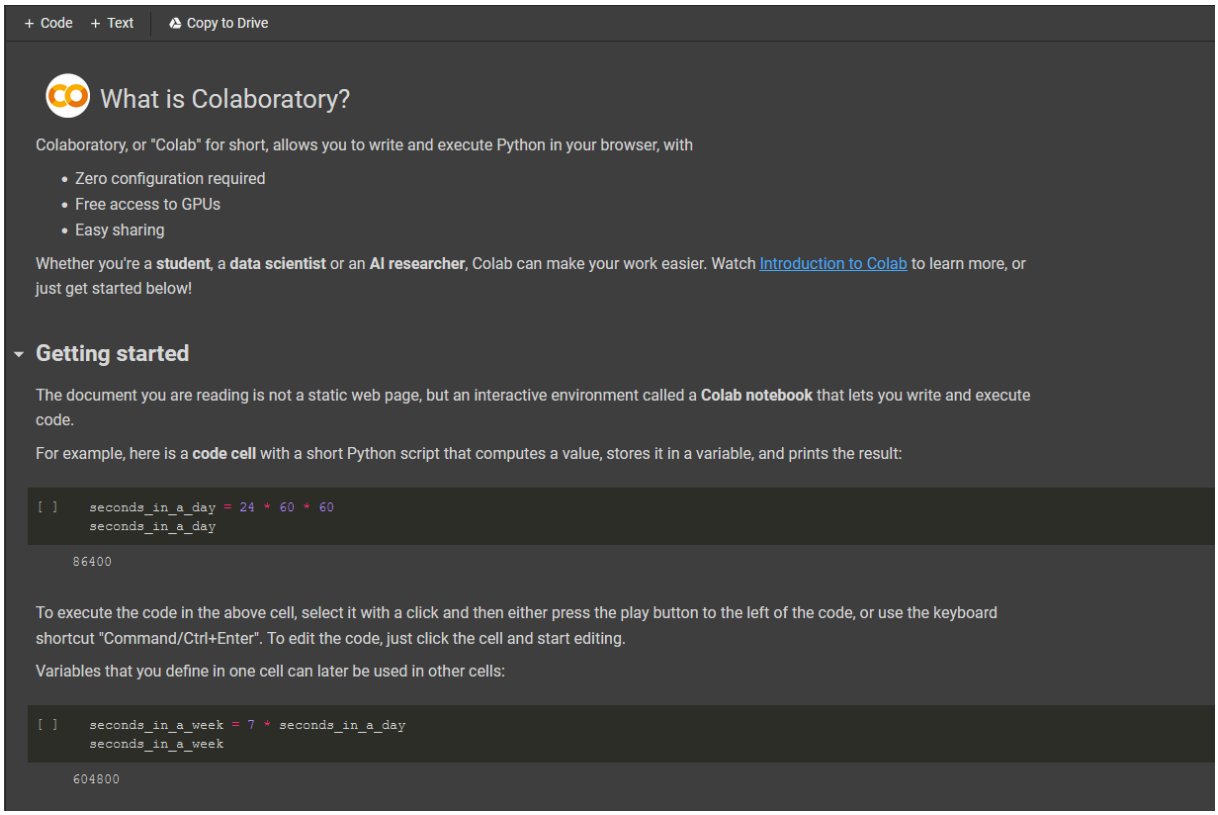
Após finalizar a anotação, a máscara de saída é salva no formato PNG com as mesmas dimensões da imagem original. Além disso, o Pixel Annotation Tool também salva mais dois arquivos monocromáticos representando a *watershed mask* e as pinturas feitas pelo usuário.

3.1.3 Colaboratory

O Google Colaboratory, também conhecido como Colab, é uma plataforma que permite o desenvolvimento e execução de código Python por meio de navegadores web.

A ferramenta funciona como um ambiente interativo, seguindo os moldes do Jupyter Notebook. Nela, o usuário é capaz de realizar anotações com as linguagens Markdown, HTML e LaTeX, adicionar imagens e executar blocos de código utilizando recursos computacionais de disco, GPU e CPU hospedados na nuvem da Google. A Figura 11 ilustra o ambiente do Colab, que também suporta as diversas bibliotecas da linguagem Python.

FIGURA 11 - NOTEBOOK DE INTRODUÇÃO AO GOOGLE COLAB



FONTE: Os Autores (2021)[3].

Facilitando e incentivando o estudo nas áreas de Inteligência Artificial e Ciência de Dados, o Colab armazena os notebooks salvos no Google Drive do autor,

3 Disponível em <https://colab.research.google.com/notebooks/intro.ipynb>. Acesso em: 28 jul. 2021

possibilitando o compartilhamento com outros indivíduos, que podem adicionar comentários e realizar edições no documento.

Por se aproveitar das soluções disponibilizadas na nuvem, o sistema pode ser usado em máquinas com menos potência, necessitando apenas de uma conexão com a internet e um navegador web. Ao executar as células de código em um notebook, o usuário recebe uma máquina virtual temporária, que tem vida útil de no máximo 12 horas no plano gratuito. Caso necessário, é possível assinar o Colab Pro, que garante GPUs mais rápidas, mais memória RAM, armazenamento em disco e tempo de uso.

3.1.4 Tensorflow

De acordo com Abadi *et al*. (2015), o Tensorflow é uma plataforma que possibilita expressar e executar algoritmos de aprendizado de máquina. Esta, foi desenvolvida para operar em larga escala e em ambientes heterogêneos, desde dispositivos móveis até sistemas distribuídos com milhares de computadores. (ABADI *et al*, 2016).

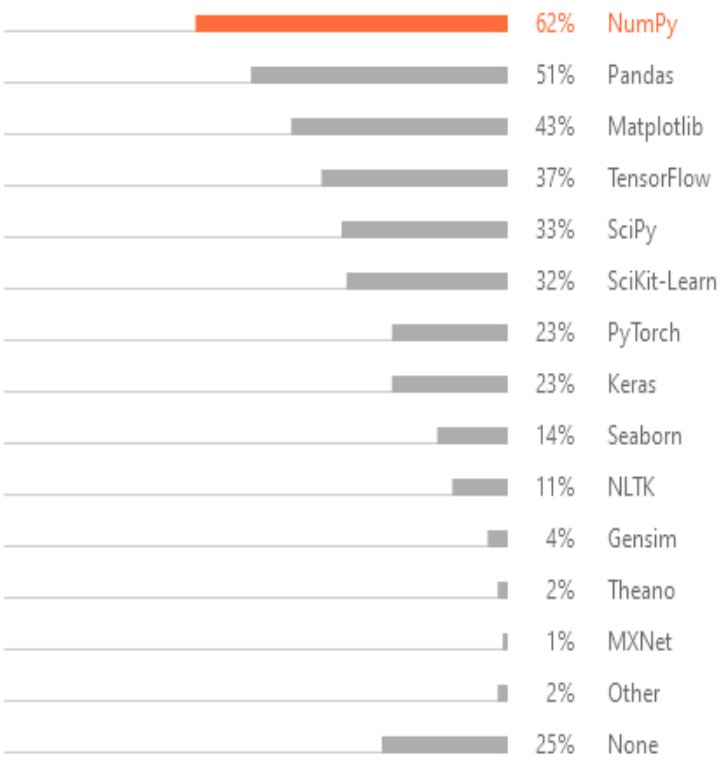
O Tensorflow foi lançado gratuitamente como pacote open-source sob licença Apache 2.0 em novembro de 2015. Segundo Abadi *et al*.[4] (2015, p. 1, tradução nossa):

> O sistema é flexível e pode ser usado para expressar uma extensa variedade de algoritmos, como treinamento e inferência para modelos de redes neurais profundas, e tem sido utilizado para conduzir pesquisas e implantar, em produção, sistemas de aprendizado de máquina em múltiplas áreas da ciência da computação e outros campos, incluindo reconhecimento de fala, visão computacional, robótica, recuperação de informação, processamento de linguagem natural, extração de informação geográfica e descoberta computacional de drogas.

4 The system is flexible and can be used to express a wide variety of algorithms, including training and inference algorithms for deep neural network models, and it has been used for conducting research and for deploying machine learning systems into production across more than a dozen areas of computer science and other fields, including speech recognition, computer vision, robotics, information retrieval, natural language processing, geographic information extraction, and computational drug discovery.

Nos últimos anos, a ferramenta vem ganhando popularidade, com destaque no contexto de aplicações em Python. Segundo pesquisa realizada pela JetBrains em 2020, o Tensorflow aparece como quarto framework de ciência de dados mais utilizado por desenvolvedores nessa linguagem. No Gráfico 1 podemos ver a figura que ilustra essa informação.

GRÁFICO 1 - “QUAIS *FRAMEWORKS DE CIÊNCIA DE DADOS VOCÊ UTILIZA JUNTO AO PYTHON*?”



FONTE: JETBRAINS (2020).

Com o lançamento da versão 2.0 em 2017, dentre as diversas novidades, a API Keras foi incorporada ao Tensorflow através do módulo *tf.keras*, promovendo funcionalidades mais acessíveis ao usuário no contexto de *deep learning*.

Em Python, a instalação pode ser feita através do gerenciador de pacotes pip, com o comando *pip install tensorflow.* Para o Google Colab, a biblioteca já vem pré-instalada, necessitando apenas que o usuário realize a importação caso for utilizá-la.

3.2 ESTUDO DAS FERRAMENTAS E REFERÊNCIAS

A etapa inicial da pesquisa consistiu em estudar as referências literárias e avaliar as ferramentas que seriam necessárias para desenvolver o código-fonte da aplicação. No processo, procuramos optar por soluções caracterizadas pelo renome, licença (código aberto), estabilidade e documentação íntegra.

3.3 PROCESSO DE SELEÇÃO E ELABORAÇÃO DOS DADOS

A seção atual explicará quais foram as etapas necessárias e critérios utilizados para selecionar e produzir os *datasets* de treino, validação e testes.

3.3.1 Imagens sintéticas de entrada

Apesar do GauGAN possuir uma grande variedade de pincéis, com o objetivo de facilitar o treinamento e a criação das imagens, decidimos utilizar apenas uma parcela dessas classes em nossa aplicação. Para isso, realizamos um processo de seleção, no qual foram geradas múltiplas imagens, abrangendo a diversidade disponibilizada pela ferramenta. Dessa forma, ao analisar visualmente o resultado da triagem, foi possível identificar os seguintes pontos:

- Determinados elementos são gerados de maneira menos realista, sendo possível observar padrões de forma e textura que não ocorrem na natureza
- Algumas classes exigem uma máscara de segmentação extremamente detalhada para serem reproduzidas corretamente
- A ferramenta tem mais facilidade para gerar cenários no ponto de vista frontal, ao tentar reproduzir cenários vistos de cima ou de ângulos alternativos, o resultado obtido foi insatisfatório
- A classe Árvore é melhor representada no contexto de florestas, quando desenhamos árvores individuais ou separadas a saída obtida não é convincente na maior parte dos casos

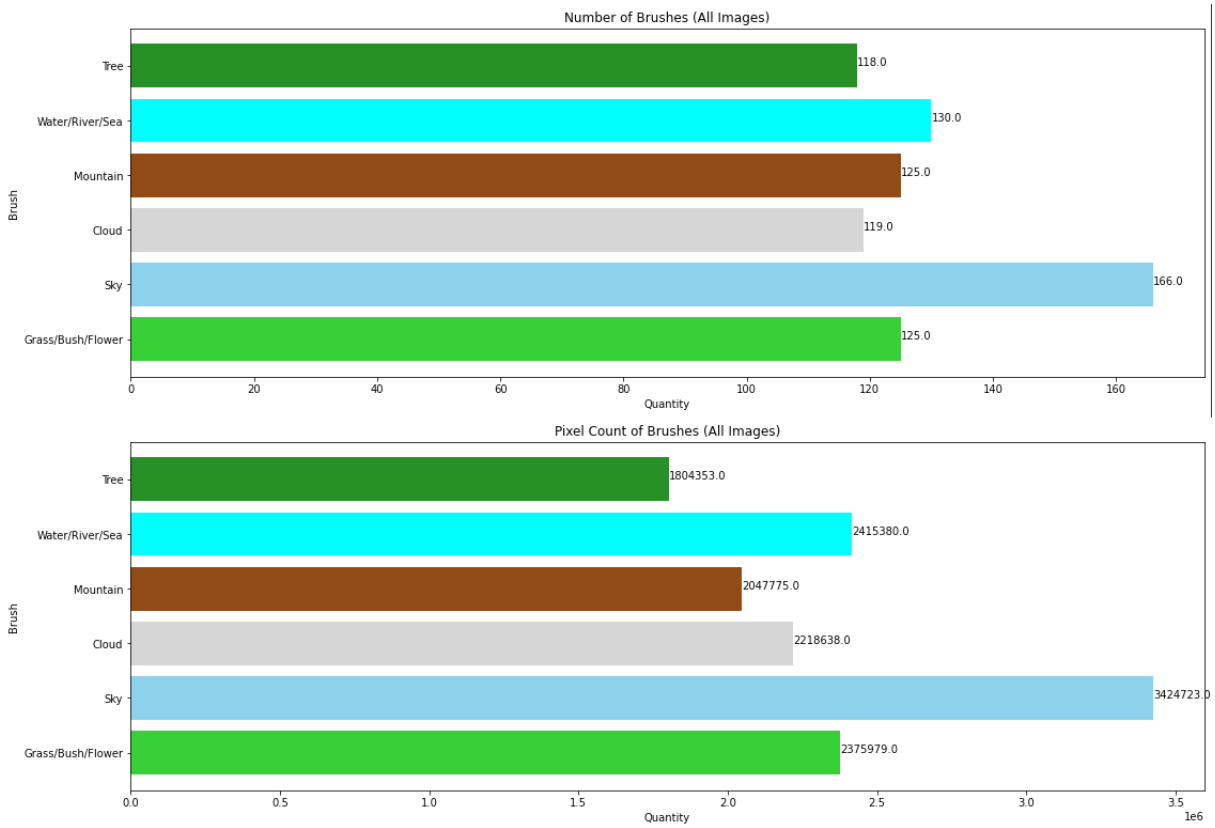
Com essas informações, foi possível definir de modo mais assertivo as classes a serem selecionadas. Levando em consideração que o trabalho se trata de paisagens naturais, decidimos descartar os elementos da categoria “Construção”, como “Telhado”, “Cerca”, “Parede de Madeira”, etc. Em seguida, priorizando elementos abundantes na natureza e que foram bem representados em nossos testes, chegamos na seguinte lista:

- Grama
- Céu
- Nuvem
- Montanha
- Água
- Mar
- Rio
- Árvore

Conforme mencionado anteriormente, pelo fato das árvores não serem bem reproduzidas de maneira individual, convencionamos que em nossa aplicação seriam desenhados apenas exemplos onde as árvores representam florestas. Além disso, para evitar classes muito específicas, consideramos o conjunto Água, Mar e Rio como o mesmo elemento.

Durante a produção das ilustrações, a fim de evitar o desequilíbrio na quantidade de classes, adicionamos no código um trecho responsável por quantificar as classes por imagem no *dataset* e gerar um gráfico com essas informações. O Gráfico 2 demonstra um resultado da execução, no qual o primeiro diagrama mostra a quantidade de pincéis (classes) considerando todas as imagens do *dataset* e o segundo exibe o valor total de *pixels* para cada classe no mesmo conjunto.

GRÁFICO 2 - QUANTIDADE DE CLASSES POR IMAGENS NO *DATASET*



FONTE: Os autores (2021).

Analisando de forma constante os gráficos gerados, foi possível manter a quantidade de classes relativamente estável, com exceção da classe céu, que por ser um elemento que tem maior presença nas paisagens em geral, naturalmente acabou aparecendo uma quantidade maior de vezes.

3.3.2 Imagens reais de validação e teste

Assim como no *dataset* de entrada, as imagens reais, utilizadas para validação e teste, precisaram passar por um critério de seleção. No caso, através do repositório gratuito Unsplash, foram escolhidas paisagens que possuíam no mínimo duas das classes utilizadas em nosso modelo. Além disso, procuramos favorecer imagens similares às geradas pelo GauGAN, evitando exemplos extremamente “artísticos”, com excesso de efeitos de iluminação e perspectivas alternativas (FIGURA 12).

FIGURA 12 – EXEMPLOS DE IMAGENS QUE FORAM DESCONSIDERADAS NA SELEÇÃO DO *DATASET* DE VALIDAÇÃO

FONTE: Compilação dos Autores (2021)[5].

Ao final do processo, foram selecionadas um total de quarenta imagens reais, divididas na metade para testes e validação. Para adicioná-las ao *dataset*, o primeiro passo foi redimensioná-las para 512x512 *pixels*. Em seguida, através do Pixel Annotation Tool, realizamos a anotação dos rótulos com o mesmo padrão de cores/classes do GauGAN, que foram definidos em um arquivo de configuração customizado para nossa aplicação.

3.4 MÉTRICAS DE AVALIAÇÃO

Para medir a performance do modelo, foram utilizadas as seguintes métricas de avaliação: *Pixel Accuracy, Intersection over Union* e matriz de confusão. De

5 Montagem criada a partir de imagens coletadas no repositório Unsplash.

maneira mais simples, a primeira calcula apenas a porcentagem dos *pixels* que foi classificada corretamente, ou seja:

$$PA = \frac{C}{T}$$

onde C representa a quantidade de *pixels* classificada corretamente e T o total de *pixels* nas imagens. Por considerar apenas a proporção de pixels corretos, essa métrica pode acabar gerando resultados ilusórios, já que em imagens onde a região segmentada é pequena, em caso de erro na classificação, a acurácia não vai sofrer impactos consideráveis.

A métrica *Intersection over Union*, também conhecida como Índice de Jaccard*,* quantifica a porcentagem de sobreposição da máscara de segmentação real com a predição resultante da rede. Para isso, o resultado da intersecção da máscara real com a máscara obtida é dividido pela união de ambas:

$$IoU = \frac{M_{real} \cap M_{obt}}{M_{real} \cup M_{obt}}$$

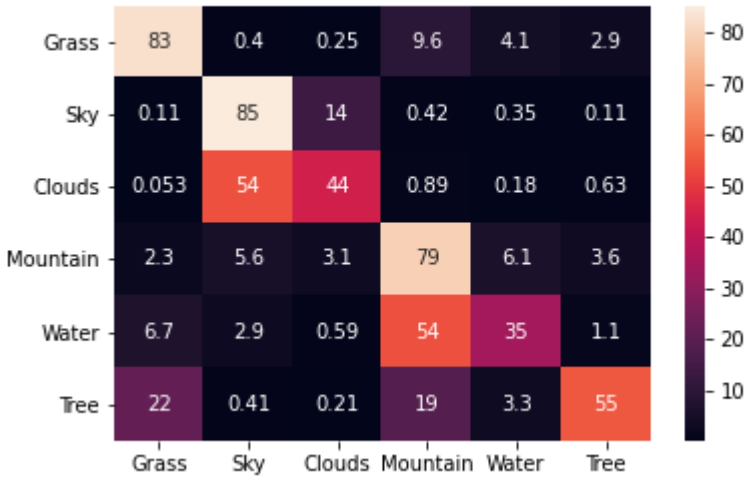
Em nossa implementação, utilizamos o componente tf.keras.metrics.MeanIoU, que calcula o IoU separadamente para cada classe e depois retorna a média dos valores obtidos.

Diferente da técnica anterior, o IoU lida melhor com os casos específicos de erro na segmentação de elementos pequenos, já que nessas situações a intersecção será mínima, consequentemente reduzindo o resultado obtido pela métrica.

Por fim, de forma a expandir as informações obtidas através das duas métricas anteriores, implementamos uma matriz de confusão. Para tal, utilizamos a função *confusion_matrix* do Tensorflow para o cálculo dos valores da matriz, juntamente com o DataFrame da biblioteca *pandas,* para estruturação dos valores obtidos, e o *heatmap* da biblioteca *Seaborn*, obtendo uma matriz gráfica com código de cores para melhor identificação dos valores apresentados.

A matriz gerada possibilita identificar não apenas quais rótulos o modelo tem maior dificuldade de identificar, como também quais os erros de classificação mais frequentes. Na Figura 13, por exemplo, fica claro que 54% dos pixels rotulados como “clouds” foram classificados incorretamente como “sky”. Com isso, o processo de balanceamento do *dataset* de treinamento é facilitado.

FIGURA 13 – EXEMPLO DE MATRIZ DE CONFUSÃO

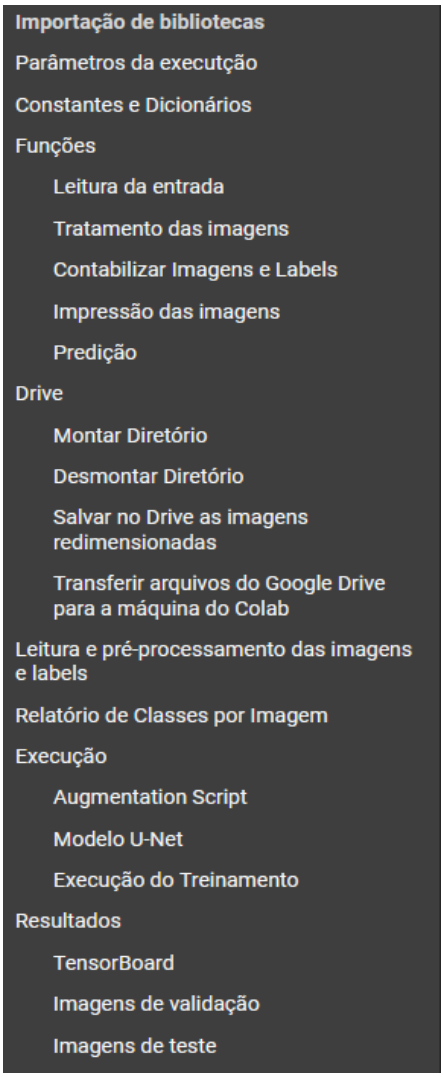


FONTE: Os Autores (2021).

3.5 ORGANIZAÇÃO DO NOTEBOOK

Através das anotações em Markdown, o notebook com o código-fonte da aplicação foi subdividido em tópicos, facilitando a leitura e interpretação do documento (FIGURA 14).

FIGURA 14 – ESTRUTURA DE TÓPICOS NO NOTEBOOK



FONTE: Os Autores (2021).

No início da execução, o usuário deve selecionar os seguintes parâmetros de configuração:

a) Executar o script de data augmentation? (s/n)
b) Caso “s” no *input* anterior: Insira o valor de multiplicador para o data augmentation
c) Redimensionar e salvar as imagens no Drive? (s/n)
d) Insira o tamanho das imagens que serão utilizadas (512, 256, etc)
e) Insira um nome personalizado para identificar este treinamento

Para utilizar imagens em resoluções alternativas, que não foram cadastradas anteriormente, é necessário habilitar a opção no item b), que criará um novo diretório no Google Drive com as imagens redimensionadas.

Com essas opções, o código é então executado seguindo o fluxo customizado de acordo com os valores inseridos.

3.6 CONSTRUÇÃO DO ALGORITMO

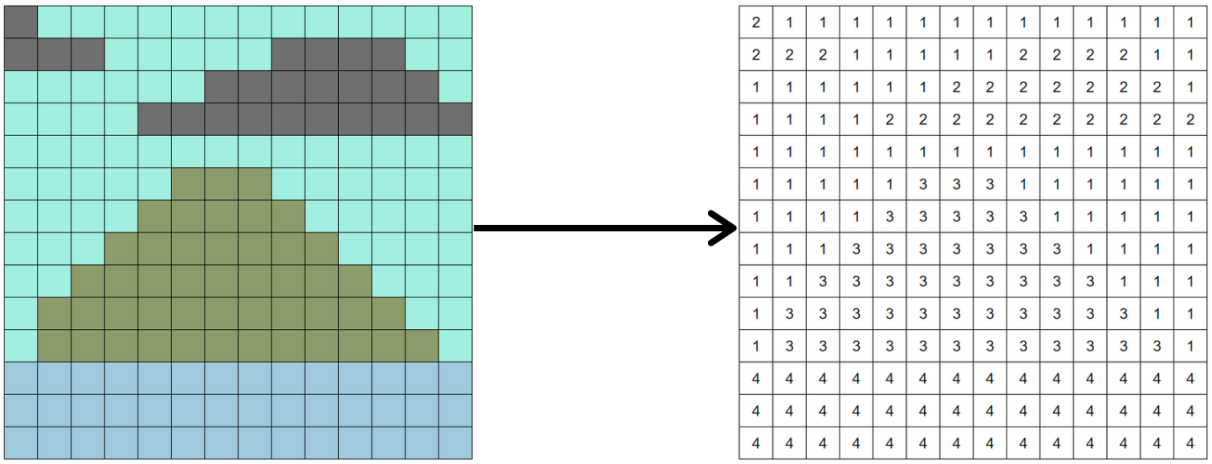
Neste tópico será discutida a implementação do código-fonte responsável pela leitura dos dados de entrada, treinamento e testes da rede U-Net, desenvolvida para segmentar elementos em paisagens reais. A aplicação foi arquitetada em um *notebook* na linguagem Python, e pode ser acessada a partir de um navegador web na plataforma Google Colaboratory.

3.6.1 Processamento dos dados de entrada

Por oferecer uma integração direta com o Colab, o Google Drive foi a plataforma escolhida para o armazenamento dos dados de entrada. Com o módulo *drive* da biblioteca google.colab, as imagens são exportadas para o diretório da máquina virtual concedida pela IDE. Em seguida, o processo de leitura é desempenhado com auxílio do OpenCV, que transforma os arquivos de entrada em matrizes manipuláveis no código.

Para o caso específico das máscaras de segmentação, é realizada uma etapa adicional de padronização para a introdução dos dados na rede. O procedimento consiste em converter os canais RGB em valores inteiros únicos, ou seja, as máscaras iniciais com pixels multivalorados representando cores específicas precisam ser transformadas em novas imagens de mesma largura e altura, mas com somente um único valor que indica a classe naquele pixel. Este processo é ilustrado na Figura 15, onde a imagem RGB a direita é codificada na matriz resultante do lado oposto.

FIGURA 15 – CONVERSÃO DA ENTRADA RGB PARA MATRIZ DE CANAL ÚNICO



FONTE: Os autores (2021).

O mapeamento de cores para índices únicos que representam as classes foi definido arbitrariamente pelos autores e está ilustrado no esquema abaixo:

QUADRO 1 - MAPEAMENTO DAS CORES NO FORMATO RGB PARA OS ÍNDICES INDIVIDUAIS

Classe	RGB	Índice
Grama	123, 200, 0	0
Céu	156, 238, 221	1
Nuvem	105, 105, 105	2
Montanha	134, 150, 100	3
Água	177, 200, 255	4
Rio	147, 100, 200	4
Mar	154, 198, 218	4
Árvore	168, 200, 50	5

FONTE: Os autores (2021).

3.6.2 Utilização do Data Augmentation

De forma a otimizar a produção de dados sintéticos para o treinamento do modelo implementado, optamos por aplicar técnicas de *data augmentation* para extrair o máximo de uso de cada imagem criada manualmente. Para tal fim, utilizamos a biblioteca Albumentations, que encapsula diversos algoritmos de

processamento de imagem frequentemente utilizados para a criação de variações, tais como cortes, distorções, adição de ruído e alterações no brilho, contraste e matiz da imagem.

Nosso *script* de *data augmentation* consiste em duas etapas, sendo a primeira de alterações estruturais e a segunda de alterações visuais. Na primeira etapa é aplicada uma das seguintes transformações (escolhida aleatoriamente): *HorizontalFlip* (espelhamento horizontal), *RandomSizedCrop* (corte aleatório de tamanho 65% a 75% do tamanho da imagem original) ou ambos, todos com p = 1 (100% de probabilidade de execução quando são selecionados).

Na etapa de transformações visuais, aplicamos o filtro FancyPCA, com alpha=0.1 e probabilidade 1. O processo de augmentation é aplicado para todas as imagens do *dataset* de treinamento, “N” vezes por imagem, sendo este “N” escolhido pelo usuário no início do programa.

3.6.3 Implementação da rede

A rede neural convolucional treinada no trabalho foi implementada com base na documentação do Tensorflow para segmentação de imagens. O modelo funciona seguindo as características de uma U-net, com algumas modificações. Na etapa de *downsampling*, são utilizadas as saídas intermediárias de um modelo MobileNetV2 pré-treinado, disponibilizado pela API do Keras no Tensorflow. O *upsampling* consiste em uma sequência de quatro camadas de deconvolução e *batch normalization* com função de ativação ReLU. No método de criação do modelo é realizada a configuração das *skip connections*, responsáveis por conectar os níveis correspondentes do *encoder* (*downsampler*) com o *decoder* (*upsampler*).

A rede é então compilada com o “adam”, um algoritmo de otimização estocástica de primeira ordem baseado em gradiente computacionalmente eficiente, e utilizando a acurácia e meanIoU como métricas para o treinamento e validação do modelo. Também utilizamos o *SparseCategoricalCrossentropy* como métrica de *loss*, conforme recomendado pela documentação do Tensorflow para segmentação de imagens, uma vez que um label é atribuído a cada pixel, e a saída da rede consiste

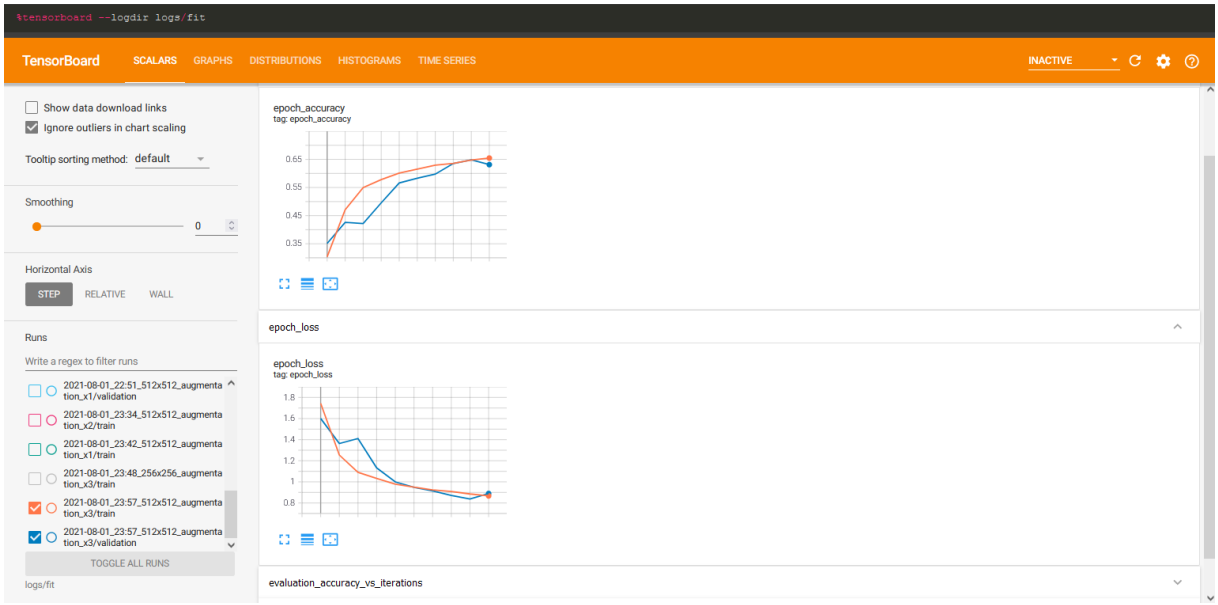
em um canal para cada classe, o que torna a função em questão ideal para tal cenário.

3.6.4 Logs de execução

A fim de salvar os resultados de cada treinamento da rede, integramos a ferramenta de visualização Tensorboard, do Tensorflow, ao nosso modelo. Através de uma função de *callback* chamada durante o processo de treinamento, o Tensorboard armazena uma série de informações sobre o modelo e gera um relatório com gráficos e histogramas de fácil visualização.

Os logs são gravados no diretório logs/fit do Google Drive, assim, é possível comparar duas ou mais execuções distintas através da interface disponibilizada.

FIGURA 16 – INTERFACE DE VISUALIZAÇÃO DO TENSORBOARD



FONTE: Os autores (2021).

4 APRESENTAÇÃO DOS RESULTADOS

Os resultados que serão apresentados nesta seção foram obtidos a partir dos treinamentos e testes realizados na plataforma Google Colaboratory. Para cada execução os recursos de processamento foram diferentes, já que a distribuição de máquinas virtuais é aleatória. Portanto, antes de cada inicialização, foi necessário executar o comando *cat /proc/cpuinfo* para listar as especificações de CPU. No que se refere ao armazenamento, por termos contratado o plano Colab Pro, a capacidade de memória RAM ficou sempre em torno de 25GB, e a quantidade de armazenamento em disco aproximadamente 107GB.

4.1 PARÂMETROS DE TREINAMENTO

Para fazer o *tuning* (aperfeiçoamento) dos parâmetros da rede e de *data augmentation* antes de gerar os resultados com a base de testes, foram executados diversos treinamentos e validações. Grande parte destes foi feita utilizando imagens de tamanho 256x256 pixels, de forma a acelerar a obtenção das métricas, já que durante este período estávamos utilizando a versão gratuita do Colab, com menos memória e poder de processamento. A partir destes testes, decidimos utilizar os seguintes parâmetros, que obtiveram os melhores resultados:

- *Data augmentation*: Habilitado, gerando 4 imagens *“*aumentadas*”* para cada imagem de treinamento, utilizando *flip, crop* ou ambos e o FancyPCA para alterações de cor. *Seed* do *augmentation* pseudo-aleatória (baseada no horário atual);
- Tamanho das imagens: 512x512 pixels;
- Quantidade de imagens: 1155 para treinamento (sintéticas), sendo 231 únicas e 924 *augmentations*, 20 para validação (reais) e 20 para teste (reais)*;*
- Parâmetros da U-Net: *Dropout* habilitado (valor padrão da implementação de exemplo do TensorFlow), 32 imagens por *batch* (quantidade de imagens antes de executar o ajuste dos pesos), 15 épocas;

Foram executados, ao todo, 10 treinamentos distintos, cada qual tendo seu desempenho avaliado no *dataset* de imagens de teste. Por termos usado uma semente de aleatoriedade para o *data augmentation* baseada no horário, as redes de cada execução foram treinadas com algumas imagens levemente diferentes. Nossa execução mais longa levou 3 horas, 22 minutos e 4 segundos para treinar a rede, e a mais rápida levou 2 horas e 28 minutos.

4.2 RESULTADOS OBTIDOS NAS MÉTRICAS DE AVALIAÇÃO

As tabelas 1, 2 e 3 apresentam os resultados das 10 execuções previamente citadas. A melhor e pior acurácia obtidas nos testes foram de 0,816 e 0,785, respectivamente, enquanto o melhor resultado para o IoU foi de 0,699 e o pior de 0,642. Para a acurácia de teste, obtivemos uma média de 0,800 com desvio padrão 0,008, e para o IoU, uma média de 0,673 com desvio padrão 0,015. No Gráfico 3, são apresentados também dois *boxplot* dos testes, em que a linha verde pontilhada é a média e a linha laranja sólida é a mediana.

TABELA 1 - LOSS FUNCTION

Execução	Loss (Treinamento)	Loss (Validação)
1	0,463	0,776
2	0,461	0,779
3	0,479	0,794
4	**0,450**	**0,863**
5	0,471	0,807
6	0,477	0,769
7	0,465	0,765
8	0,468	0,809
9	0,478	0,797
10	**0,495**	**0,734**
Média ± DP	**0,471 ± 0,012**	**0,789 ± 0,032**

FONTE: Os autores (2021).

TABELA 2 - PIXEL ACCURACY

Execução	Accuracy (Treinamento)	Accuracy (Validação)	Accuracy (Teste)
1	0,835	0,706	0,807
2	0,835	0,714	**0,785**
3	0,827	0,716	0,798
4	**0,837**	**0,694**	0,796
5	0,832	0,696	0,796
6	0,830	0,706	0,796
7	0,833	0,715	0,807
8	0,832	0,702	0,803
9	0,830	0,704	0,798
10	**0,820**	**0,719**	**0,816**
Média ± DP	**0,831 ± 0,005**	**0,707 ± 0,008**	**0,800 ± 0,008**

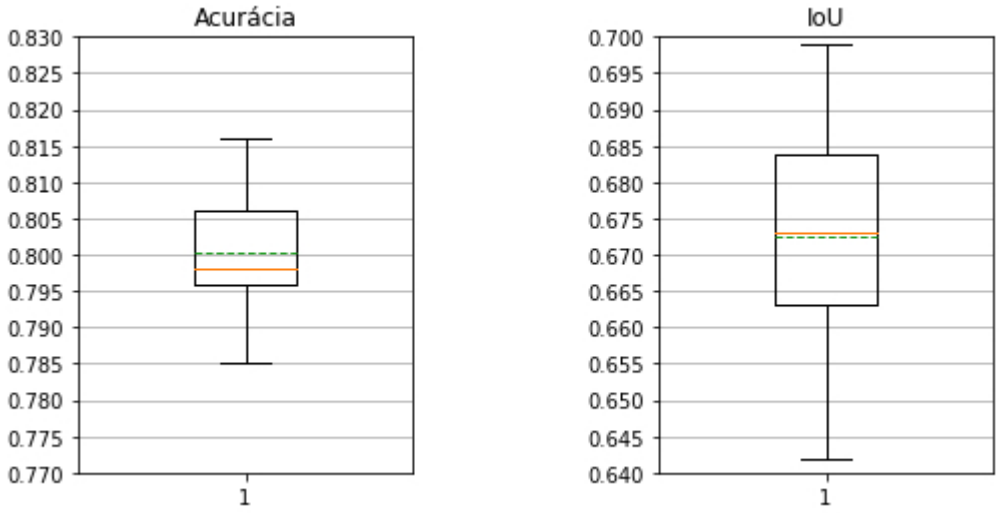
FONTE: Os autores (2021).

TABELA 3 - INTERSECTION OVER UNION (IOU)

Execução	IoU (Treinamento)	IoU (Validação)	Iou (Teste)
1	0,714	0,541	0,687
2	0,713	0,542	**0,642**
3	0,701	0,547	0,672
4	**0,718**	**0,517**	0,663
5	0,709	0,528	0,664
6	0,705	0,536	0,663
7	0,711	0,547	0,687
8	0,708	0,534	0,674
9	0,705	0,536	0,674
10	**0,692**	**0,557**	**0,699**
Média ± DP	**0,708 ± 0,007**	**0,538 ± 0,011**	**0,673 ± 0,015**

FONTE: Os autores (2021).

GRÁFICO 3 - BOXPLOT DA ACURÁCIA E IOU DOS TESTES

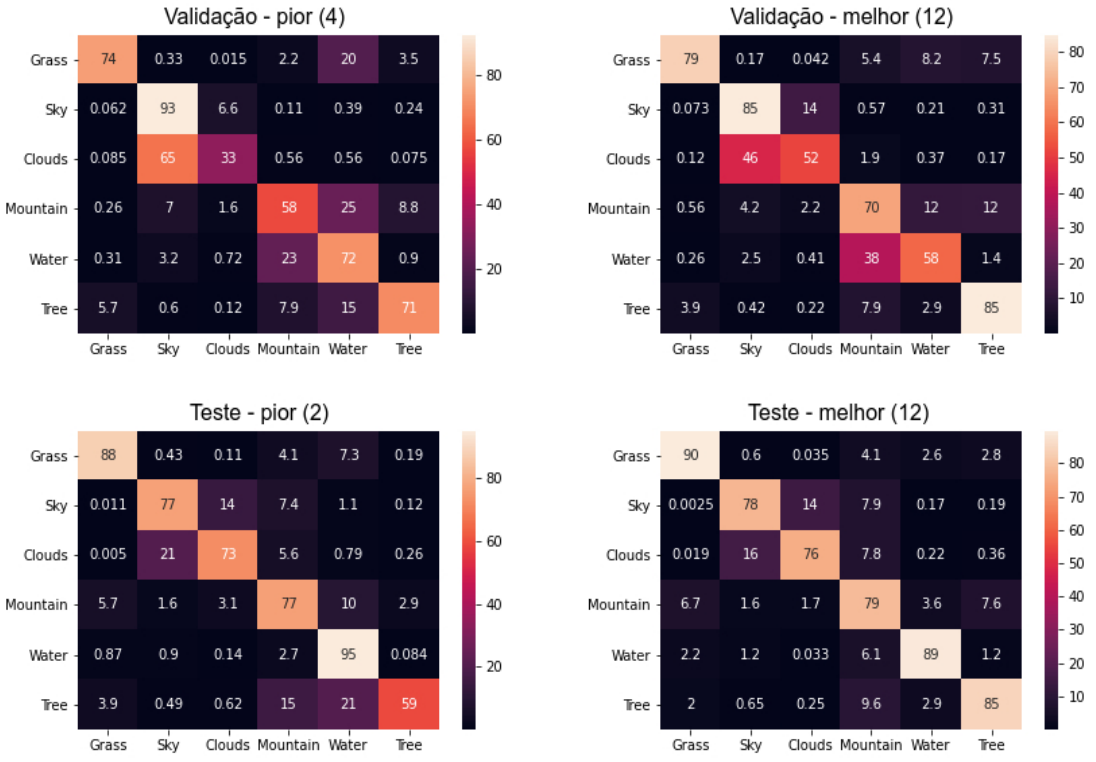


FONTE: Os autores (2021).

4.3 ANÁLISE DOS RESULTADOS

Tanto durante o treinamento quanto durante a validação e testes, uma das situações observadas foi a dificuldade para segmentar as classes Nuvem e Céu, o que pode ser observado nas matrizes de confusão (principalmente dos dados de validação) da Figura 17 (linha “Clouds” com coluna “Sky” e linha “Sky” com coluna “Clouds”). Além das nuvens serem um conjunto complexo devido às suas características físicas, outra situação que prejudicou o algoritmo foi a geração deste elemento no GauGAN. Ao desenhar uma máscara de segmentação, o resultado obtido nem sempre tem a mesma forma e ocupa o mesmo espaço determinado na entrada.

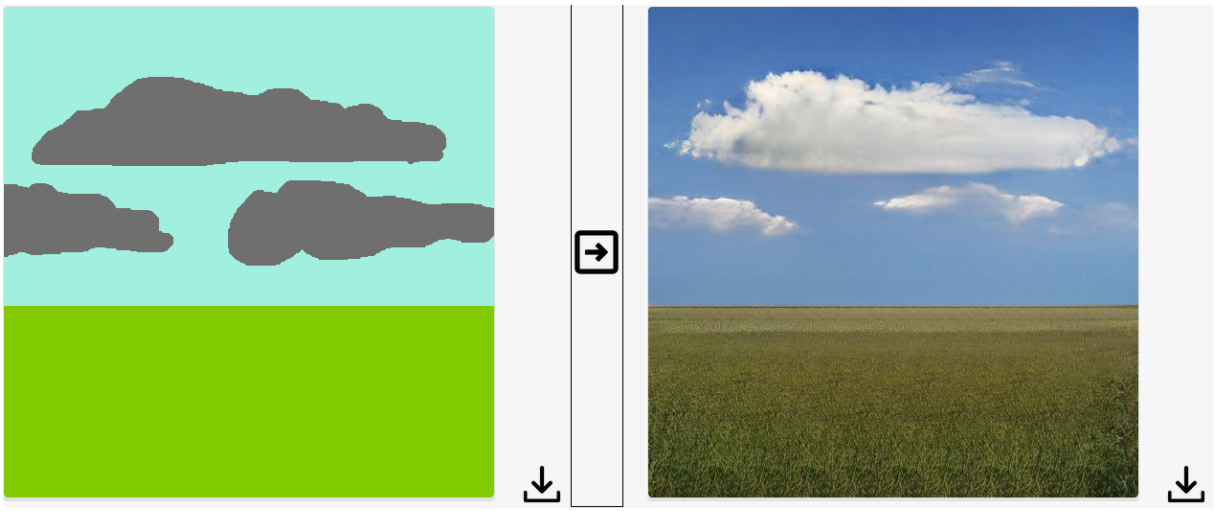
FIGURA 17 – MATRIZES DE CONFUSÃO DOS MELHORES E PIORES RESULTADOS



FONTE: Os autores (2021).

Como demonstrado na Figura 18, as nuvens criadas no resultado final (imagem à direita) não correspondem exatamente ao espaço determinado na anotação de entrada (imagem à esquerda). Essa situação acaba atrapalhando a capacidade do modelo entender as características de cada classe, consequentemente afetando seu desempenho e as métricas de avaliação.

FIGURA 18 – EXEMPLO DE SAÍDA DO GAUGAN COM NUVENS



FONTE: Os autores (2021).

Outra situação semelhante é que, ao desenhar determinadas máscaras com montanhas, é possível perceber que a figura resultante acaba sendo produzida com áreas de vegetação nos locais designados para essa classe. Na Figura 19, temos um exemplo no qual as montanhas geradas estão cobertas por tais áreas verdes, que podem ser interpretadas visualmente como árvores ou grama. Por mais que o resultado final não deixe de ser uma montanha, essa condição acaba causando ambiguidade com os demais rótulos, como evidenciado pelos números elevados nas células de encontro dos rótulos “Mountain” e “Tree” ou “Grass”, na Figura 17.

FIGURA 19 – EXEMPLO DE MONTANHAS COM VEGETAÇÃO

FONTE: Os autores (2021).

4.3.1 Discussões sobre as segmentações resultantes

Nesta seção serão avaliadas algumas das saídas obtidas com o modelo, procurando apresentar as classificações corretas e incorretas mais frequentes. Os resultados apresentados são provenientes da décima execução, na qual obtivemos a maior acurácia nos testes.

Na Figura 20 temos um exemplo de cenário que apresenta todas as classes utilizadas no modelo, com exceção das montanhas. Comparando a anotação original, produzida manualmente pelos autores com a ferramenta *Pixel Annotation Tool*, com a saída gerada, é possível observar uma semelhança grande entre ambas. Apesar de possuir pequenos ruídos, o resultado obtido conseguiu reproduzir bem a segmentação original, identificando os diversos elementos da paisagem, incluindo as árvores ao fundo da cena.

Há, contudo, certa divergência entre as segmentações das áreas verdes próximas à água, as quais foram rotuladas como grama (verde) pelo modelo e como árvores (verde amarelado) pelos autores. Fora das limitações do número de rótulos, contudo, estes elementos deveriam ser identificados como arbustos, um meio-termo entre “grama” e “árvore”, o que nos leva a considerar a classificação da U-Net como plausível. Nossos resultados, portanto, também podem ter sido afetados por esta

subjetividade de interpretação, pois certos componentes das paisagens poderiam ter sido classificados de forma diferente se outro indivíduo tivesse criado as máscaras de segmentação reais.

FIGURA 20 – VISUALIZAÇÃO DA SAÍDA GERADA PARA PAISAGEM DE RIO

FONTE: Os autores (2021).

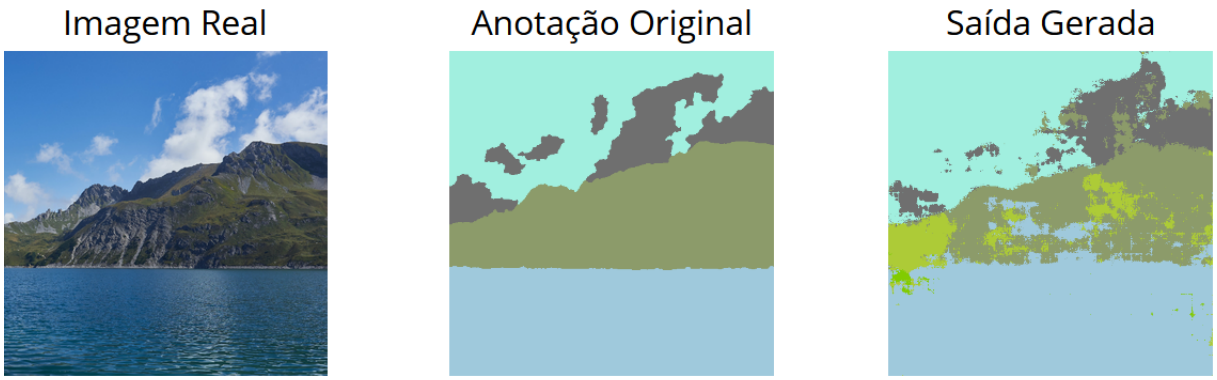
Já na Figura 21, podemos observar outra situação de divergência de rótulos. Nesse caso a grama foi precisamente classificada, com exceção de alguns *pixels* do horizonte. Contudo, conforme levantado anteriormente, vemos que há diferenças nas segmentações de Céu e Nuvem, que apesar de não prejudicarem o entendimento visual, afetam as métricas de avaliação.

FIGURA 21 – VISUALIZAÇÃO DA SAÍDA GERADA PARA PAISAGEM DE CAMPO

FONTE: Os autores (2021).

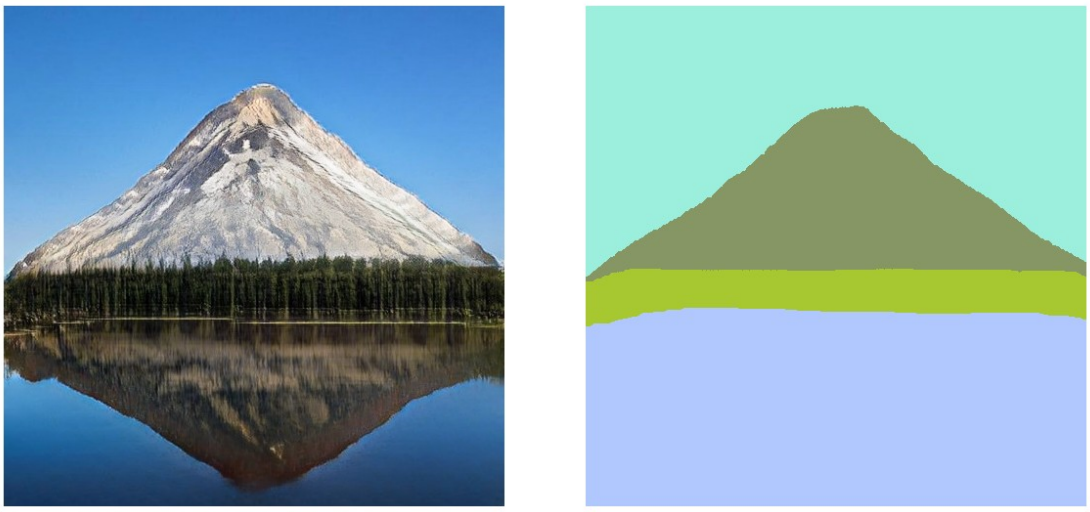
Por último, temos um exemplo de confusão na classificação de uma montanha, na Figura 22. Novamente temos certa ambiguidade na rotulação, em que a montanha em questão está coberta por áreas verdes, mas foi classificada como um único elemento no mapa de segmentação criado manualmente. A saída gerada, por sua vez, traz seções da paisagem rotuladas como Árvore, o que é compreensível, mas também traz porções de água incorretamente posicionadas. Acreditamos que tal falha de identificação pode estar relacionada com reflexos de elementos na água nas imagens de treinamento, como exemplificado na Figura 23.

FIGURA 22 – VISUALIZAÇÃO DA SAÍDA GERADA PARA PAISAGEM DE MAR E MONTANHA



FONTE: Os autores (2021).

FIGURA 23 – EXEMPLO DE IMAGEM DE TREINAMENTO COM REFLEXO NA ÁGUA



FONTE: Os autores (2021).

5 CONSIDERAÇÕES FINAIS

Apesar das confusões entre classes cometidas pelo modelo treinado, a U-Net foi capaz de obter um resultado relativamente satisfatório, considerando o tamanho do *dataset* de treino e as particularidades das imagens sintéticas geradas pelo GauGAN (como distorções e padrões de textura que quebram o realismo).

É importante destacar os resultados obtidos para as próprias imagens de treinamento, como a acurácia média de 0,831 e o IoU médio de 0,708, ambos os quais são apenas cerca de 0,03 mais elevados que os resultados obtidos nos testes, de acordo com as tabelas 2 e 3. Isso nos leva a acreditar que o modelo tem dificuldade para aprender a classificar as próprias imagens de entrada, muito provavelmente devido às semelhanças entre algumas classes, como previamente citado.

Todavia, percebemos que há diversos pontos de melhoria no desenvolvimento desta pesquisa. Primeiramente, acreditamos que um *dataset* de treinamento maior poderia ter aperfeiçoado os resultados. Existe a possibilidade de que as confusões entre árvores e montanhas, por exemplo, tenha sido apenas fruto da insuficiência de dados para que o modelo desenvolvesse uma identificação de padrões eficaz.

Um estudo mais aprofundado sobre técnicas de *data augmentation* também poderia ter afetado positivamente os resultados, uma vez que optamos por implementar manualmente transformações simples ao invés de técnicas do estado da arte. Tal observação pode ser exemplificada pelo próprio uso do FancyPCA, que substituiu nossa versão manual de *augmentation* baseada em alterações de matiz, contraste, brilho, saturação e remoção de canais de cor por ter obtido resultados melhores durante nosso *tuning*.

Por fim, vale observar que as imagens de validação e teste podem ter sido afetadas por viés dos autores, uma vez que antes da seleção destas já havíamos visto e criado diversas imagens utilizando o GauGAN e executado testes com a U-Net para avaliar a viabilidade da pesquisa. Dessa forma, é possível que o resultado tenha sido afetado positivamente justamente porque as imagens de teste escolhidas são as mais adequadas para a situação.

5.1 RECOMENDAÇÕES PARA TRABALHOS FUTUROS

Como trabalho futuro, no contexto da nossa pesquisa, entendemos que existe espaço para treinar o modelo proposto com uma variedade maior de classes e um *dataset* de treinamento com mais amostras sintéticas. Também é plausível realizar testes com parâmetros diferentes para a rede e outros filtros de *augmentation* para ampliar o conjunto de entrada, buscando resultados melhores que os obtidos nesta pesquisa.

Além disso, é possível desenvolver GANs Condicionais como geradoras de entradas para problemas distintos, como por exemplo na área da saúde, envolvendo imagens biomédicas.

REFERÊNCIAS

ABADI, Martín *et al*. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. [*s. l.*], p. 1-19, 9 nov. 2015. DOI 10.5281/zenodo.5095721. Disponível em: https://www.tensorflow.org/about/bib. Acesso em: 20 maio 2021.

ABADI, Martín *et al*. TensorFlow: A System For Large-Scale Machine Learning. [*s. l.*], p. 1-18, 27 maio 2016. Disponível em: https://arxiv.org/abs/1605.08695. Acesso em: 20 maio 2021.

ANDREWS, Gerard. What Is Synthetic Data?: Synthetic data generated from computer simulations or algorithms provides an inexpensive alternative to real-world data that's increasingly used to create accurate AI models. **The NVIDIA Blog**, [*S. l.*], p. 1-1, 8 jun. 2021. Disponível em: https://blogs.nvidia.com/blog/2021/06/08/what-is-synthetic-data/. Acesso em: 22 jul. 2021.

BREHERET, Amaury. **Pixel Annotation Tool**. 1.4.0. [*S. l.*], 16 nov. 2017. Disponível em: https://github.com/abreheret/PixelAnnotationTool. Acesso em: 2 jul. 2021.

DVORNIK, Nikita *et al*. On the Importance of Visual Context for Data Augmentation in Scene Understanding. , [*S. l.*], p. 1-7, 6 set. 2018. Disponível em: https://arxiv.org/abs/1809.02492. Acesso em: 5 ago. 2021.

GOODFELLOW, Ian J. *et al*. **Generative Adversarial Networks.** [*s. l.*], p. 1-9, 10 jun. 2014. Disponível em: https://arxiv.org/abs/1406.2661. Acesso em: 27 maio 2021.

GOODFELLOW, Ian J. *et al*. **Deep Learning**. [S. l.]: MIT Press, 2016. ISBN 978-0262035613. Disponível em: http://www.deeplearningbook.org. Acesso em: 8 ago. 2021.

HURWITZ, Judith; KIRSCH, Daniel. Machine Learning For Dummies ®: IBM Limited Edition. [S. l.]: **John Wiley & Sons, Inc.**, 2018. ISBN 978-1-119-45494-6. E-book (75p.).

JETBRAINS. **Python Developers Survey 2020 Results**. [*S. l.*], out. 2020. Disponível em: https://www.jetbrains.com/lp/python-developers-survey-2020/. Acesso em: 19 jul. 2021.

LECUN, Yann *et al*. Gradient-based Learning Applied to Document Recognition. **Proceedings of the IEEE**, [s. l.], v. 86, ed. 11, p. 2278 - 2324, Nov. 1998. DOI 10.1109/5.726791. Disponível em: http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf. Acesso em: 9 ago. 2021.

MAHESH, Batta. Machine Learning Algorithms - A Review. **International Journal of Science and Research (IJSR)**, [s. l.], v. 9, ed. 1, p. 1-6, 1 jan. 2020. DOI 10.21275/ART20203995. Disponível em: https://www.ijsr.net/archive/v9i1/ART20203995.pdf. Acesso em: 7 ago. 2021.

MIRZA, Mehdi *et al*. Conditional Generative Adversarial Nets. [*S. l.*], p. 1-7, 6 nov. 2014. Disponível em: https://arxiv.org/abs/1411.1784. Acesso em: 5 ago. 2021.

NIKOLENKO, Sergey I. Synthetic Data for Deep Learning. **Synthesis.ai**, [*S. l.*], p. 1-156, 26 set. 2019. Disponível em: https://arxiv.org/abs/1909.11512. Acesso em: 16 jul. 2021.

PARK, Taesung *et al*. Semantic Image Synthesis with Spatially-Adaptive Normalization. **Conference on Computer Vision and Pattern Recognition (CVPR)**, Long Beach, p. 1-19, 18 mar. 2019. Disponível em: https://arxiv.org/abs/1903.07291. Acesso em: 27 maio 2021.

RONNEBERGER, Olaf; FISCHER, Philipp; BROX, Thomas. U-Net: Convolutional Networks for Biomedical Image Segmentation. **Medical Image Computing and Computer-Assisted Intervention**, [s. l.], 2015. DOI 10.1007/978-3-319-24574-4_28. Disponível em: https://arxiv.org/pdf/1505.04597.pdf. Acesso em: 9 ago. 2021.

ROSEBROCK, Adrian. Keras ImageDataGenerator and Data Augmentation. **Pyimagesearch**, [*S. l.*], p. 1, 8 jul. 2019. Disponível em: https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/. Acesso em: 21 jul. 2021.

SHALEV-SHWARTZ, Shai; BEN-DAVID, Shai. Understanding Machine Learning: From Theory to Algorithms. [S. l.]: **Cambridge University Press**, 2014. ISBN 978-1-107-05713-5. E-book (449p.).