UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

PAULO TAVERNARO FRALETTI

Classificação de Imagens por Segmentação: uma análise de fotos aéreas da Floresta Amazônica através de Aprendizado de Máquina com Python, e com implementação em um Sistema Embarcado

São Carlos
2023

PAULO TAVERNARO FRALETTI

Classificação de Imagens por Segmentação: uma análise de fotos aéreas da Floresta Amazônica através de Aprendizado de Máquina com Python, e com implementação em um Sistema Embarcado

Monografia apresentada ao Curso de Engenharia Mecatrônica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Mecatrônico.

Orientador: Prof. Dr. Glauco Augusto de Paula Caurin

São Carlos

2023

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Fraletti, Paulo Tavernaro
F797c Classificação de Imagens por Segmentação: uma análise de fotos aéreas da Floresta Amazônica através de Aprendizado de Máquina com Python, e com implementação em um Sistema Embarcado / Paulo Tavernaro Fraletti; orientador Glauco Augusto de Paula Caurin. São Carlos, 2023.

Monografia (Graduação em Engenharia Mecatrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2023.

1. Floresta Amazônica. 2. Classificação de imagens. 3. Segmentação. 4. Machine Learning. 5. Aprendizado de Máquina. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE AVALIAÇÃO

Candidato: Paulo Tavernaro Fraletti

Título:

Classificação de Imagens por Segmentação: uma análise de fotos aéreas da Floresta Amazônica através de Aprendizado de Máquina com Python, e com implementação em um Sistema Embarcado.

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos da Universidade de São Paulo Curso de Engenharia Mecatrônica.

BANCA EXAMINADORA

Prof. Dr. Glauco Augusto de Paula Caurin (Orientador)

Nota atribuída: 9,0 (Nove)

(assinatura)

Dr.Eng. Antonio Carlos Daud Filho

Nota atribuída: 9,0 (Nove)

(assinatura)

Msc. Eng. Henrique Borges Garcia

Nota atribuída: 9,0 (Nove)

(assinatura)

Média: 9,0 (NOVE)

Resultado: APROVADO

Data: 12/12/2023.

Este trabalho tem condições de ser hospedado no Portal Digital da Biblioteca da EESC

SIM [X] NÃO [ ] Visto do orientador

RESUMO

FRALETTI, P. T. **Classificação de Imagens por Segmentação**: uma análise de fotos aéreas da Floresta Amazônica através de Aprendizado de Máquina com Python, e com implementação em um Sistema Embarcado. 2023. X f. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2023.

A preocupação quanto à preservação do meio ambiente é evidente em tempos atuais. Nesse sentido, diminuir os casos de desmatamento é uma das ações consideradas no combate às mudanças climáticas, principalmente em regiões de florestas nativas e com enorme influência no clima como a Floresta Amazônica. Com isso, por meio de classificação de imagens por segmentação através de aprendizado de máquina, este trabalho tem como objetivo analisar e mapear fotos da região amazônica, de modo a categorizar seu conteúdo. As categorias analisadas são: Vegetação de Floresta (VF), Vegetação Rasteira e Arbustos (VRA) e Sem Vegetação (SV). Para isso, é necessário extrair informações de intensidade de brilho e contorno de fronteiras das imagens por meio de filtros. Foram processadas 120 fotos, sendo mapeadas pixel a pixel, de maneira que estes representam as amostras a serem treinadas e testadas pelo modelo de *machine learning* através do algoritmo de Descida de Gradiente Estocática e do classificador *Support Vector Machine*. Após a obtenção do modelo treinado, este é implementado em um sistema embarcado para que, em um projeto futuro, seja acoplado a um veículo aéreo não tripulado (*drone*), realizando o processamento das imagens em tempo real. Por meio disso, é possível atingir o objetivo final de monitorar ao longo do tempo a região amazônica e contribuir na identificação de áreas desmatadas para que autoridades competentes tomem a devida providência. A exatidão do modelo de aprendizado de máquina foi de 66,62%, além da precisão média de 64%, *recall* médio de 60% e *F1-Score* médio de 61%. Por fim, conclui-se que a performance do modelo pode ser melhorada através de outras estratégias como utilizar imagens com três canais de cores, como o RGB, utilizar como amostras as fotos e não cada um de seus pixeis, além de utilizar uma câmera hiperespectral para capturar as imagens, permitindo maior resolução em diferentes bandas do espectro eletromagnético.

Palavras-chave: Floresta Amazônica. Classificação de imagens. Segmentação. *Machine Learning* (Aprendizado de Máquina).

ABSTRACT

FRALETTI, P. T. **Image Classification through Segmentation**: an analysis of aerial pictures of the Amazon Forest through Machine Learning with Python, implementing it in an Embedded System. 2023. X p. Monograph (Course final project) – São Carlos School of Engineering, University of São Paulo, São Carlos, 2023.

Concerns regarding environmental preservation are evident nowadays. This way, reducing deforestation is one of the actions taken to face climate changes, mostly in native forest regions with enormous influence on climate, such as the Amazon Forest. Hence, employing image classification using segmentation through machine learning, this project aims to analyze and map pictures of the Amazon region to categorize their content. The analyzed categories are Forest Vegetation, Understorey Vegetation and Bushes, and No Vegetation. For that, it is necessary to extract bright intensity and border contour information from the images by applying filters. A hundred-twenty pictures were processed, being mapped pixel by pixel, which represent the samples to be trained and tested by the machine learning model through the Stochastic Gradient Descent algorithm and the Support Vector Machine classifier. After obtaining the trained model, it is implemented in an embedded system in order to, in a future project, be assembled on an Unmanned Aerial Vehicle (UAV), processing the images in real-time. Thus, it is possible to achieve the final objective of monitoring throughout the time the Amazon region and contributing to the identification of deforested areas so the competent authorities can take the appropriate measures. The accuracy of the machine learning model was 66.62%, while the average precision was 64%, the average recall was 60%, and the average F1-Score was 61%. Finally, it is possible to conclude that the performance of the model can be improved through other strategies such as using images with three color channels, as RGB, utilizing pictures as samples instead of each one of their pixels, beyond using a hyperspectral camera to capture the images, bringing higher resolution in different bands of the electromagnetic spectrum.

Keywords: Amazon Forest. Image classification. Segmentation. Machine Learning.

LISTA DE ILUSTRAÇÕES

Figura 1 – Mapa dos principais biomas no Brasil, Pantanal e Campos Sulinos. …. 20

Figura 2 – Arco do desmatamento. …….. 24

Figura 3 – Fluxograma do processo de treinamento e teste do modelo. …….. 28

Figura 4 – Imagem dividida em quatro, com suas respectivas partes rotacionadas em 180°. …….. 30

Figura 5 – Exemplo de imagem de referência segmentada na escala RGB. …….. 31

Figura 6 – Exemplo de imagem de referência segmentada na escala de cinza. …. 32

Figura 7 – Exemplo de imagem original e sua respectiva versão com um dos filtros Gabor. …… 35

Figura 8 - Exemplo de imagem original e sua respectiva versão com o filtro Canny. …… 36

Figura 9 - Exemplo de imagem original e sua respectiva versão com o filtro Roberts. …….. 37

Figura 10 - Exemplo de imagem original e sua respectiva versão com o filtro Sobel. …… 38

Figura 11 - Exemplo de imagem original e sua respectiva versão com o filtro Prewitt. …… 39

Figura 12 - Exemplo de imagem original e sua respectiva versão com um dos filtros Gaussiano. …… 40

Figura 13 - Exemplo de imagem original e sua respectiva versão com o filtro Mediana. …… 41

Figura 14 - Exemplo de imagem original e sua respectiva versão com o filtro Chan-Vese. …….. 42

Figura 15 - Classificação de dados com dois atributos mediante o hiperplano e suas margens. …….. 46

Figura 16 - Função de perda hinge. …… 47

Figura 17 – Exemplo de hiperplano para classificação de imagens. …….. 49

Figura 18 – Matriz de confusão considerando cada uma das três classes. …….. 52

Figura 19 - Matriz de confusão do modelo. ……. 58

Figura 20 - Exatidão x Acumulado de dados de teste por imagens treinadas. ……. 59

Figura 21 – Rio e floresta são confundidos na versão classificada pelo modelo. … 60

Figura 22 – Imagem com apenas floresta sendo classificada pelo modelo com outras classes. ……. 61

Figura 23 - Precisão x Acumulado de dados de teste por imagens treinadas. ……. 62

Figura 24 - Exemplo de imagens diferentes de floresta, com claridades diferentes. 63

Figura 25 - Recall x Acumulado de dados de teste por imagens treinadas. ……. 64

Figura 26 - F1-Score x Acumulado de dados de teste por imagens treinadas. …… 65

Figura 27 - Módulo Apalis iMX8QM 4GB WB IT. ……. 68

Figura 28 - Placa base Ixora. ……. 69

Figura 29 - Estrutura geral de um sistema embarcado utilizando um contêiner Docker. ……. 71

Figura 30 - Resultado visual da aplicação de classificação de imagem. …….. 73

Figura 31 – Exemplo de uma boa detecção de classes pelo modelo. …….. 75

Figura 32 – Exemplo de uma detecção falha pelo modelo (rio confundido com floresta). ……. 76

Figura 33 - Composição espectral de imagens de solo e plantações de batata, chá e repolho. ……. 77

SUMÁRIO

1 INTRODUÇÃO 17

2 CONTEXTUALIZAÇÃO 19

2.1 CARACTERÍSTICAS GEOGRÁFICAS 19

2.2 CARACTERÍSTICAS SOCIOAMBIENTAIS 20

2.3 PROPOSTA DE TRABALHO 22

3 METODOLOGIA 27

3.1 COMPOSIÇÃO DO ACERVO DAS IMAGENS ORIGINAIS 28

3.2 COMPOSIÇÃO DO ACERVO DAS IMAGENS DE REFERÊNCIA 30

3.3 EXTRAÇÃO DE *FEATURES* – APLICAÇÃO DE FILTROS 32

3.3.1 Pixeis Originais 33

3.3.2 Filtros Gabor 33

3.3.3 Filtro Canny 35

3.3.4 Filtro Roberts 36

3.3.5 Filtro Sobel 37

3.3.6 Filtro Prewitt 38

3.3.7 Filtro Gaussiano 39

3.3.8 Filtro Mediana 40

3.3.9 Filtro Chan-Vese 41

3.4 MAPEAMENTO E ORGANIZAÇÃO DOS PIXEIS EM MATRIZ 42

3.5 DIVISÃO DAS MATRIZES ENTRE TREINO E TESTE 43

3.6 TREINO E TESTE DO MODELO DE *MACHINE LEARNING* 44

3.6.1 Algoritmo *Stochastic Gradient Descent* (SGD) 45

3.6.2 Parâmetros do Algoritmo 50

3.6.3 Treino do Modelo 51

3.6.4 Teste do Modelo 51

3.6.4.1 Matriz de Confusão 52

3.6.4.2 Exatidão 53

3.6.4.3 Precisão 54

3.6.4.4 *Recall* 54

3.6.4.5 *F1-Score* 54

4 RESULTADOS 57

4.1 EXECUÇÃO DO CÓDIGO 57

4.2 MATRIZ DE CONFUSÃO DO MODELO 57

4.3 EXATIDÃO DO MODELO 59

4.4 PRECISÃO DO MODELO 61

4.5 *RECALL* DO MODELO 63

4.6 *F1-SCORE* DO MODELO 65

5 APLICAÇÃO 67

5.1 VISÃO GERAL DO SISTEMA EMBARCADO UTILIZADO 67

5.1.1 Apalis iMX8QM 68

5.1.2 Placa base Ixora 69

5.1.3 Torizon OS 69

5.2 CONTEINERIZAÇÃO DA APLICAÇÃO 70

5.3 RESULTADO DA APLICAÇÃO 72

6 CONCLUSÃO 75

APÊNDICE A - CÓDIGO COMENTADO DA OBTENÇÃO DO MODELO DE *MACHINE LEARNING* 79

REFERÊNCIAS 87

1 INTRODUÇÃO

Com o passar dos anos, busca-se cada vez mais medidas de preservação da natureza frente aos problemas climáticos enfrentados desde meados do século XIX. Nesse contexto, a Floresta Amazônica chama a atenção do mundo quanto à luta para sua conservação, uma vez que é extremamente importante para regular o clima e os níveis de carbono (ALSHEHRI; OUADOU; SCOTT, 2023). Essa luta se dá pela exploração do bioma através do garimpo e da substituição da floresta pela pecuária e agricultura em larga escalas através do desmatamento (Rivero et al.[1], 2009 *apud* ZANOTTA et al., Automatic Methodology for Mass Detection of Past Deforestation in Brazilian Amazon, 2019). Consequentemente, o desmatamento traz consequências não só para o clima, mas também para a biodiversidade e a qualidade do solo e da água (Foley et al.[2], 2005 *apud* PISL, Classification of Tropical Deforestation Drivers with Machine Learning and Satellite Image Time Series, 2023).

Nesse sentido, na tentativa de colaborar com a preservação do meio ambiente, técnicas de inteligência artificial, como o aprendizado de máquina (*machine learning*) podem contribuir para ajudar no problema do desmatamento. Com isso, através da classificação de imagens por segmentação, é possível determinar regiões de uma foto que contenham áreas de floresta, bem como outras classes, para compreender o ambiente a ser analisado.

A segmentação de imagem possui diversas aplicações (AKAL; BARBU, 2019), dentre elas a detecção de objetos/regiões em uma imagem. Dessa forma, a classificação de imagem, que é um processo de visão computacional, classifica imagens baseado em seu conteúdo, direcionando para alguma das categorias predefinidas (CHUGH et al., 2020).

No entanto, uma vez que as imagens podem conter mais de uma classificação, este trabalho tende a analisar pixel a pixel de cada imagem com o

1 RIVERO, S. et al. Pecuária e desmatamento: uma análise das principais causas diretas do desmatamento na Amazônia, Revista Nova Economia Belo Horizonte - MG, vol. 19, no. 1, pp. 41-66, 2009.

2 FOLEY, J. et al. Global consequences of land use, Science (New York N.Y.), vol. 309, Aug. 2005.

intuito de classificá-los nas diferentes categorias. As categorias escolhidas foram: Vegetação de Floresta, que compreende toda área com árvores ou floresta densa; Vegetação Rasteira ou Arbustos, compreendendo gramados, campos e arbustos; e Sem Vegetação, que inclui construções humanas, estradas e vias, rios, lagos e regiões desmatadas.

As imagens a serem analisadas foram retiradas do programa Google Earth, que disponibiliza imagens de satélite de diferentes regiões do mundo. Assim, foram capturadas fotos da região amazônica com diversos elementos, dentre eles florestas, rios, lagos, gramados, e mais.

Para gerar o modelo de classificação de imagens, é necessário utilizar de algum algoritmo de *machine learning*. Neste trabalho, foi utilizado o algoritmo de Descida Gradiente Estocástica junto do classificador de *Support Vector Machine*, explicados ao longo deste trabalho.

Com isso, espera-se obter um modelo com exatidão elevada, capaz de detectar com boa precisão os diferentes elementos de uma foto, classificando-os dentre as três categorias preestabelecidas. Além disso, com o modelo pronto, a proposta final deste trabalho é de implementá-lo em um sistema embarcado, com o intuito de, em um novo projeto, acoplá-lo em um *drone* para que a análise das imagens seja feita em tempo real. A partir disso, o objetivo final deste trabalho é fornecer informações visuais de possíveis mudanças no ambiente dado uma área geográfica específica, de modo que, ao longo do tempo, possa-se analisar se a determinada região sofreu desmatamento. Assim, com essas informações, é possível que o poder público e autoridades competentes possam promover ações que mitiguem a destruição da Floresta Amazônica.

2 CONTEXTUALIZAÇÃO

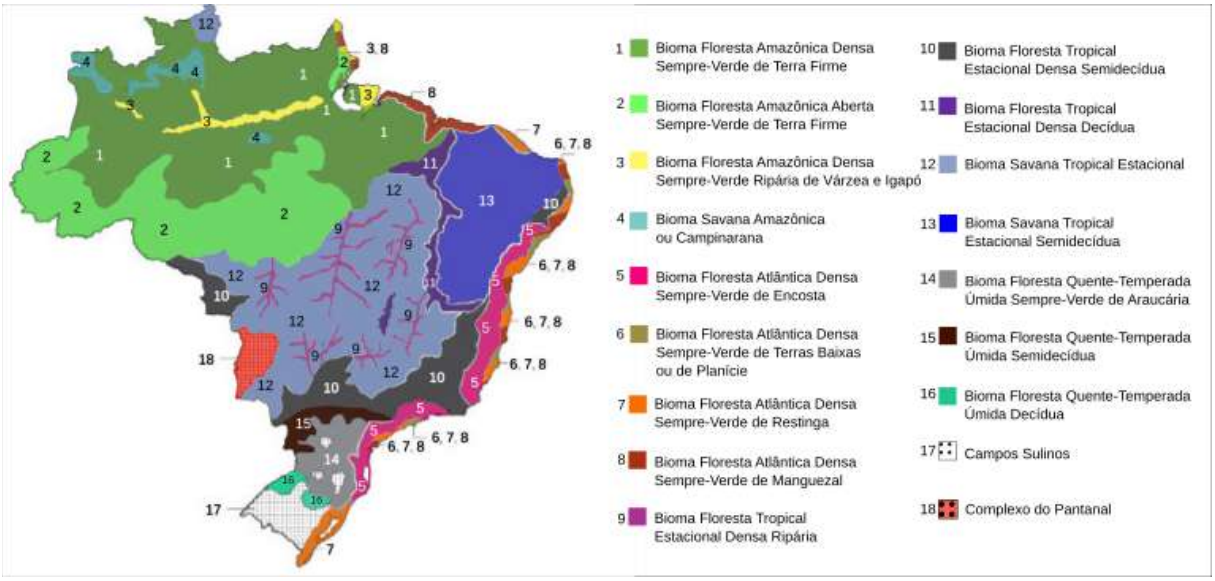
Com o intuito de apresentar um contexto para a aplicação deste trabalho, as seções seguintes constroem o panorama necessário para compreender o cenário atual do Brasil quanto à Floresta Amazônica, de modo que os resultados deste trabalho possam colaborar com sua preservação.

2.1 CARACTERÍSTICAS GEOGRÁFICAS

A região amazônica ocupa um extenso território na América do Sul. Sua cobertura compreende áreas da Bolívia, Peru, Equador, Colômbia, Venezuela, Guiana, Guiana Francesa, Suriname e Brasil, o qual possui cerca de 60% de todo o bioma. Estima-se que 15% da biodiversidade do planeta esteja presente nesse ecossistema (SECAS…, 2013), sendo conhecido apenas “300 espécies de mamíferos, mais de 1.000 de aves, 240 de répteis, 600 de anfíbios, 3.000 de formigas, 3.000 de abelhas e 1.800 de borboletas” (BIOMAS…, 2016).

Além da exuberância da fauna amazônica, a vegetação e solo também são bastante diversos. No contexto nacional, embora a Floresta Amazônica seja conhecida como apenas um bioma com características uniformes em toda sua área, Biomas… (2016) expõe um conjunto de quatro deles, com características distintas: Floresta Amazônica Densa Sempre-Verde de Terra Firme, Floresta Amazônica Aberta Sempre-Verde de Terra Firme, Floresta Amazônica Densa Sempre-Verde Ripária de Várzea e Igapó, e Savana Amazônica ou Campinarana.

Figura 1 – Mapa dos principais biomas no Brasil, Pantanal e Campos Sulinos.



Fonte: Editado pelo autor[3].

Com foco nos dois maiores biomas amazônicos, o de Floresta Amazônica Densa Sempre-Verde de Terra Firme é constituído por centenas de espécies de árvores densamente posicionadas, formando um dossel (“teto”) em suas copas, o que promove sombra em seu interior. Com características semelhantes, o bioma Floresta Amazônica Aberta Sempre-Verde de Terra Firme se diferencia por ter árvores mais espaçadas e cobertas por cipó, permitindo maior entrada de luz solar (BIOMAS…, 2016).

2.2 CARACTERÍSTICAS SOCIOAMBIENTAIS

Apesar do enorme potencial natural a ser estudado através da biodiversidade e recursos hídricos, além da importância e influência no clima global (BECKER; STENNER, 2008, p.8), a Floresta Amazônica enfrenta o contínuo avanço predatório das ações antrópicas. Essas, acometem porções relevantes do território com o objetivo de extrair seus recursos, ou então somente substituir o terreno por alguma atividade que consideram mais lucrativa.

3 Edição de imagem retirada de . Biomas brasileiros. 1. ed. São Paulo: Oficina de Textos, 2016.

Nesse sentido, o desmatamento se encontra no cerne da questão, pois é o responsável por viabilizar desastres ecológicos em detrimento de um ecossistema em pleno equilíbrio e funcionamento. Dentre os desastres, as queimadas, o empobrecimento do solo e a redução da qualidade da água e do ar têm papel fundamental no declínio da Floresta Amazônica.

Os períodos de seca tendem a propiciar a ocorrência de queimadas. Através deles, há um decaimento no número de árvores e perda de folhas (Philips et al.[4], 2009 *apud* . Secas na Amazônia: causas e consequências, 2013), o que permite maior incidência de luz solar através da abertura do dossel, ocasionando no ressecamento do material orgânico e consequente vulnerabilidade à queimadas (SECAS…, 2013). No entanto, Secas… (2013) conclui que apesar de o clima ser determinante no aumento das queimadas, o uso do solo garante que aconteçam. Dessa forma, a ação do ser humano por meio do corte seletivo e desmatamento intensifica a probabilidade de queimadas (Uhl; Kauffman[5], 1990; Cochrane et al.[6], 1999; Cochrane; Schulze[7], 1999; Barlow; Peres[8], 2004; Nepstad et al.[9], 2004 *apud* . Secas na Amazônia: causas e consequências, 2013), além da redução da precipitação (Laurance; Williamson[10], 2001; Laurance et al.[11], 2002; Nobre; Sellers;

4 PHILLIPS, O. L. et al. Drought sensitivity of the Amazon rainforest. Science , v. 323, p. 1344-1347, 2009.

5 UHL, C.; KAUFFMAN, J. B. Deforestation, fire susceptibility, and potential tree responses to fire in the eastern amazon. Ecology , v. 71, n. 2, p. 437-449, 1990.

6 COCHRANE, M. A. et al. Positive feedbacks in the fire dynamic of closed canopy tropical forests. Science , v. 284, n. 5421, p. 1832-1835, 1999.

7 COCHRANE, M. A.; SCHULZE, M. D. Fire as a recurrent event in tropical forests of the Eastern Amazon: Effects on Forest Structure, Biomass, and Species Composition. Biotropica , v. 31, p. 1, 2-16, 1999.

8 BARLOW, J.; PERES, C. A. Ecological responses to El Nino-induced surface fires in central Brazilian Amazonia: management implications for flammable tropical forests. Philosophical Transactions of the Royal Society B-Biological Sciences, v. 359, p. 1443, 367-380, 2004.

9 NEPSTAD, D. et al. Amazon drought and its implications for forest flammability and tree growth: a basin-wide analysis. Glob. Change Biol ., v. 10, p. 704-717, 2004.

10 LAURANCE, W. F.; WILLIAMSON, G. B. Positive feedbacks among forest fragmentation, drought, and climate change in the Amazon. Conservation Biology , v. 15, n. 6, p. 1529-1535, 2001.

11 LAURANCE, W. F. et al. Ecosystem decay of Amazonian forest fragments: A 22-year investigation. Conservation Biology , v. 16, n. 3, p. 605-618, 2002.

Shukla[12], 1991; Silva Dias et al.[13], 2005; Costa et al.[14], 2007 *apud* . Secas na Amazônia: causas e consequências, 2013).

Ademais, apesar de a Floresta Amazônica oferecer uma beleza natural ímpar, com árvores atingindo dezenas de metros de altura, seu solo é considerado escasso em nutrientes. Dessa forma, o que mantém a exuberância da floresta é a eficiência na reciclagem de nutrientes através do retorno constante da biomassa ao solo e sua reabsorção facilitada por meio de fungos nas raízes das árvores (BIOMAS…, 2016). Todavia, ainda que a pobreza do solo seja uma característica natural, Biomas… (2016) enfatiza que o desmatamento e as queimadas tendem a piorar essa situação, quebrando o ciclo de renovação dos nutrientes, pois como não há árvores, estes são lixiviados, sendo levados aos rios pelas chuvas, ou perdidos para a atmosfera.

Por fim, as explorações agropecuária, madeireira e fundiária (seja pelo uso industrial, ou pelo garimpo) impulsionam o desmatamento na região amazônica, afetando as bacias dos principais afluentes do rio Amazonas, contaminando-as através da fumaça, insumos agrícolas e do mercúrio por meio das atividades de garimpo (BECKER; STENNER, 2008). Becker e Stenner (2008) evidenciam que o Brasil se encontra no grupo dos dez maiores emissores de gás carbônico, sendo que o maior responsável é o desmatamento da Amazônia. Além disso, os autores apontam que “o aumento em curso da demanda global por produtos intensivos em água, como a carne e a soja, representa uma pressão extra sobre as grandes reservas d'águas”, motivando a substituição da floresta pelo pasto.

2.3 PROPOSTA DE TRABALHO

Em vista do que foi discutido nas seções anteriores, este trabalho tem a intenção de proporcionar uma análise do desmatamento na Floresta Amazônica,

12 NOBRE, C. A.; SELLERS, P. J.; SHUKLA, J. Amazonian deforestation and regional climate change. Journal of Climate , v. 4, n. 4, p. 957-988, 1991.
13 SILVA DIAS, M. A. F.; COHEN, J. C. P.; GANDU, A. W. Clouds, rain and biosphere interactions in Amazon. Acta Amazonica , v. 35, n. 2, p. 215-222, 2005.
14 COSTA, M. H. et al. Climate change in Amazonia caused by soybean cropland expansion as compared to caused by pastureland expansion. Geophysical Research Letters , v. 34, 2007.

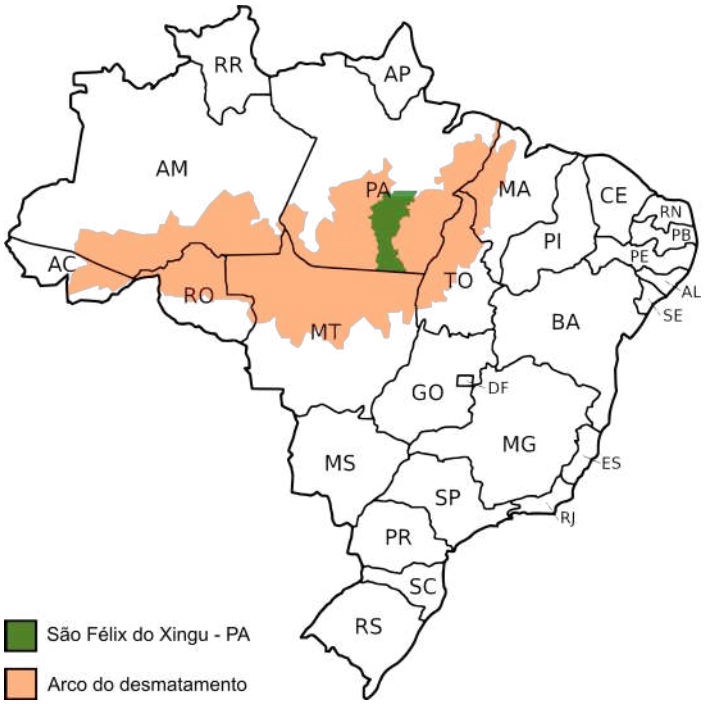
com o intuito de evidenciar regiões com e sem cobertura vegetal de floresta por meio de imagens aéreas.

A área amazônica a ser estudada encontra-se no “Arco do desmatamento”. De acordo com o Instituto de Pesquisa Ambiental da Amazônia (IPAM), o Arco do desmatamento é a:

> Região onde a fronteira agrícola avança em direção à floresta e também onde encontram-se os maiores índices de desmatamento da Amazônia. São 500 mil km² de terras que vão do leste e sul do Pará em direção oeste, passando por Mato Grosso, Rondônia e Acre.

A Figura 2 destaca a região correspondente ao Arco do desmatamento. Como exposto anteriormente, a área em destaque tem predominância dos biomas Floresta Amazônica Aberta Sempre-Verde de Terra Firme e Floresta Amazônica Densa Sempre-Verde de Terra Firme. Contudo, o recorte a ser analisado compreende as proximidades de São Félix do Xingu-PA, tendo em vista a degradação do alto curso do rio Tapajós devido à atividade humana (BECKER; STENNER, 2008), além de ser a cidade que mais emite CO_2 do Brasil pelo mesmo motivo (GRILLI, 2021).

Figura 2 – Arco do desmatamento.



Fonte: Editado pelo autor[15].

Dessa forma, a partir da definição da área de interesse, o passo seguinte será obter imagens aéreas do local. Para isso, considerando as limitações deste trabalho, as fotografias serão obtidas pelo Google Earth, software que permite a visualização da Terra através de imagens de satélite. As imagens capturadas terão distância do solo de 600 metros.

Em seguida, será aplicado um modelo de visão computacional que analisará as imagens, tendo-as como base para treinar a inteligência artificial. Os métodos discutidos em seções posteriores terão como objetivo identificar três categorias: regiões com floresta (Vegetação de Floresta), regiões de vegetação não arbórea (Vegetação Rasteira e Arbustos) e áreas urbanas, rios e lagos (Sem Vegetação). Com isso, espera-se analisar regiões desmatadas e com recuperação de floresta ao longo do tempo, monitorando a Floresta Amazônica com o passar dos anos, e

15 Edição e compilação de imagens retiradas de Mapa do Brasil, Info Escola – https://www.infoescola.com/geografia/mapa-do-brasil/; Fronteira do desmatamento na Amazônia avançou entre 2018 e 2019, afirma estudo, O Globo, 2019 – https://oglobo.globo.com/brasil/fronteira-do-desmatamento-na-amazonia-avancou-entre-2018-2019-afirma-estudo-1-24141480; São Félix do Xingu, Wikipedia – https://pt.wikipedia.org/wiki/S%C3%A3o_F%C3%A9lix_do_Xingu.

oferecendo dados para basear políticas públicas ambientais. Vale mencionar que o objetivo deste trabalho é treinar o modelo matemático para que possa ser utilizado periodicamente em campo.

Com o modelo treinado, será feito sua integração a um hardware embarcado, de modo que as imagens sejam processadas nele. Nesse sentido, este trabalho pode servir como alicerce para outros projetos que queiram, por exemplo, analisar em tempo real a região amazônica. Ainda, havendo a possibilidade de adaptar o hardware embarcado a um *drone* (veículo aéreo não tripulado) com uma câmera apropriada, é possível fotografar e processar a imagem de maneira simultânea.

3 METODOLOGIA

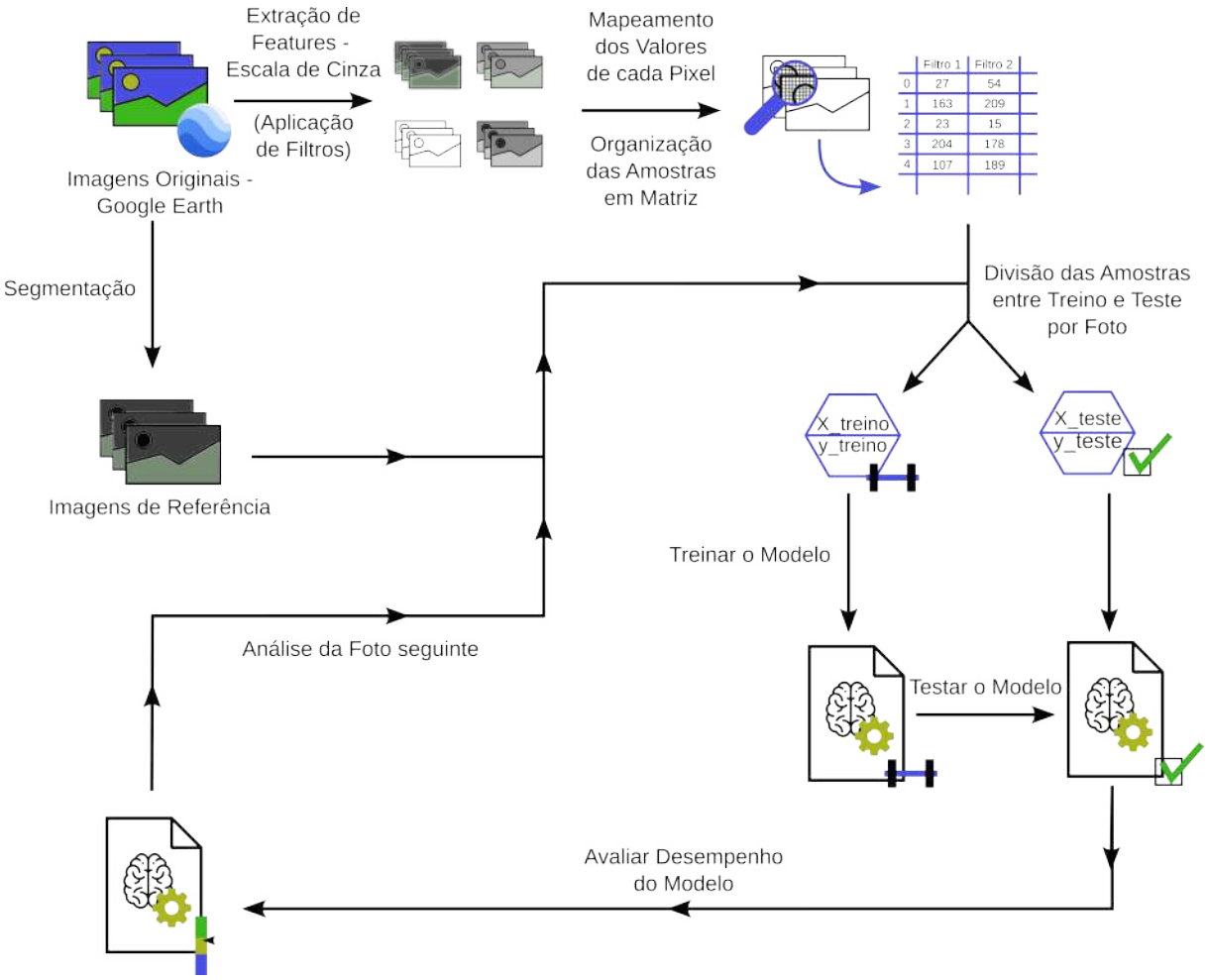
Tendo em vista a situação atual da Floresta Amazônica apresentada anteriormente, vê-se necessário planos de ação para contribuir com sua manutenção. Nesse sentido, a proposta estudada nesta seção tem como objetivo mapear a região amazônica através de fotos de satélite, de modo a segmentá-las em três classes: Vegetação de Floresta, Vegetação Rasteira e Arbustos, e Sem Vegetação (como rios, lagos e elementos de intervenção humana – edificações, vias e áreas desmatadas). Com isso, é possível determinar ao longo do tempo mudanças na paisagem de áreas específicas a partir do aumento ou diminuição das regiões classificadas.

A análise de classificação das regiões das fotos é feita através de aprendizado de máquina tradicional (do inglês *Traditional Machine Learning*, referenciado ao longo do texto como ML) com a linguagem de programação Python. Sendo uma área da Inteligência Artificial, ML compreende a análise de dados e algoritmos com a intenção de simular o aprendizado humano, melhorando-o gradualmente (IBM, 2023). Assim, o modelo de aprendizado de máquina deste trabalho é treinado imagem por imagem, de modo a assimilar diferentes texturas, contornos e tonalidades para definir em novas fotos as diferentes classificações possíveis.

A figura abaixo ilustra todo o processo discutido nesta seção. Primeiramente, as imagens originais são reunidas, servindo de base para realizar a classificação nas imagens de referência. Logo, são aplicados filtros nas imagens originais com o intuito de destacar formas e luminosidade para facilitar sua compreensão. Assim, cada imagem é mapeada pixel a pixel, sendo seus dados então organizados em matrizes. Em sequência, cada um dos pixeis de cada imagem original, carregando consigo suas características definidas pelos filtros, é alocado ou em um grupo de treino ou um de teste. Dessa forma, o respectivo pixel da respetiva imagem de referência é designado a um grupo próprio de mesma classe (treino ou teste). Por fim, o modelo é treinado com os grupos de treino de cada imagem e então testado com os grupos de teste de todas as imagens treinadas até o momento para assim

obter-se informações importantes como a exatidão. Após isso, todo o processo de alocação dos grupos de treino e teste, treinamento e depois teste do modelo é repetido para cada imagem.

Figura 3 – Fluxograma do processo de treinamento e teste do modelo.



Fonte: Compilação do autor.

3.1 COMPOSIÇÃO DO ACERVO DAS IMAGENS ORIGINAIS

Considerando que modelos de *machine learning* necessitam de grandes quantidades de dados como amostras para seu treinamento, é esperado que neste caso o banco de fotos seja suficientemente grande. No entanto, é preciso definir o que são amostras no caso em questão.

De acordo com a página do Governo do Canadá (*Government of Canada*, 2023), na seção “Satellite imagery, elevation data, and air photos”, a classificação digital de imagem:

> […] utiliza a informação espectral representada pelos números digitais em uma ou mais bandas espectrais, e tenta classificar cada pixel baseado nessa informação espectral. […] o objetivo é atribuir todos os pixeis de uma imagem a classes ou temas específicos.

Em outras palavras, cada pixel é representado por um ou mais valores pelos quais, em conjunto, definem a qual categoria de classificação aquele pixel pertence. Como exemplo, dada uma imagem qualquer na escala RGB (*Red, Green, Blue* – Vermelho, Verde e Azul, em português) de tamanho 1920 x 1080 pixels (px), esta é interpretada computacionalmente como uma matriz de três dimensões: 1920 por 1080 pixeis (duas dimensões), tendo cada pixel três valores (terceira dimensão - cada um deles indicando o valor de cada matiz dos quais vermelho, verde e azul). Assim, considerando essa imagem como um *array* da biblioteca utilizada *NumPy*[16], dada a função *shape*[17], a representação seria a seguinte: (1920, 1080, 3). Para facilitar a computação dos dados, e neste caso para adequá-los aos requisitos das bibliotecas trabalhadas em Python, as imagens são redimensionadas e convertidas em escala de cinza, sendo então representadas por uma matriz de duas dimensões: 2.073.600 linhas (multiplicação de 1920 por 1080, agora em um mesmo eixo) por uma coluna, representando os valores de cada pixel na escala cinza. Portanto, considerando a representação da imagem como um *array*, esse seria o formato: (2073600, 1).

Nesse sentido, neste trabalho, cada amostra é uma linha dessa matriz de duas dimensões, cujo valor do pixel esperado (a classe que pertence) encontra-se na matriz de duas dimensões da imagem correspondente de referência com o mesmo índice.

A partir disso, para o modelo em questão foram reunidas 20 fotos de satélite, retiradas do Google Earth, de tamanho 1920 x 1080 px da região amazônica (mais

16 Para mais informações sobre *array*, acesse: https://numpy.org/doc/stable/reference/generated/numpy.array.html.

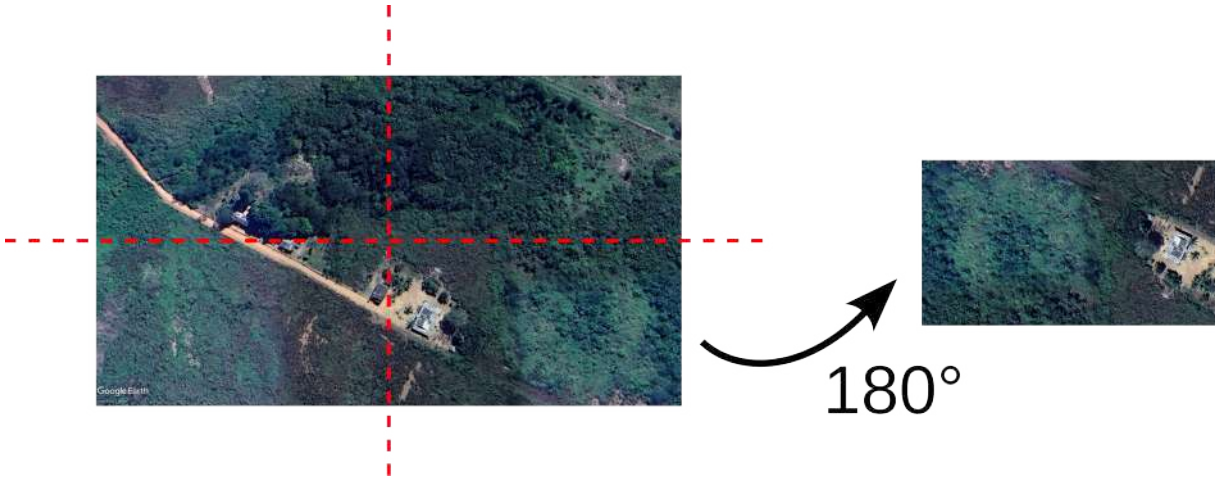
17 Mais informações sobre a função em: https://numpy.org/doc/stable/reference/generated/numpy.ndarray.shape.html.

precisamente da cidade de São Félix do Xingu – PA). As imagens foram capturadas a uma distância de 600 m do solo.

Assim, considerando as limitações de hardware do computador utilizado para treinar o modelo de ML (processador Intel® Core™ i3-8130U CPU @ 2.20GHz, 2 núcleos, 4 *threads*; 20 GB de RAM), essas imagens foram divididas em quatro, totalizando 80 fotos de 960 x 540 px. Ainda, com o intuito de aumentar a quantidade de fotos, cada uma delas teve uma cópia rotacionada em 180°.

Portanto, o banco de imagens originais totaliza 160 fotos, cada uma de tamanho 960 x 540 px. Além disso, o número de amostras total é então de 82.944.000, que são divididos em grupos de treino e teste, como discutido anteriormente.

Figura 4 – Imagem dividida em quatro, com suas respectivas partes rotacionadas em 180°.



Fonte: Compilação do autor.

3.2 COMPOSIÇÃO DO ACERVO DAS IMAGENS DE REFERÊNCIA

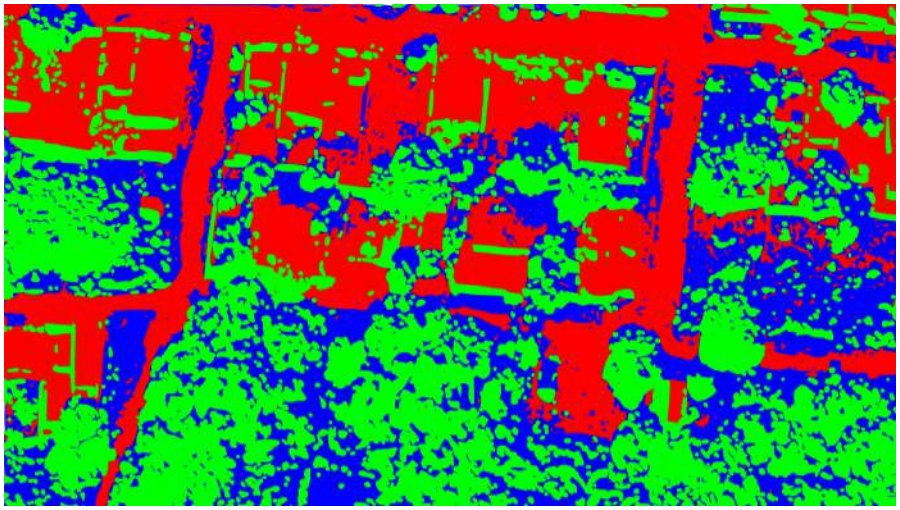
Baseadas em suas respectivas imagens originais, as imagens de referência são aquelas que foram segmentadas dentre as três categorias:

- Vegetação de Floresta (VF)
- Vegetação Rasteira e Arbustos (VRA)

- Sem Vegetação (SV)

Com isso, cada uma das classificações foi representada por uma cor na escala RGB: VF representada por verde (0, 255, 0), VRA por azul (0, 0, 255) e SV por vermelho (255, 0, 0).

Figura 5 – Exemplo de imagem de referência segmentada na escala RGB.



Fonte: Compilação do autor.

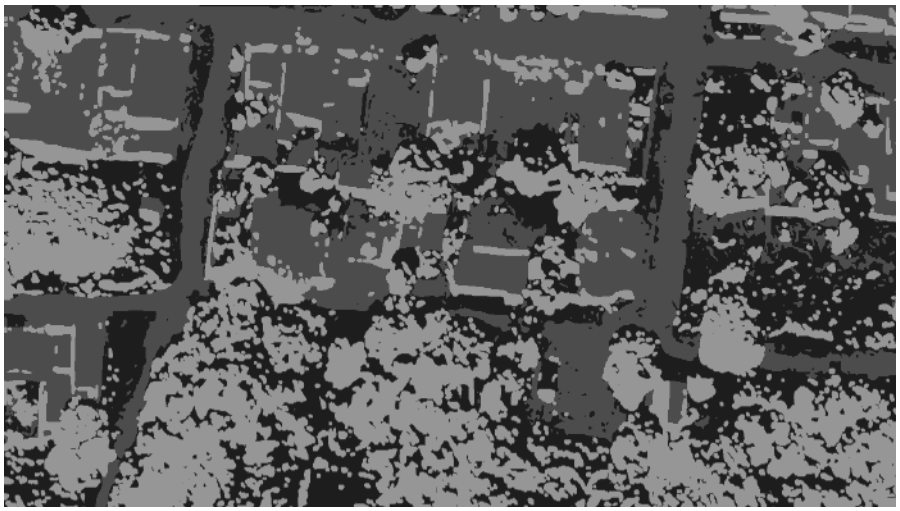
A segmentação foi realizada no editor de imagens Inkscape[18], sendo cada categoria definida manualmente com as ferramentas do software. No entanto, percebeu-se após alguns testes do modelo que a quantidade de classificações era bastante superior a três. A partir disso, é possível que no momento de exportar as imagens de referência do editor em formato PNG, as fronteiras entre as cores tivessem valores RGB diferentes, e quando convertidos na escala de cinza, teriam também valores diferentes.

Com isso, foi necessário realizar uma outra classificação. O primeiro ponto foi considerar que a conversão do vermelho (RGB: 255, 0, 0) para escala de cinza é 76, e da mesma forma para o verde (RGB: 0, 255, 0) sendo 150 e para o azul (RGB: 0, 0, 255) sendo 29. Então, pixel a pixel de cada imagem de referência, depois de ser convertida na escala de cinza, foi analisado e se estivesse dentro dos limites de 0 e

18 Mais informações sobre o software em: https://inkscape.org/about/overview/.

52 seria categorizado como 29 (azul), de 53 a 113 como 76 (vermelho), e de 114 a 255 como 150 (verde). Dessa forma, foram geradas outras 160 imagens de referência com três classificações e na escala de cinza.

Figura 6 – Exemplo de imagem de referência segmentada na escala de cinza.



Fonte: Compilação do autor.

3.3 EXTRAÇÃO DE *FEATURES* – APLICAÇÃO DE FILTROS

Como visto anteriormente, no caso da classificação de imagem deste trabalho, as amostras para treinar e testar o modelo de ML são pixeis com diferentes valores, analisados em conjunto para determinar a qual categoria pertence cada pixel. Nesse sentido, cada um desses valores é indicado como uma *feature (*um atributo, uma característica) do pixel. Assim, um pixel pode ser entendido como um *array*, um arranjo de atributos.

Esses atributos são obtidos por meio da aplicação de filtros nas imagens originais, de modo que realcem estruturas das fotos, como contornos, texturas e intensidade de brilho. Segundo Shijin Kumar e Dharun (2016), “a extração de características reduz o tamanho de dados de uma imagem, obtendo informações necessárias da imagem [a ser] segmentada”. Assim, a grande quantidade de

informações que cada pixel possui em uma dimensão é filtrada e separada em várias camadas com as características mais importantes.

Contudo, Shijin Kumar e Dharun (2016) afirmam que obter as melhores *features* pode ser difícil e levar tempo. Por outro lado, fornecer os atributos adequados ao modelo de ML proporciona exatidão e eficiência na tomada de decisão (SHAHAJAD; GAMBHIR; GANDHI, 2021).

A partir disso, nas subseções a seguir são apresentados os filtros utilizados para detectar as características dos pixeis das imagens originais. É importante mencionar que ao aplicar cada filtro, as respectivas funções em Python recebem a imagem já convertida em escala de cinza, filtram e transformam-na em uma dimensão. Logo, os valores são organizados em *dataframes* da biblioteca *pandas*: estruturas de dados tabular de duas dimensões (pandas, 2023).

A estratégia de extração de *features* por meio de filtros foi retirada de Bhattiprolu (2019).

3.3.1 Pixeis Originais

Considerando os valores originais dos pixeis das imagens retiradas do Google Earth após o processamento na escala de cinza, é possível que nem todas as informações importantes de cada um deles sejam percebidas pelos filtros aplicados. Devido a isso, como estratégia, utilizou-se os valores originais dos pixeis como uma das *features*.

3.3.2 Filtros Gabor

Os filtros Gabor são produtos da modulação de sinais sinusoidais e Gaussianos (MEHROTRA; NAMUDURI; RANGANATHAN, 1992). A partir do estudo

aprofundado e da representação matemática, modelos computacionais baseados em Gabor surgiram para detectar contornos e texturas, além de proporcionar estimativas de fluxo ótico e compactação de dados de imagem (Manjunath; CheUappa[19], 1991; Porat; Zeevi[20], 1989; Daugman[21], 1988 *apud* MEHROTRA R., NAMUDURI K.R., RANGANATHAN N. Gabor filter-based edge detection, Pattern Recognition, 1992).

O filtro Gabor é representado por uma expressão matemática no plano dos complexos, tendo uma parte real e outra imaginária. Abaixo encontra-se a expressão, na qual λ é o comprimento de onda do sinal sinusoidal, θ é a orientação das faixas normais às paralelas da função Gabor, Φ é o deslocamento de fase, σ é o desvio padrão do sinal Gaussiano, γ é a proporção espacial, e x e y são as coordenadas da imagem.

$$g(x,y;\lambda,\theta,\Phi,\sigma,\gamma)=\exp\left(\frac{-x'^2+\gamma^2 y'^2}{2\sigma^2}\right)\exp\left(i\left(2\pi\frac{x'}{\lambda}+\Phi\right)\right)$$

Onde:

$$x'=x\cos\theta+y\,sen\theta \qquad y'=-x\,sen\theta+y\cos\theta$$

Após alguns testes, foram definidos os valores a seguir:

$$\lambda=[0,\frac{\pi}{4},\frac{2\pi}{4},\frac{3\pi}{4}];\theta=[\frac{\pi}{4},\frac{2\pi}{4},\frac{3\pi}{4}];\Phi=0;\sigma=[1,3];\gamma=[0.05,0.5]$$

Os filtros Gabor inseridos no código em Python foram aplicados através das funções *getGaborKernel()* e *filter2D()* da biblioteca OpenCV[22]. Assim, considerando todas as permutações possíveis, obteve-se 48 filtros Gabor para cada imagem original, ou seja, 48 atributos diferentes para cada pixel de cada imagem. Abaixo há um exemplo de foto com um dos filtros Gabor.

---

19 MANJUNATH, B.S. and CHEUAPPA, R. A computational approach to boundary detection, Proc. CVPR 91, pp. 358-363, 1991.

20 PORAT, M. and ZEEVI, Y. Y. The generalized Gabor scheme of image representation in biological and machine vision, IEEE Trans. Pattern Analysis Mach. LntelL 10(4), 452-468, 1989.

21 DAUGMAN, J.G. Complete 2-D Gabor transforms by neural networks for image analysis and compression, IEEE Trans. ASSP 36(7), 1169-1179, 1988.

22 Mais informações sobre as funções em: https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#gae84c92d248183bd92fa713ce51cc3599 e https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#ga27c049795ce870216ddfb366086b5a04.

Figura 7 – Exemplo de imagem original e sua respectiva versão com um dos filtros Gabor.

Fonte: Google Earth (2023); Compilação do autor.

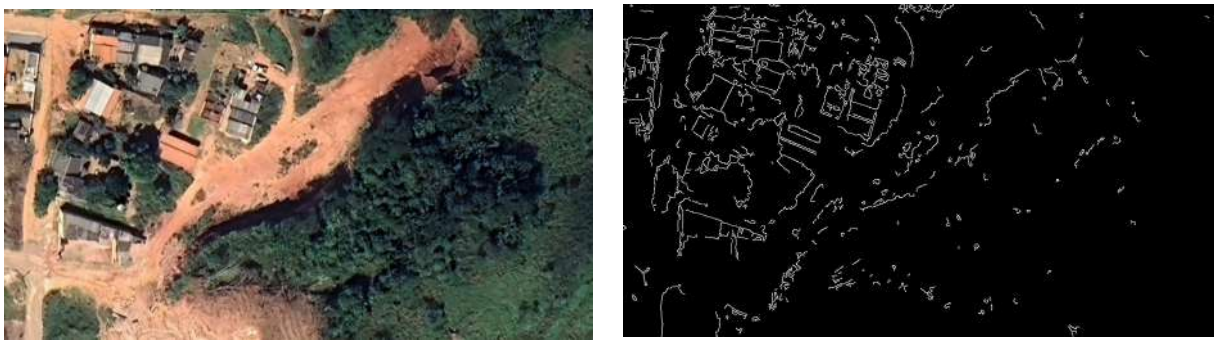
3.3.3 Filtro Canny

O filtro Canny é amplamente usado em visão computacional como forma de detectar contornos e mudanças acentuadas de intensidade (DING; GOSHTASBY, 2001). Nesse sentido, o filtro Canny é operado por convolução da primeira derivada da função Gaussiana, que indica o ponto de máximo dela, representando o contorno (UFF, 2023). Segundo Ding e Goshtasby (2001), o filtro “classifica um pixel como borda se seu gradiente de magnitude for maior do que o daqueles que estão ao seu redor na direção de máxima mudança de intensidade”. Assim, são definidos limiares superior e inferior de gradientes para servir como referência de quais pixeis devem ser considerados borda (método chamado *thresholding* com histerese).

O filtro foi aplicado no código através da função *Canny()* da biblioteca OpenCV[23]. Abaixo apresenta-se um exemplo de imagem original após a aplicação de Canny.

23 Mais informações sobre a função em: https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga04723e007ed888ddf11d9ba04e2232de.

Figura 8 - Exemplo de imagem original e sua respectiva versão com o filtro Canny.



Fonte: Google Earth (2023); Compilação do autor.

3.3.4 Filtro Roberts

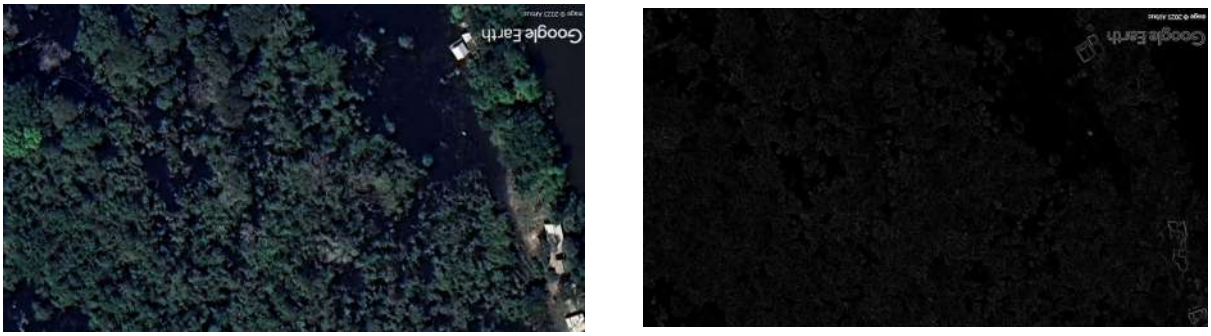
O filtro Roberts é utilizado para detectar linhas horizontais e verticais (Owotogbe; Ibiyemi; Adu[24], 2019 *apud* AMORIM, A.; POLASTRI, M. J. Evaluation of Edge Detection Filters Applied to Corroded Steel Sheets, 2020). O método utilizado é simples, através de medições de gradientes espaciais por meio de núcleos 2x2 px e suas cópias rotacionadas em 90° (FISHER et al., 2003-a). Dessa forma, o filtro realça regiões de alta frequência espacial que indicam as bordas.

No código, foi utilizada a função *roberts()* da biblioteca *scikit-image*[25]*.* A seguir, a aplicação de Roberts em uma das imagens originais.

24 OWOTOGBE, J. S., IBIYEMI, T. S., ADU, B. A. Edge Detection Techniques on Digital Images - A Review, Int. J. Innov. Sci. Res. Technol., vol. 4, no. 11, pp. 329–332, 2019.

25 Mais informações da função em: https://scikit-image.org/docs/stable/api/skimage.filters.html#skimage.filters.roberts.

Figura 9 - Exemplo de imagem original e sua respectiva versão com o filtro Roberts.



Fonte: Google Earth (2023); Compilação do autor.

3.3.5 Filtro Sobel

Semelhante ao filtro Roberts, o filtro Sobel tende a ser mais lento para computar. Além disso, em vez de um núcleo 2x2 px, utiliza um 3x3 px e sua cópia rotacionada em 90° (FISHER et al., 2003-b). A aplicação do filtro é ideal para encontrar bordas com curvas suaves (AMORIM; POLASTRI, 2020).

No código em Python, foram utilizadas as funções *sobel_h() e sobel_v()* da biblioteca *scikit-image*[26]. Abaixo encontra-se um exemplo de aplicação do filtro.

26 Mais informações das funções em: https://scikit-image.org/docs/stable/api/skimage.filters.html#skimage.filters.sobel_h e https://scikit-image.org/docs/stable/api/skimage.filters.html#skimage.filters.sobel_v.

Figura 10 - Exemplo de imagem original e sua respectiva versão com o filtro Sobel.

Fonte: Google Earth (2023); Compilação do autor.

3.3.6 Filtro Prewitt

O filtro Prewitt, assim como Roberts, é adequado para detectar bordas verticais e horizontais (AMORIM; POLASTRI, 2020). Ademais, Prewitt é também semelhante a Sobel, uma vez que também utiliza núcleos 3x3 px (The University of Auckland, 2023).

No código em Python, as funções *prewitt_h()* e *prewitt_v()* da biblioteca *scikit-image*[27] foram usadas. Abaixo encontra-se um exemplo de aplicação do filtro.

27 Mais informações das funções em: https://scikit-image.org/docs/stable/api/skimage.filters.html#skimage.filters.prewitt_h e https://scikit-image.org/docs/stable/api/skimage.filters.html#skimage.filters.prewitt_v.

Figura 11 - Exemplo de imagem original e sua respectiva versão com o filtro Prewitt.

Fonte: Google Earth (2023); Compilação do autor.

3.3.7 Filtro Gaussiano

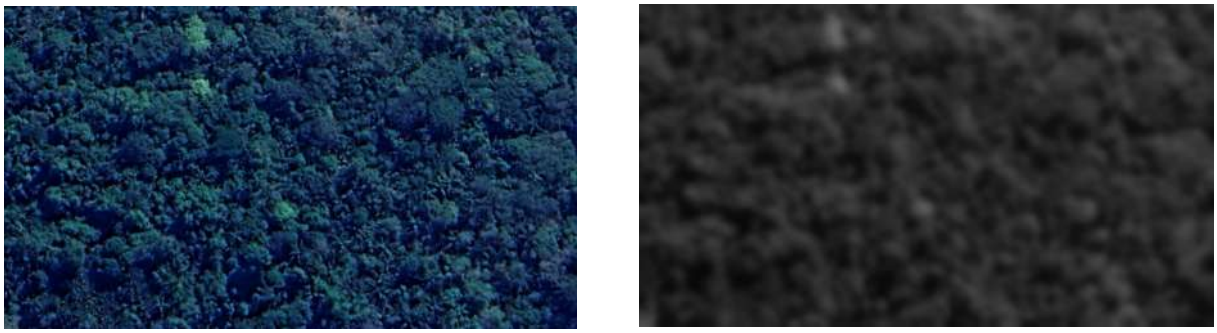
O filtro Gaussiano atua de modo a aplicar média de pesos nos pixeis de acordo com a distribuição Gaussiana (NIXON; AGUADO, 2002). Assim, o filtro consegue desfocar a imagem, removendo detalhes e ruído (FISHER et al., 2003-c). A expressão matemática é apresentada abaixo, na qual σ define o grau de desfoque, e x e y são as coordenadas da imagem.

$$G(x, y; \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(\frac{-x^2 + y^2}{2\sigma^2}\right)$$

Após testes, os valores de σ foram determinados como 1 e 3. O código em Python utilizou da biblioteca *SciPy* a função *gaussian_filter()*[28]. A seguir, um exemplo de foto com o filtro Gaussiano aplicado.

28 Mais informações sobre a função em: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.gaussian_filter.html.

Figura 12 - Exemplo de imagem original e sua respectiva versão com um dos filtros Gaussiano.



Fonte: Google Earth (2023); Compilação do autor.

3.3.8 Filtro Mediana

O filtro Mediana é usado para reduzir ruído, de modo a preservar detalhes da imagem (FISHER et al., 2003-d). Atua de modo a substituir o valor de um pixel pela mediana dos valores dos pixeis ao redor.

No código, foi utilizada a função *median_filter()* da biblioteca *SciPy*[29]. Abaixo há um exemplo de imagem com o filtro Mediana aplicado.

29 Mais informações sobre a função em: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.median_filter.html#scipy.ndimage.median_filter.

Figura 13 - Exemplo de imagem original e sua respectiva versão com o filtro Mediana.

Fonte: Google Earth (2023); Compilação do autor.

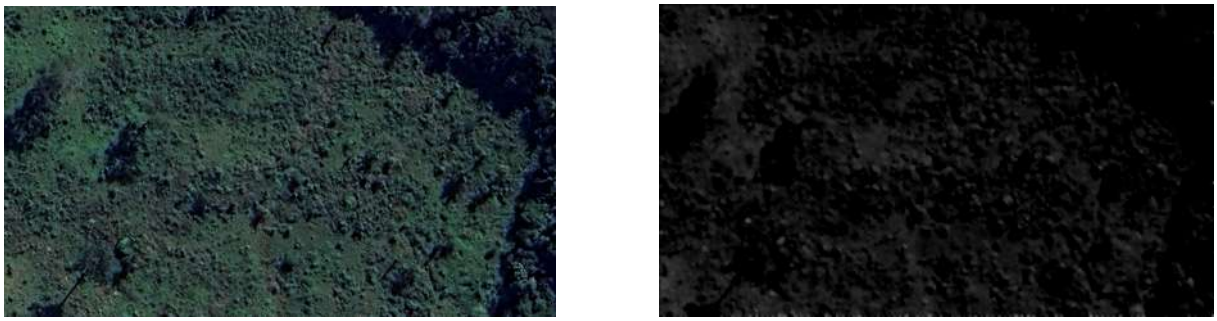
3.3.9 Filtro Chan-Vese

Segundo Akal e Barbu (2019), o filtro Chan-Vese aplica um método de baixo nível de segmentação de imagem, de modo a manter constante a regularização de comprimento das bordas em união com algoritmos de otimização para encontrar regiões de interesse. Em outras palavras, o filtro Chan-Vese não depende das bordas, mas sim de modelos de intensidade constante que definem regiões internas e externas para que assim as fronteiras sejam encontradas.

O código utilizou da função *chan_vese()* da biblioteca *scikit-image*[30]. Abaixo vê-se um exemplo de aplicação desse filtro.

30 Mais informações sobre a função em: https://scikit-image.org/docs/stable/api/skimage.segmentation.html#skimage.segmentation.chan_vese.

Figura 14 - Exemplo de imagem original e sua respectiva versão com o filtro Chan-Vese.



Fonte: Google Earth (2023); Compilação do autor.

3.4 MAPEAMENTO E ORGANIZAÇÃO DOS PIXEIS EM MATRIZ

Depois de definidos as 57 variações de filtros, é necessário aplicá-los a cada imagem. Assim, cada pixel é mapeado e recebe um atributo a partir de cada filtro. Com isso, no total, cada pixel é representado por um arranjo de 57 atributos. Após a aplicação de cada filtro, a imagem é redimensionada em apenas um eixo.

A partir disso, a cada imagem, os dados são organizados em um *dataframe* resultante de outros *dataframes* cujas informações provêm de cada aplicação de filtro. Um exemplo de *dataframe* é exibido a seguir:

	Gabor1	Gabor2	Gabor3	Gabor4	Gabor5	...	Prewitt	Gaussiana_s3	Gaussiana_s7	Mediana	Chan_Vese
0	0	0	104	92	27	...	4	42	60	18	10
1	0	0	127	112	9	...	26	45	61	18	19
2	0	0	182	163	27	...	50	50	61	32	20
3	0	0	255	241	34	...	57	56	62	44	20
4	0	0	255	255	35	...	75	63	62	63	15
...	...	...	...	...	...	...	...	...	...	...	...
518395	0	0	255	255	37	...	21	60	59	54	66
518396	0	0	255	255	38	...	27	61	59	63	146
518397	0	0	255	255	39	...	9	62	59	66	200
518398	0	0	255	255	42	...	5	62	59	67	240
518399	0	0	255	255	37	...	3	62	59	67	255

[518400 rows x 57 columns]

Em seguida, cada *dataframe* é alocado em um dicionário, cuja chave é o número da imagem.

De maneira análoga, o mesmo processo acontece com as imagens de referência. No entanto, em vez de de existirem *dataframes* que compõem um outro,

como só há uma dimensão (uma vez que são os valores de referência), só existe um *dataframe* no processo. Abaixo encontra-se a saída do *dataframe* de uma das imagens de referência.

```
        Imagens de Referência
0                   29
1                   29
2                   29
3                   29
4                   29
...                  ...
518395              29
518396              29
518397              29
518398              29
518399              29

[518400 rows x 1 columns]
```

Logo depois, ambos os dicionários gerados pelas funções que processam as imagens originais filtradas e as imagens de referência são convertidos em listas com apenas os *dataframes*. Isso acontece para que seja possível iterar sobre cada *dataframe* resultante de cada imagem original filtrada e respectiva imagem de referência.

3.5 DIVISÃO DAS MATRIZES ENTRE TREINO E TESTE

Na função principal do código, cada *dataframe* de cada imagem original filtrada e cada *dataframe* da respectiva imagem de referência são iterados por vez. Nesse sentido, respectivamente, os *dataframes* são denominados como dados *X* e *y*.

Assim, X e y são enviados para a função que tem o propósito de treinar o modelo de *machine learning* junto ao modelo parcial até aquele momento, além de todas as amostras de X e y de teste até então. Isso acontece a cada iteração para cada imagem, de modo que no início, o modelo parcial é exatamente o classificador com seus parâmetros pré-definidos (discutido na seção seguinte), e as amostras de treino são, respectivamente, um *dataframe* e um *array* vazios.

Dessa forma, já na função que treina o modelo de ML, X e y passam pela função *train_test_split()* da biblioteca *scikit learn*[31], de modo que 80% de cada grupo de amostras sejam direcionados para treino, enquanto o restante seja direcionado para teste. É importante mencionar que, para garantir a reprodutibilidade dos resultados, o argumento *random_state* é definido como um inteiro. Assim, dado o mesmo conjunto X e y, os grupos de treino e teste serão os mesmos.

Tendo os grupos de treino e teste definidos, obtém-se, portanto, *X_treino, y_treino, X_teste* e *y_teste*. Logo, y_treino e y_teste são convertidos para arranjos de uma dimensão para adequar-se aos requisitos das funções que os utilizam. Por fim, o modelo é treinado parcialmente com X_treino e y_treino através do atributo *partial_fit()* da biblioteca *scikit learn*[32], o que será discutido adiante na próxima seção.

3.6 TREINO E TESTE DO MODELO DE *MACHINE LEARNING*

Para realizar classificação de imagens, muitos algoritmos de *machine learning* podem ser utilizados. Esses algoritmos, para o caso de classificação de imagens, são chamados de classificadores (CHUGH et al., 2020).

Nesse sentido, após testes com diferentes classificadores como *RandomForestClassifier()*[33] e *AdaBoostClassifier()*[34], ambos da biblioteca *scikit learn*, verificou-se que o algoritmo mais apropriado para este caso é o *SGDClassifier()*[35], também da biblioteca *scikit learn*. Isso se deve não só por ter se mostrado efetivo no treinamento de modelos linerares (Shalev-Shwartz; Ben-David[36], 2014 *apud*

31 Mais informações sobre a função em: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.
32 Mais informações sobre o atributo em: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier.partial_fit.
33 Mais informações sobre o classificador em: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.
34 Mais informações sobre o classificador em: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html.
35 Mais informações sobre o classificador em: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html.
36 SHALEV-SHWARTZ, S.,BEN-DAVID, S. Understanding machine learning: From theory to algorithms, Cambridge University Press, 2014.

CATAPANG, J. K., Optimizing Speed and Accuracy Trade-off in Machine Learning Models via Stochastic Gradient Descent Approximation, 2022), mas também por oferecer a possibilidade de realizar treinos parciais por meio do atributo *partial_fit()*. Essa característica do algoritmo é ideal para o caso em questão, uma vez que a quantidade de amostras é bastante elevada (82.944.000) e existem limitações de hardware (20 GB de RAM e processador com quatro núcleos). Assim, a possibilidade de treinamentos parciais foi decisiva na escolha deste algoritmo, uma vez que devido à quantidade de amostras, quando treinadas em uma única vez com os outros classificadores, havia sobrecarga de RAM, e o código era encerrado antes do fim.

3.6.1 Algoritmo *Stochastic Gradient Descent* (SGD)

De acordo com a página da biblioteca *scikit learn* (2023),

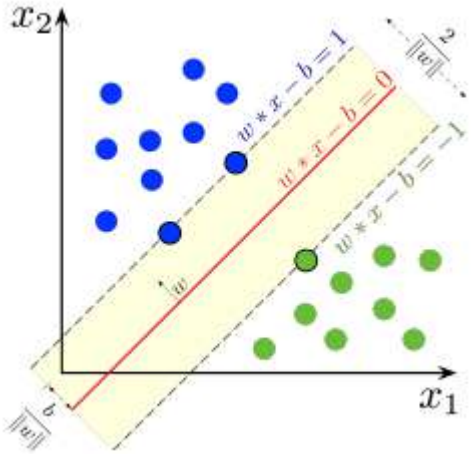
> [O algoritmo SGD] implementa modelos lineares regularizados com aprendizado de descida gradiente estocástica: o gradiente de perda é estimado a cada [iteração] e o modelo é atualizado ao longo do processo por meio de um cronograma de força decrescente (conhecido como “taxa de aprendizado”).

No caso deste trabalho, o modelo linear regularizado usado para treinar o modelo de *machine learning* é um SVM (*Support Vector Machine*). Um SVM é um algoritmo supervisionado de ML que implementa fronteiras de decisão (ou um hiperplano multidimensional) de modo a designar dados com n dimensões (neste caso as amostras com 57 atributos) em classes, permitindo que novos dados sejam facilmente classificados na classe correta (SAHU; SHARMA, 2023).

Considerando os dados de treino (X_treino) com 57 atributos cada e seus respectivos dados de referência (y_treino), o hiperplano neste caso deve ter n-1 dimensões, onde n é a quantidade de atributos/características que definem os dados de treino. Assim, o hiperplano neste caso teria 56 dimensões, algo impossível de ser representado graficamente. Por conta disso, para facilitar a explicação, a Figura 15

apresenta o caso de um hiperplano com uma dimensão (linha vermelha) dado um conjunto de dados com duas dimensões (atributos), representados por X_1 e X_2, do qual os dados de referência podem ter valores +1 ou -1, indicando as classes às quais os dados a serem treinados pertencem.

Figura 15 - Classificação de dados com dois atributos mediante o hiperplano e suas margens.



Fonte: SAHU, Chandan Kumar; SHAMAR, Maitrey. HINGE LOSS IN SUPPORT VECTOR MACHINES, School of Computer Sciences, National Institute of Science Education and Research, Bhubaneshwar, Homi Bhabha National Institute, 2023. Disponível em: https://www.niser.ac.in/~smishra/teach/cs460/23cs460/lectures/lec11.pdf. Acesso em: 28 nov. 2023.

É possível verificar que as margens do hiperplano são definidas por outros dois hiperplanos paralelos ao central. Os hiperplanos são definidos como segue:

$$Hiperplano\,Central : \hat{w^T} \cdot \hat{x} - b = 0$$

$$Hiperplano\,Margem\,Superior : \hat{w^T} \cdot \hat{x} - b = 1$$

$$Hiperplano\,Magem\,Inferior : \hat{w^T} \cdot \hat{x} - b = -1$$

Das expressões matemáticas dos hiperplanos, w^T é o vetor transposta dos pesos do classificador SVM, x é o vetor de todas as amostras e b é valor de viés do classificador. Dessa forma, todas as amostras acima da margem superior pertencem a uma classe e, similarmente, todas as amostras abaixo da margem inferior pertencem a outra classe. Dessa forma, a intenção do classificador é maximizar o

comprimento da margem para melhor definir as regiões de cada classe e, para evitar que amostras encontrem-se dentro dela, estipula-se um limitador como segue:

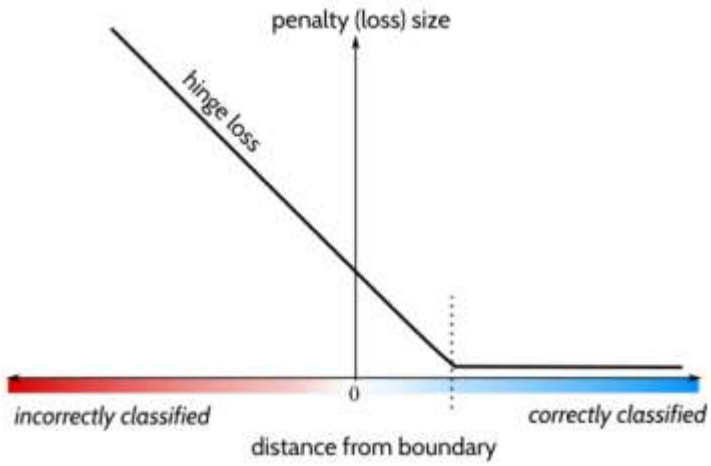
$$y_i(\hat{w^T} \cdot \hat{x}_i - b) \geq 1$$

Nessa expressão, y_i é o dado de referência de cada amostra, x_i é o vetor de cada amostra e o termo entre parênteses é o valor associado à previsão do modelo.

Quando existem dados que se encontram dentro da margem, o método utilizado é chamado de *soft-margin*, que permite “sacrificar” algumas das amostras quanto à classificações incorretas.

Dessa forma, para definir a *soft-margin*, utiliza-se da função de perda. Neste caso, a função de perda é definida como *hinge*, passada pelo parâmetro *loss* do *SGDClassifier()*. A função de perda calcula o erro entre o valor previsto para uma classe e o valor real daquela classe para cada amostra. Assim, para otimizar o sistema, é necessário minimizar esse erro. A Figura 16 mostra a função de perda *hinge.*

Figura 16 - Função de perda *hinge*.



Fonte: SAHU, Chandan Kumar; SHAMAR, Maitrey. HINGE LOSS IN SUPPORT VECTOR MACHINES, School of Computer Sciences, National Institute of Science Education and Research, Bhubaneshwar, Homi Bhabha National Institute, 2023. Disponível em: https://www.niser.ac.in/~smishra/teach/cs460/23cs460/lectures/lec11.pdf. Acesso em: 28 nov. 2023.

Ainda, a função de perda *hinge* é representada matematicamente pela expressão abaixo:

$$L(y) = max(0, 1 - y_i(\hat{w^T} \cdot \hat{x}_i - b))$$

Assim, quando a multiplicação do valor real (y_i) e do valor previsto (termo entre parênteses) é maior ou igual a um, significa que a amostra x_i foi classificada de maneira correta e a penalidade para essa amostra é mínima ($L(y) = 0$). Por outro lado, caso $L(y) > 0$, significa que a amostra foi classificada em uma das outras classes e que a penalidade terá valor proporcional a quão longe da classificação correta a amostra se encontra.

Com isso, o algoritmo de Gradiente Descendente Estocástico tenta otimizar o processo de classificação por meio da correção do erro da função de perda nas iterações seguintes com novas amostras. Isso acontece através do cálculo dos novos pesos a partir da derivada parcial da função em relação aos pesos. As expressões do novo peso e da derivada parcial são mostradas a seguir:

$$w_{t+1} = w_t - \eta \frac{\partial L}{\partial w_t}$$

$$\frac{\partial L}{\partial w_t} = \nabla_w L(w_t; x_i; y_i)$$

Assim, w_{t+1} indica os pesos da iteração seguinte *t+1,* η é o valor da taxa de aprendizado (*learning rate*), e a derivada parcial da função de perda em relação aos pesos é o gradiente da função de perda em relação aos pesos anteriores w_t, da amostra x_i e de seu valor de referência y_i (KANDEL; CASTELLI; POPOVIČ, 2020). Além disso, a escolha da amostra x_i e sua respectiva referência y_i para corrigir os pesos é aleatória.

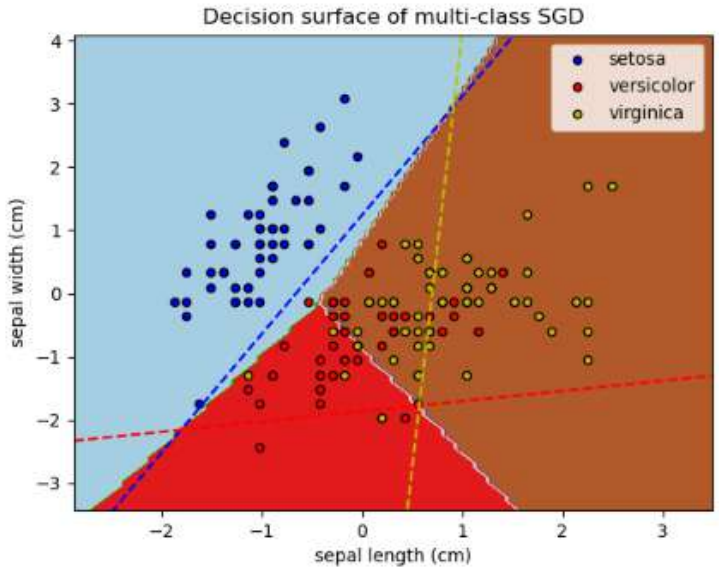
Por conseguinte, a partir das correções de pesos e assim da classificação das novas amostras, o classificador SVM consegue aumentar a margem do hiperplano e definir com maior precisão a separação das classes.

No caso de um problema com três classes como o em questão, o algoritmo SGD utiliza do método *one versus all* (um contra todos), de modo que para cada classe existe um classificador SVM binário que qualifica a amostra entre pertencente

a essa classe ou às outras. Por consequência, cada classificador gera um hiperplano que delimita a classe considerada verdadeira (a classe de seu classificador) das outras. Assim, durante o treino de classificação, todos os três classificadores são acionados, e aquele que tem o maior grau de confiança determina a qual classe pertence a amostra.

Desse modo, a cada imagem, todo o processo de posicionamento dos hiperplanos e margens, e classificação de amostras é repetido. Como já discutido, no caso deste trabalho existem 57 atributos por arranjo de pixel. Assim, a demonstração visual do hiperplano seria complexa. Por isso, abaixo há um exemplo retirado da página da biblioteca *scikit learn* no qual são classificadas amostras de flores entre três espécies (classes) de acordo com dois atributos: comprimento e espessura de sépalas.

Figura 17 – Exemplo de hiperplano para classificação de imagens.



Fonte: 1.5. Stochastic Gradient Descent. scikit learn, 2023. Disponível em: https://scikit-learn.org/stable/modules/sgd.html. Acesso em: 20 nov. 2023.

Na Figura 17, cada linha tracejada representa um hiperplano de um classificador binário do método “um contra todos”, enquanto no plano de fundo

encontram-se as regiões coloridas que representam as áreas de decisão induzidas pelos hiperplanos.

3.6.2 Parâmetros do Algoritmo

O algoritmo SGD possui muitos parâmetros a serem definidos para melhorar seu desempenho. Nesse sentido, após testes com diferentes valores, os mais adequados são os apresentados em seguida.

- *loss*: como já discutido, a função de perda definida pelo parâmetro *loss* foi escolhida como *hinge.*
- *shuffle*: configurado como *True*, esse parâmetro garante que as amostras de treino sejam embaralhadas a cada imagem, de modo que diferentes índices relativos às amostras estejam no grupo de treino seguinte.
- *random_state*: também já discutido, garante a reprodutibilidade dos resultados todas as vezes em que o código é executado.
- *warm_start*: configurado como *True*, reutiliza a solução do último *partial_fit* como inicialização para o próximo.
- *learning_rate*: configurado como *adaptive*, mantém a taxa de aprendizado inicial constante, desde que a perda continue a diminuir.
- *eta0*: definido como *0.01*, indica a taxa de aprendizado inicial.
- *class_weight*: definido como *balanced*, retorna o peso associado a cada classe, de modo a ajustá-los inversamente proporcional à frequência das classes nos dados de referência. Esse parâmetro é redefinido a cada imagem, contribuindo com o cálculo dos novos pesos dos classificadores SVM.
- *average*: definido como *True*, computa a média dos pesos de descida de gradiente estocástica ao longo de todas as atualizações de peso, e armazena o resultado.

3.6.3 Treino do Modelo

Definido o classificador e o algoritmo SGD, o modelo deve ser treinado. Como exposto anteriormente, o treinamento é feito em partes através do atributo *partial_fit()*. Este recebe como parâmetros os dados de treino de X e y (X_treino e y_treino), além das três classes (29 – azul - VRA, 76 – vermelho - SV, 150 – verde - VF).

O processo de treinamento se repete até que todas as fotos tenham sido analisadas.

3.6.4 Teste do Modelo

Após cada ciclo de treinamento, é obtida a exatidão do modelo através do atributo *score()*[37], que analisa todas as amostras de teste (X_teste e y_teste) reunidas até o momento. Além disso, a precisão, o *recall* e o F*1-Score* também são obtidos, mas através da função *classification_report()*[38]. Ao fim de todas as classificações, a matriz de confusão é devolvida através da função *confusion_matrix*[39].

Em seguida, são apresentados os conceitos de matriz de confusão, exatidão, precisão, *recall* e F*1-Score*.

37 Mais informações sobre o atributo em: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier.score.

38 Mais informações sobre a função em: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html.

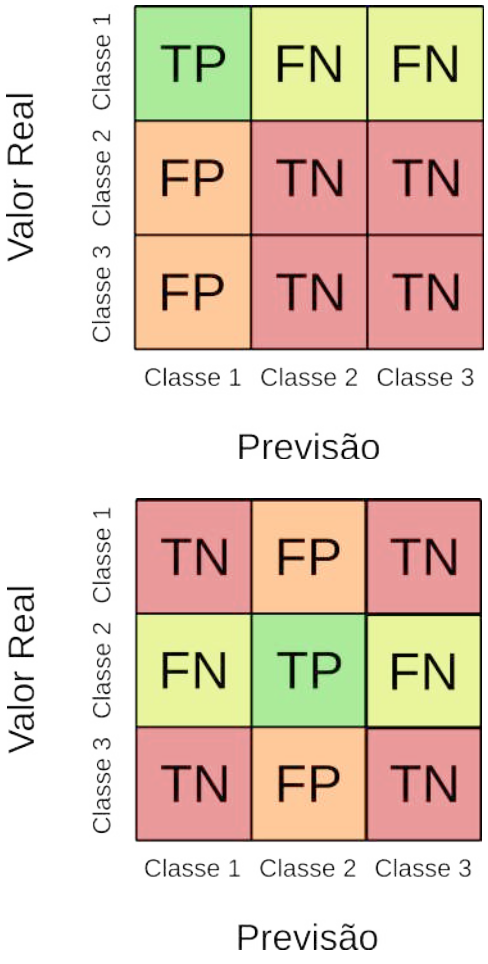
39 Mais informações sobre a função em: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html.

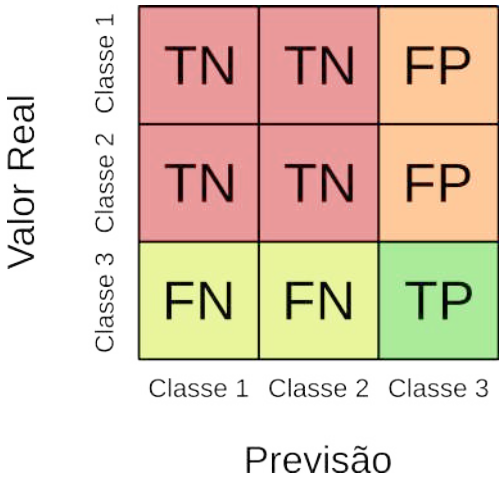
3.6.4.1 Matriz de Confusão

A matriz de confusão é um método bastante utilizado para avaliar problemas de classificação de imagem (KULKARNI; CHONG; BATARSEH, 2020). Ela representa a contagem de valores previstos e reais. Através dela, consegue-se calcular a exatidão, precisão, *recall* e *F1-Score*.

Considerando um problema de três classes como o em questão, a matriz será de tamanho 3x3. A seguir, são ilustradas a mesma matriz de confusão, porém em três imagens e configurações diferentes, de modo a facilitar a apresentação dos cálculos para cada classe.

Figura 18 – Matriz de confusão considerando cada uma das três classes.





Fonte: Compilação do autor.

Das figuras apresentadas, *TP* significa *True Positive* e indica a quantidade de amostras previstas como dentro da classe e que de fato pertencem a ela. Por outro lado, *FP*, que significa *False Positive*, indica as amostras previstas como pertencentes a determinada classe, mas que na verdade não são. Além disso, *FN* (*False Negative*) indica as amostras que não foram classificadas em uma classe específica, mas que na verdade pertencem a ela. Por fim, *TN*, *True Negative*, indica as amostras que não foram classificadas em uma classe específica e que de fato não pertencem a ela. As classes 1, 2 e 3 representam, respectivamente, *VF*, VRA e SV (Vegetação de Floresta, Vegetação Rasteira e Arbustos, e Sem Vegetação, como já apresentado).

3.6.4.2 Exatidão

Exatidão é uma métrica utilizada para avaliar modelos de classificação de imagem. Ela define a porcentagem de amostras previstas com exatidão em relação a todas as previsões (GOOGLE MACHINE LEARNING, 2023). A exatidão é calculada para todas as classes, de modo a classificar o modelo como um todo. A seguir apresenta-se a fórmula para calculá-la.

$$Acurácia = \frac{[(TP_{VF} + TP_{VRA} + TP_{SV}) + (TN_{VF} + TN_{VRA} + TN_{SV})]}{[(TP_{VF} + TP_{VRA} + TP_{SV}) + (TN_{VF} + TN_{VRA} + TN_{SV}) + (FP_{VF} + FP_{VRA} + FP_{SV}) + (FN_{VF} + FN_{VRA} + FN_{SV})]}$$

3.6.4.3 Precisão

De acordo com Kulkarni, Chong e Batarseh (2020), precisão evidencia “o quão exato é o modelo quanto a predizer valores positivos”. Assim, precisão pode ser entendida como a exatidão das previsões tidas como positivas (Bruce, P.; Bruce, A[40], 2017 *apud* KULKARNI, Ajay; CHONG, Deri; BATARSEH, Feras A. Data Democracy, At the Nexus of Artificial Intelligence, Software Development, and Knowledge Engineering, 2020). Desse modo, precisão é expressa como segue.

$$Precisão = \frac{(TP_{VF} + TP_{VRA} + TP_{SV})}{[(TP_{VF} + TP_{VRA} + TP_{SV}) + (FP_{VF} + FP_{VRA} + FP_{SV})]}$$

3.6.4.4 *Recall*

Recall mede a força de um modelo prever resultados positivos, sendo conhecido como a sensibilidade do modelo (KULKARNI; CHONG; BATARSEH, 2020). Em outras palavras, *recall* indica a proporção de previsões positivas em relação a todos os resultados realmente positivos. A expressão a seguir define matematicamente *recall*.

$$Recall = \frac{(TP_{VF} + TP_{VRA} + TP_{SV})}{[(TP_{VF} + TP_{VRA} + TP_{SV}) + (FN_{VF} + FN_{VRA} + FN_{SV})]}$$

3.6.4.5 *F1-Score*

F1-Score é outra medida para avaliar um modelo de classificação de imagem. Nesse caso, ele reúne a precisão e o *recall* balanceados em uma média harmônica. Assim, para a classificação de valores positivos, contribui para entender o equilíbrio

40 BRUCE, P., BRUCE, A. Practical Statisctics for Data Scientists: 50 Essential Concepts, O'Reilly Media, 2017.

entre correção e cobertura de amostras do modelo (Alberto et al.[41], 2018 *apud* KULKARNI, Ajay; CHONG, Deri; BATARSEH, Feras A. Data Democracy, At the Nexus of Artificial Intelligence, Software Development, and Knowledge Engineering, 2020).

A medida pode ser expressa da seguinte forma, de modo que β seja igual a um.

$$F_{\beta}=(1+\beta^{2})\frac{Precisão\cdot Recall}{(\beta^{2}\cdot Precisão)+Recall}\Rightarrow F_{1}=2\frac{Precisão\cdot Recall}{Precisão+Recall}$$

41 ALBERTO, F. et al. Learning from Imbalanced Data Sets, Springer, 2018.

4 RESULTADOS

A seguir, são mostrados os resultados da performance do modelo de classificação de imagem discutido nas seções anteriores. Além disso, cada um deles será discutido nesta mesma seção.

4.1 EXECUÇÃO DO CÓDIGO

Embora o número de fotos a serem utilizadas para treinar o modelo era de 160, devido a problemas de capacidade do hardware foi necessário reduzi-lo. Assim, o novo número de fotos treinadas foi 120, tendo, portanto, 62.208.000 pixeis como amostras (49.766.400 para o treino e 12.441.600 para o teste do modelo).

Ainda, foi necessário utilizar outro computador com mais capacidade que o anterior (20 GB de RAM; quatro núcleos no processador), tendo o novo 32 GB de RAM e doze núcleos no processador. No entanto, a quantidade de RAM ainda era insuficiente para as 160 fotos, o que levou à redução a 120 imagens, como discutido, uma vez que a performance do computador atingia seu máximo e o código colapsava.

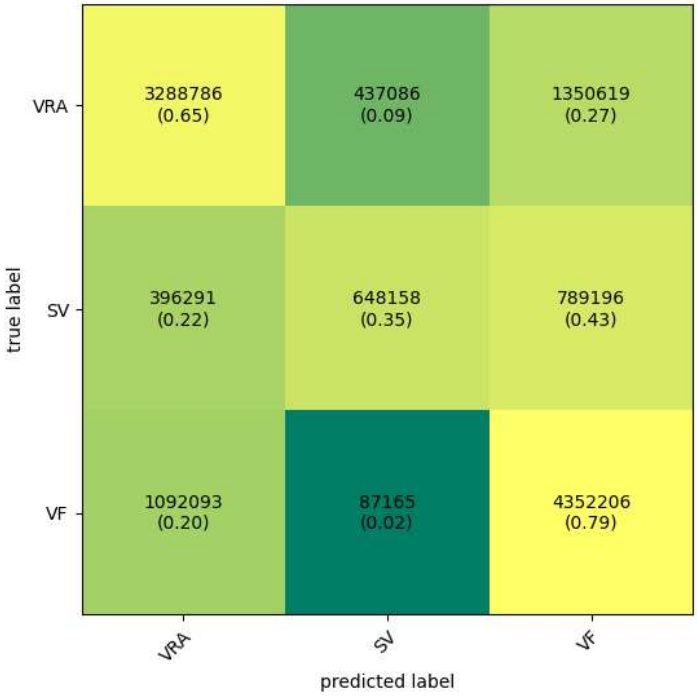
Após a redução, o código conseguiu ser concluído, e os resultados de performance do modelo, assim como ele próprio, obtidos. O código completo para gerar o modelo e seus resultados encontra-se no Apêndice A.

4.2 MATRIZ DE CONFUSÃO DO MODELO

A matriz de confusão expõe a quantidade de amostras classificadas de acordo com a previsão do modelo e também com a classificação real. A partir dela, é possível obter indicadores como exatidão, precisão, *recall* e o *F1-Score*, como

explicado anteriormente. A seguir, a matriz de confusão após todo o treinamento do modelo é apresentada.

Figura 19 - Matriz de confusão do modelo.



Fonte: Compilação do autor.

A partir da Figura 19 e do exposto na seção 3.6.4.1, é possível verificar que Vegetação de Floresta (VF) teve a maior proporção de acertos quanto aos valores reais (79%). Por outro lado, Vegetação Rasteira e Arbustos (VRA) e Sem Vegetação (SV) tiveram previsões corretas baixas (65% e 35%, respectivamente).

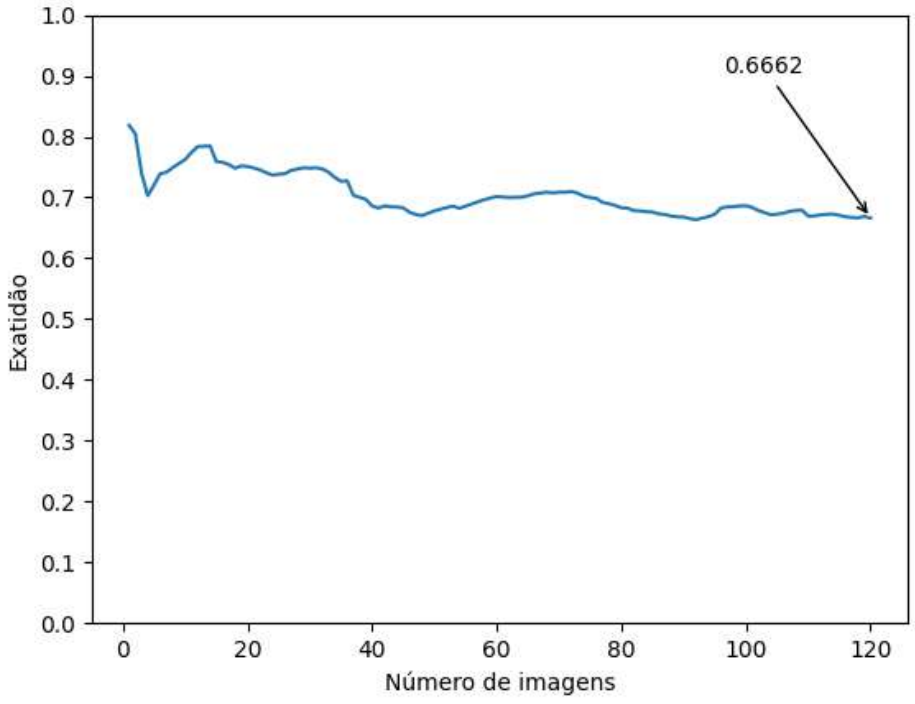
Além disso, quanto à SV, a maior parte das previsões indicaram incorretamente a classe VF, e parte considerável em VRA. Isso pode ter acontecido devido a grande abrangência de intensidades de cor e brilho dos pixeis que de fato pertencem a essa classe. Um bom exemplo é a diferença entre uma estrada de terra (comum nas fotos), com intensidade de brilho alta, e a sombra de uma edificação. Nesse sentido, como as florestas tendem a ser escuras, a sombra seria classificada como VF e não SV.

Ainda, boa parte das previsões de VRA que na verdade eram VF (20%) e de VF que eram na verdade VRA (27%) foram classificadas incorretamente. Isso pode ter acontecido pela semelhança de intensidade e brilho das duas classes, uma vez que a grama, se estiver mais escura na foto, pode ser classificada como VF, e se a floresta estiver mais clara, pode ser confundida com VRA.

4.3 EXATIDÃO DO MODELO

Como discutido na seção 3.6.4.2, a exatidão evidencia a quantidade de previsões corretas em relação a todas realizadas. Em seguida, apresenta-se o gráfico da Figura 20, que mostra os valores de exatidão a cada iteração de treino do modelo, com o acumulado dos dados de teste até o momento, relativo à quantidade de fotos utilizadas até então.

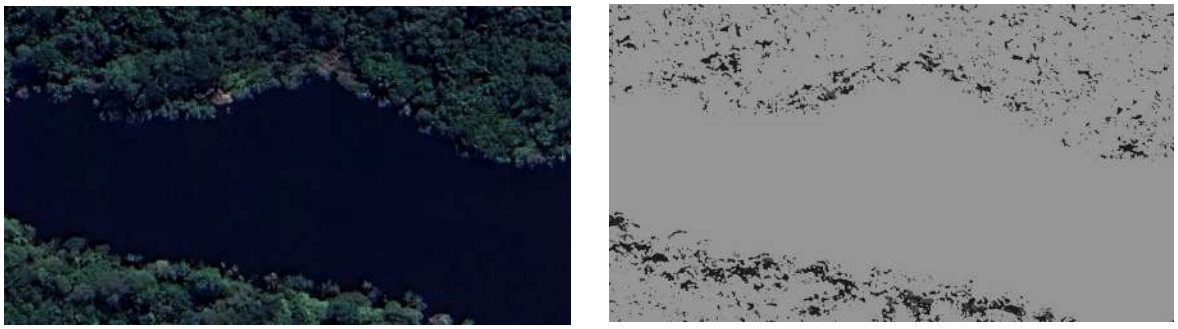
Figura 20 - Exatidão x Acumulado de dados de teste por imagens treinadas.



Fonte: Compilação do autor.

Como é possível perceber, de forma geral, a exatidão com até 35 fotos permanece acima de 70%, com uma queda contínua até 40 imagens, atingindo algo entre 70% e 67%. Esse comportamento pode ter sido provocado pela sequência dessas cinco imagens que contêm ou apenas uma classe, ou duas. Além disso, nas imagens com duas classes, existem regiões de floresta e rio e, como a coloração do rio é escura, é possível que o modelo tenha confundido com a coloração da floresta. A Figura 21 apresenta uma dessas cinco fotos e sua classificação diante do modelo.

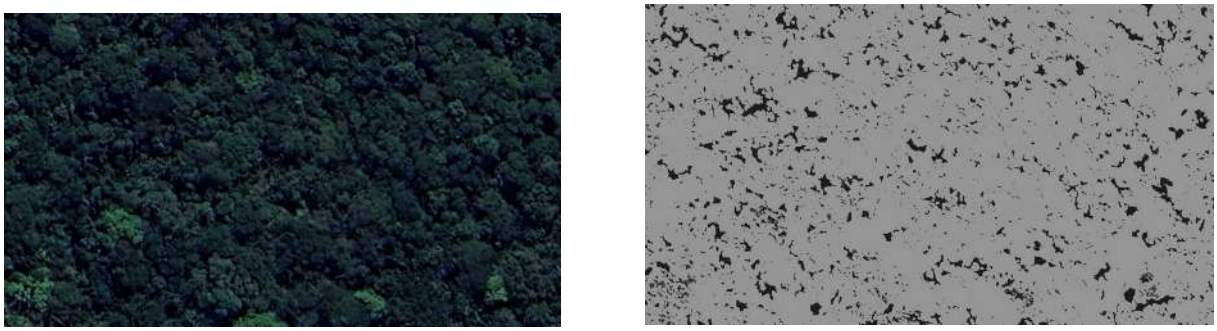
Figura 21 – Rio e floresta são confundidos na versão classificada pelo modelo.



Fonte: Google Earth (2023); Compilação do autor.

Ademais, esse comportamento de queda também acontece entre 73 e 96 imagens, intervalo com muitas fotos com apenas uma ou duas classes. A Figura 22 mostra um exemplo com apenas uma classe a ser definida (Vegetação de Floresta) e que possui pixeis de outras classes.

Figura 22 – Imagem com apenas floresta sendo classificada pelo modelo com outras classes.



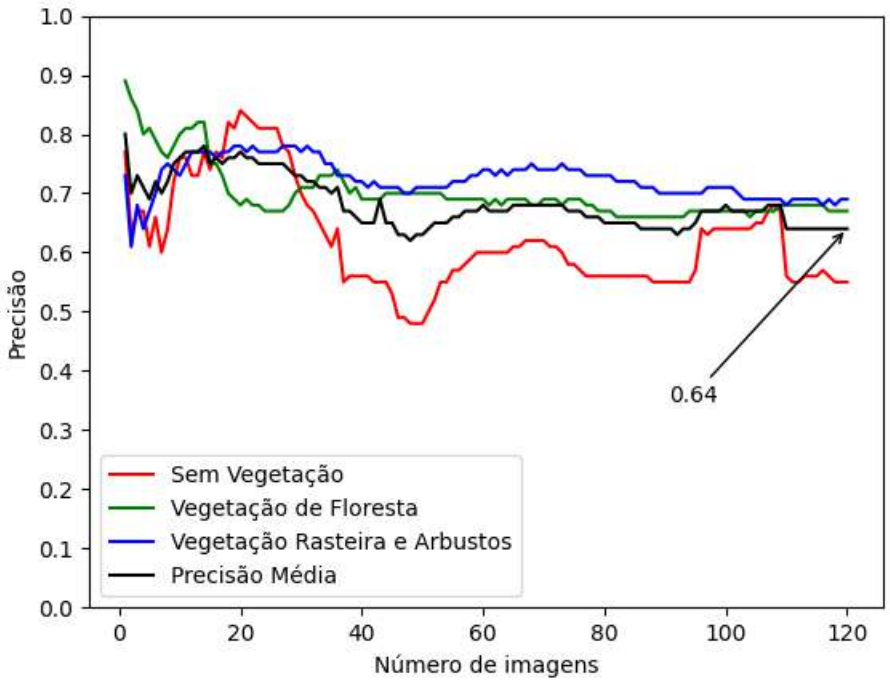
Fonte: Google Earth (2023); Compilação do autor.

Assim, é possível inferir que o modelo não seja tão exato na previsão de imagens que contenham menos de três classes, além de ser bastante sensível à intensidade das cores presentes nas imagens originais, como visto no exemplo de foto da Figura 21. Desse modo, a exatidão média final convergiu para 66,62%, valor que não infere tanta confiança no modelo, mas que indica que a classificação correta ocorre para a maior parte dos pixeis de uma imagem.

4.4 PRECISÃO DO MODELO

Em resumo, a precisão indica a proporção de previsões positivas corretas para uma classe em relação a todas as previsões positivas, corretas e incorretas. A Figura 23 mostra as curvas de precisão para cada uma das três classes, além da previsão média de todas elas em conjunto.

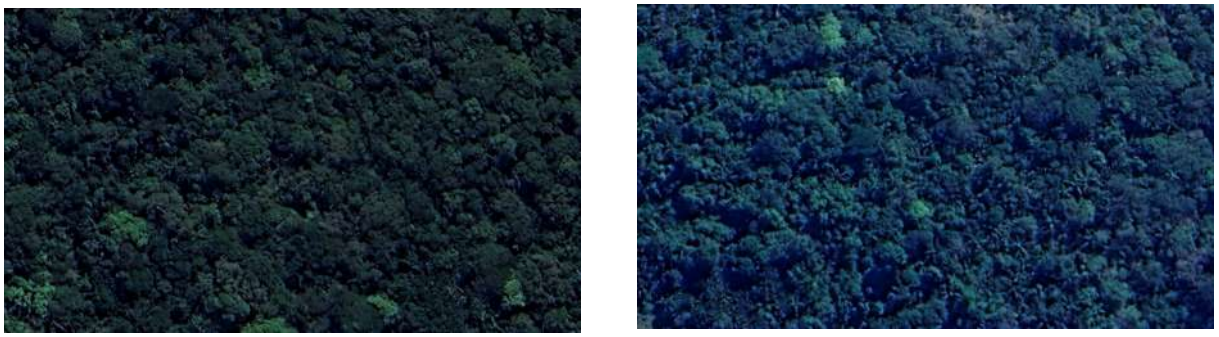
Figura 23 - Precisão x Acumulado de dados de teste por imagens treinadas.



Fonte: Compilação do autor.

A partir do gráfico, é possível perceber que apesar do início, VF e VRA se mantiveram praticamente constantes, com amplitude baixa, e terminaram próximas de 70%. Isso indica que das previsões positivas para essas classes em relação a todas as previsões positivas para as mesmas classes, ainda que incorretas, o modelo teve boa performance. No entanto, a curva de SV teve comportamento bastante oscilatório, com alta amplitude. Embora tenha alcançado um patamar próximo de 68% entre 95 e 110 fotos, terminou em um patamar muito mais baixo. Isso pode ter acontecido devido à presença de fotos a partir da 110ª com claridade acima do comum, possivelmente devido a uma nuvem ou até mesmo falha do equipamento que fotografou as imagens. Abaixo existem dois exemplos de fotos de floresta usados no treinamento com claridades diferentes.

Figura 24 - Exemplo de imagens diferentes de floresta, com claridades diferentes.



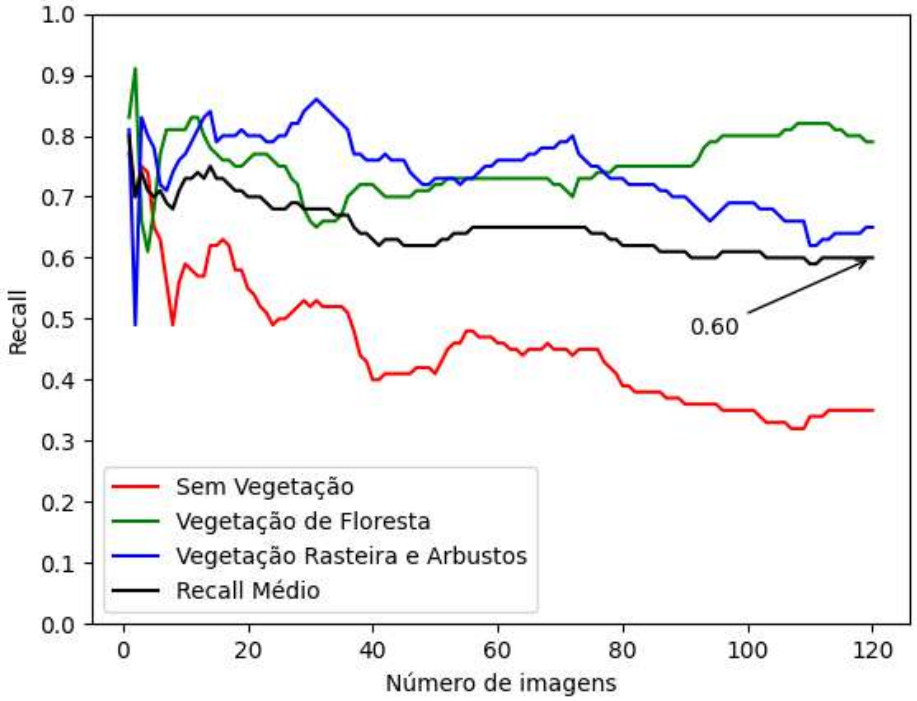
Fonte: Google Earth (2023).

Por fim, devido à queda da curva de SV, a precisão média também sofreu o impacto, terminando em 64%.

4.5 *RECALL* DO MODELO

O *recall*, como já definido, representa a proporção das previsões positivas para uma classe em relação a todas classificações verdadeiras para a mesma classe. A Figura 25 mostra os valores de *recall* para cada classe, assim como a média de todas elas.

Figura 25 - *Recall* x Acumulado de dados de teste por imagens treinadas.



Fonte: Compilação do autor.

Pode-se visualizar que a curva de VF teve comportamento inverso da de VRA: enquanto a primeira subiu, alcançando um patamar próximo dos 80%, a outra caiu de 80% até 66%. É possível que esse comportamento tenha ficado em evidência devido a semelhança nas tonalidades das duas classes, como já discutido.

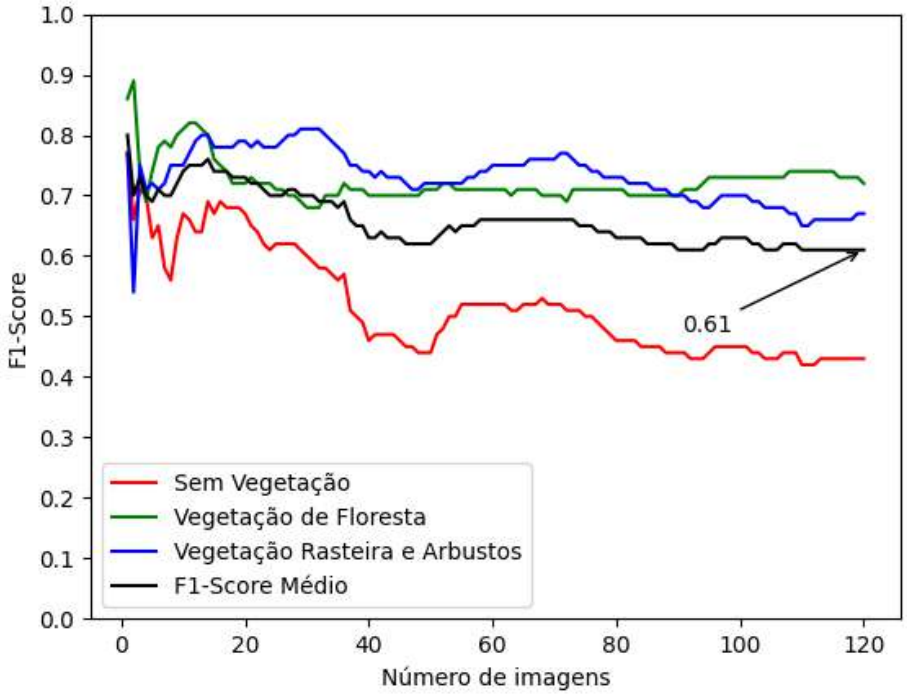
Além disso, a classe SV teve comportamento bastante inferior ao esperado: desde o início teve queda quase que constante até convergir para algo próximo de 35%. Assim, comprova-se que a previsão para essa classe teve muitas falhas, como exposto na seção 4.2, quando dito que as classes VF e VRA obtiveram grande parte das previsões que deveriam ter sido alocadas em SV. O motivo para tal pode ter sido a grande variedade de intensidade dos pixeis pertencentes a essa classe, como já relatado.

Ademais, o *recall* médio manteve-se praticamente constante apesar da grande queda da curva SV. Isso se deve à compensação tanto de VF quanto de VRA.

4.6 *F1-SCORE* DO MODELO

O *F1-Score* indica a média harmônica entre a previsão e o *recall*. Nesse sentido, pode-se entender melhor o balanço dessas duas medidas, equilibradas. A Figura 26 mostra o gráfico de *F1-Score* para cada uma das classes e a média entre elas.

Figura 26 - *F1-Score* x Acumulado de dados de teste por imagens treinadas.



Fonte: Compilação do autor.

É possível perceber que todas as quatro curvas possuem amplitudes mais amenas, permanecendo de certa maneira constantes. Além disso, como esperado, a curva de SV permaneceu abaixo de 50%. Por outro lado, as curvas de VF e VRA tiveram comportamento mais equilibrado e acima de 70%. O *F1-Score* médio convergiu em 61%.

5 APLICAÇÃO

Após o treino e a obtenção dos resultados do modelo, ocorre sua implementação em um hardware embarcado. Um hardware embarcado consiste em um conjunto de componentes computacionais e eletrônicos integrados em um chip, como memórias, GPU, CODECs e interfaces de conectividade, interconectados através de barramentos com alta taxa de transferência, formando um *System-on-Chip* – SoC (BARTÍK; PICHLOVÁ; KUBÁTOVÁ, 2016), associado a uma placa base. A placa base é responsável por receber o SoC como seu controlador, de modo a oferecer os receptores de entrada e saída (E/S) dos periféricos, como conectores de cabo de rede, USB, áudio, microfone, além dos GPIOs (*General-Purpose Input/Output* – Entradas/Saídas de Uso Geral) para uso de diversos protocolos de comunicação, como CAN, I2S, PWM e mais.

Assim, para que o usuário final utilize o hardware embarcado com mais facilidade, implementando uma aplicação em seu módulo (SoC), é necessário um software embarcado. O software embarcado nada mais é que o sistema operacional do hardware embarcado, de modo a servir de intermediário na comunicação entre os componentes do hardware e a aplicação do usuário final. Dessa forma, o usuário não precisa se preocupar com conceitos de programação de baixo nível para conseguir se comunicar diretamente com o hardware.

Com isso, quando há um hardware embarcado com um software embarcado associado, tem-se um sistema embarcado. Nesse sentido, o modelo de ML criado é implementado em um sistema embarcado, como mostrado nas seções a seguir.

5.1 VISÃO GERAL DO SISTEMA EMBARCADO UTILIZADO

O hardware utilizado foi um SoC Apalis iMX8 Quad Max, com 4 GB de RAM, com disponibilidade de Wi-Fi e Bluetooth e operacionalidade em temperatura industrial, versão V1.1 C. Associado a ele, uma placa base Ixora, versão V1.1 A,

com diversas E/S para periféricos. Ambos os componentes foram produzidos pela empresa Toradex.

5.1.1 Apalis iMX8QM

De acordo com a página da Toradex (2023-a), o módulo Apalis iMX8 Quad Max possui dois núcleos Arm Cortex A-72, além de quatro núcleos Arm Cortex A-53. Ademais, possui dois núcleos de microcontroladores Arm Cortex M4F com FPU. O módulo possui alta performance para aplicações de visão computacional devido a sua dual GPU GC7000. Ainda, possui memória *flash* de 32 GB eMMC, além dos já mencionados 4 GB de RAM LPDDR4 (64 Bit). A Figura 27 ilustra o módulo Apalis iMX8QM.

Figura 27 - Módulo Apalis iMX8QM 4GB WB IT.

Fonte: Toradex (2023-a). Disponível em: https://www.toradex.com/pt-br/computer-on-modules/apalis-arm-family/nxp-imx-8. Acesso em: 29 nov.2023.

5.1.2 Placa base Ixora

A placa base Ixora tem fator de forma pequeno e otimizado, possuindo suporte para diversas interfaces industriais (TORADEX, 2023-b). Seu tamanho é ideal para projetos em que o espaço é limitado, como o acoplamento em um *drone*. Além disso, a placa suporta interfaces de alta velocidade e multimídias. Abaixo na Figura 28 há uma foto da placa base Ixora.

Figura 28 - Placa base Ixora.

Fonte: Toradex (2023-b). Disponível em: https://www.toradex.com/pt-br/products/carrier-board/ixora-carrier-board. Acesso em: 30 nov. 2023.

5.1.3 Torizon OS

Segundo a página da Toradex (2023-c), Torizon OS é:

> […] uma imagem mínima de Linux embarcado que apresenta, dentre outros serviços essenciais, processamento em contêiner e componentes para atualizações offline seguras e remotas (*Over-the-air – OTA*).

Nesse sentido, Torizon OS é o software embarcado, sendo o sistema operacional do hardware embarcado. Assim, havendo a integração com contêineres, o processo de produção de aplicações torna-se mais simples.

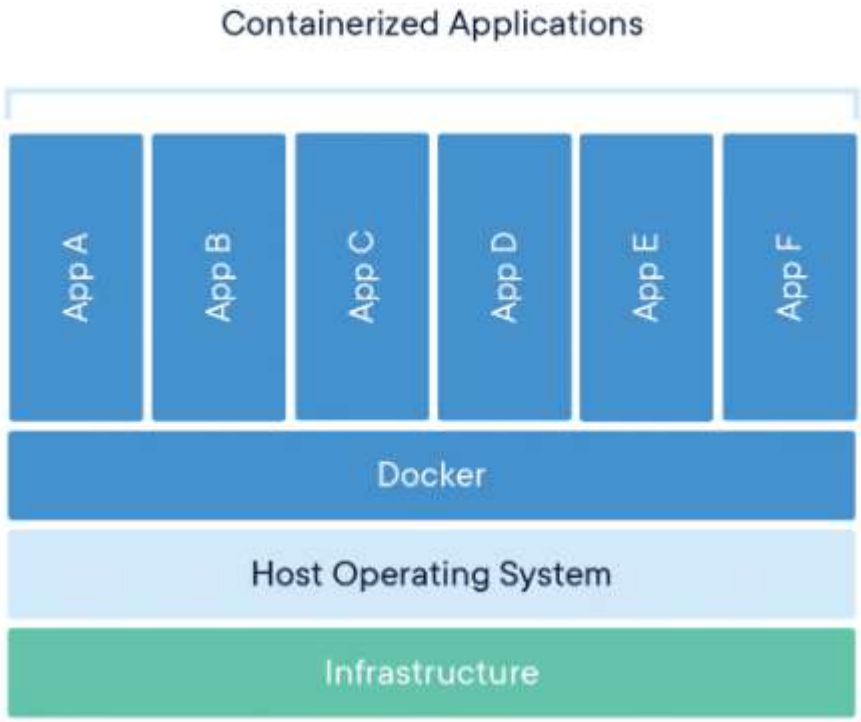
5.2 CONTEINERIZAÇÃO DA APLICAÇÃO

Uma das plataformas mais comuns de se trabalhar com contêineres é o Docker. De acordo com sua página (DOCKER, 2023), um contêiner é:

> [...] uma unidade padrão de software que empacota o código e todas as suas dependências, de modo que a aplicação seja executada rapida e seguramente de um ambiente computacional para outro. Uma imagem contêiner Docker é um pacote de software leve, autônomo e executável que inclui todo o necessário para executar uma aplicação: código, processamento, ferramentas do sistema, bibliotecas do sistema e configurações.

Assim, contêineres permitem criar o ambiente necessário para uma aplicação sem interferir diretamente no sistema operacional, como por exemplo, instalando bibliotecas diretamente nele. Com isso, utilizar contêineres se torna propício no caso deste trabalho, uma vez que a aplicação requer diversas configurações que não existem no sistema operacional base, como as bibliotecas necessárias para sua execução. Além disso, o uso de um contêiner também se faz útil a partir do momento em que é necessário converter a aplicação de uma arquitetura para outra (neste caso, da arquitetura x86 64 bit – arquitetura do computador - para arm64 – arquitetura do módulo embarcado). Abaixo há uma representação da estrutura geral do sistema embarcado utilizando contêiner, onde *Infrastructure* corresponde ao hardware embarcado (módulo) e *Host operating system* ao sistema operacional (Torizon OS).

Figura 29 - Estrutura geral de um sistema embarcado utilizando um contêiner Docker.



Fonte: Docker, 2023. Disponível em: https://www.docker.com/resources/what-container/. Acesso em: 30 nov. 2023.

Para criar um contêiner para a aplicação, é necessário utilizar um arquivo *Dockerfile*. Nele, existem todas as configurações necessárias para o funcionamento da aplicação, desde a instalação das ferramentas da linguagem de programação utilizada, de bibliotecas específicas para a aplicação, até o compilador cruzado para converter a aplicação de uma arquitetura para outra.

Tendo o *Dockerfile*, é necessário executar o comando de *build* e logo *push* para fazer o upload do contêiner no DockerHub. A seguir são mostrados os comandos executados em um terminal de um computador com Ubuntu 22.04.

```
cd <pasta_com_Dockerfile_e_aplicação>
docker build -t <login_DockerHub>/<nome_do_contêiner>:<tag_de_versão> .

docker login <login_DockerHub>

docker push <login_DockerHub>/<nome_do_contêiner>:<tag_de_versão>
```

5.3 RESULTADO DA APLICAÇÃO

Além do contêiner da aplicação, para evidenciar o resultado de forma visual, foi executado outro contêiner para receber a interface gráfica. Trata-se do Weston, que utiliza o protocolo Wayland[42]. Assim, foi utilizado um contêiner fornecido pela Toradex[43].

Com isso, para executar ambos os contêineres, deve-se utilizar o arquivo *docker*-compose. Portanto, é necessário copiar o arquivo para o módulo e depois, acessando seu terminal, executá-lo, como segue.

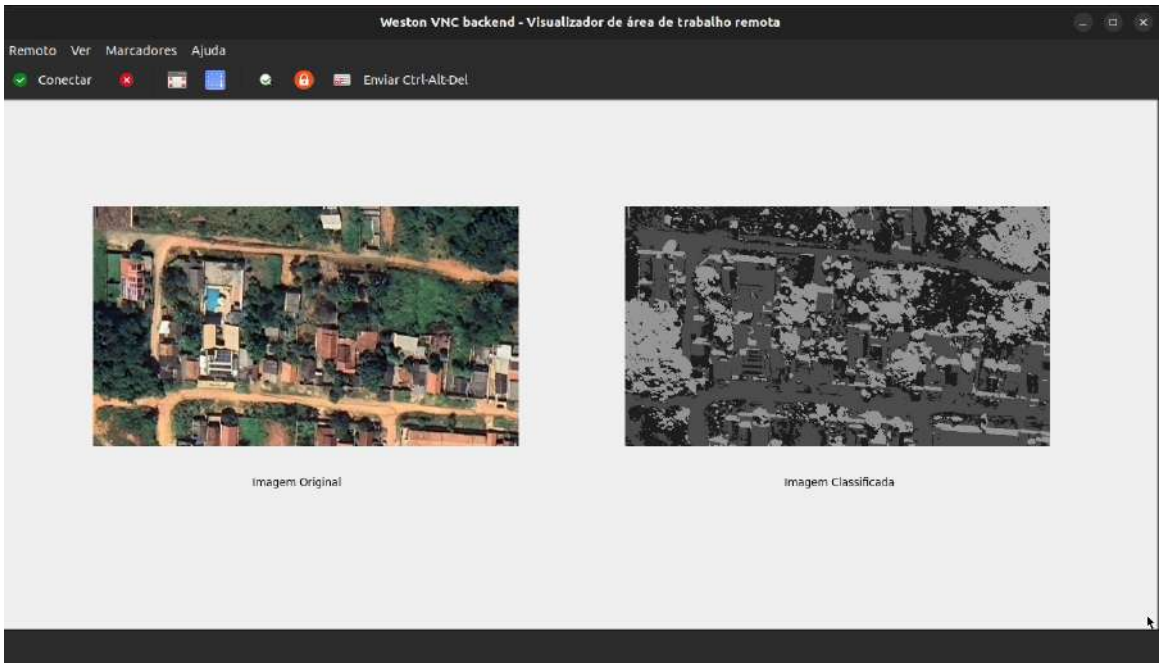
```
#No terminal do computador
cd <pasta_com_docker-compose>

scp docker-compose.yml torizon@<IP_do_módulo>:<pasta_de_destino>
```

```
#No terminal do módulo
docker-compose -f docker-compose.yml up
```

Assim, inserindo na aplicação uma interface gráfica através das ferramentas visuais do Qt[44], foi possível mostrar a foto original ao lado de sua imagem classificada, como a seguir. A aplicação é visualizada através de um visualizador VNC (*Virtual Networking Computing*).

42 Para mais informações, acesse: https://developer.toradex.com/torizon/provided-containers/working-with-weston-on-torizoncore/.
43 Mais informações sobre os contêineres disponíveis em: https://developer.toradex.com/torizon/provided-containers/debian-containers-for-torizon/#debian-containers.
44 Mais informações sobre o Qt com Python em: https://doc.qt.io/qtforpython-6/.

Figura 30 - Resultado visual da aplicação de classificação de imagem.



Fonte: Compilação do autor.

É importante mencionar que para executar a reprodução visual em um visualizador VNC pelo computador, foi necessário também conectar uma tela através do conector HDMI da placa base Ixora.

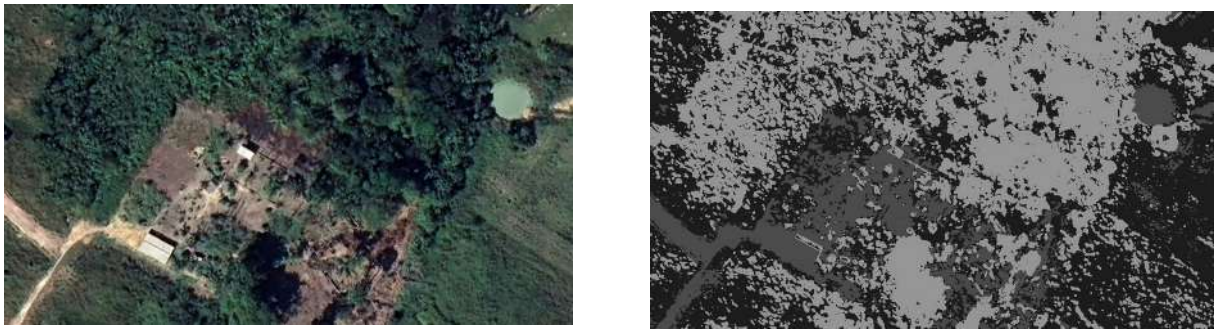
Vale lembrar que todos os códigos e arquivos utilizados para compor este trabalho estão disponíveis no GitHub a seguir para que os resultados obtidos possam ser reproduzidos: https://github.com/PauloTavernaro/classificacao-imagens.

6 CONCLUSÃO

O propósito deste trabalho foi fornecer uma abordagem de classificação de imagem por meio de segmentação de fotos da região amazônica, mais especificamente da cidade de São Félix do Xingu-PA. Nesse sentido, o objetivo era evidenciar elementos dessas fotos de acordo com três classes: Vegetação de Floresta (VF), Vegetação Rasteira e Arbustos (VRA) e Sem Vegetação (SV). Assim, através de uma análise temporal da mesma área geográfica, seria possível constatar alterações do ambiente, sugerindo, por exemplo, a presença de desmatamento em regiões de floresta, auxiliando o poder público e autoridades competentes na busca de soluções para o problema.

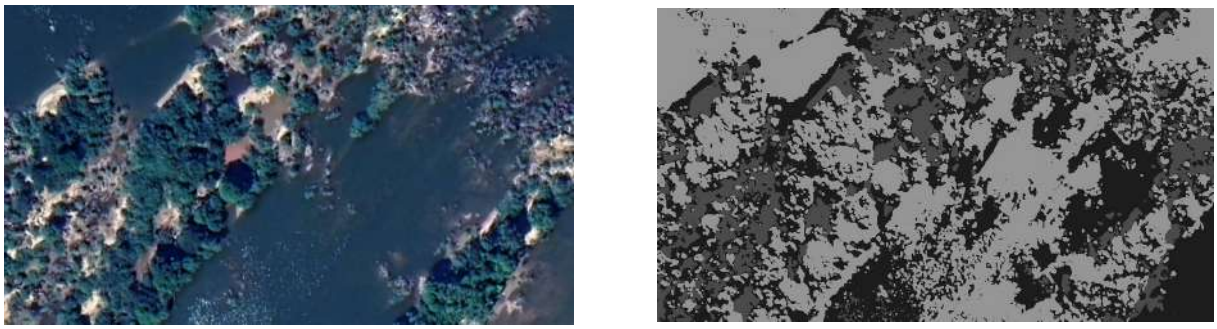
Como visto a partir da seção de resultados, a performance geral do modelo de *machine learning* para classificação das imagens não foi tão interessante. Com uma exatidão de 66,62%, não há garantia de que grande parte das fotos a serem analisadas pelo modelo sejam representadas com precisão suficiente para determinar as fronteiras de cada uma das classes em cada imagem. As Figuras 31 e 32 ilustram o comportamento do modelo frente a fotos não treinadas (portanto desconhecidas para ele) em relação a sua versão original. Enquanto a Figura 31 apresenta uma boa detecção, a Figura 32 evidencia que para determinadas intensidades de brilho e cor, além de texturas, o modelo apresenta falhas.

Figura 31 – Exemplo de uma boa detecção de classes pelo modelo.



Fonte: Google Earth (2023); Compilação do autor.

Figura 32 – Exemplo de uma detecção falha pelo modelo (rio confundido com floresta).



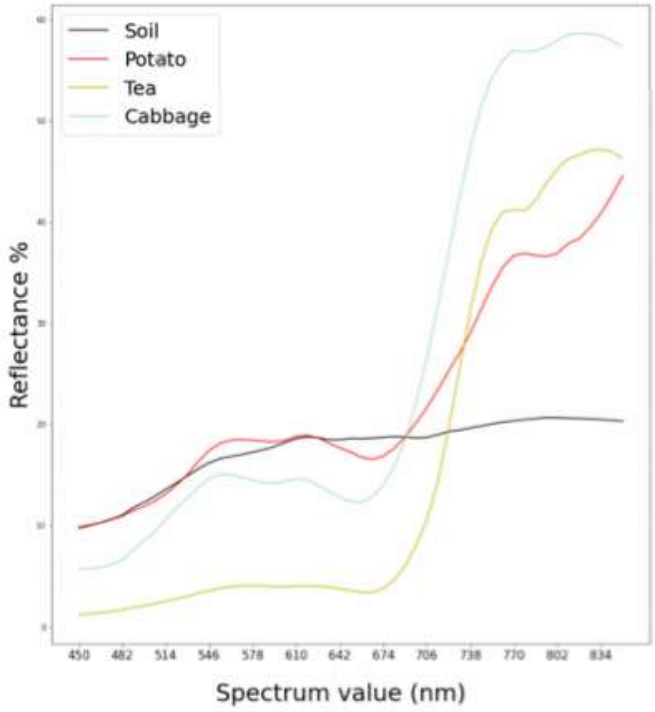
Fonte: Google Earth (2023); Compilação do autor.

Nesse sentido, é possível que esse comportamento tenha acontecido devido a dois fatores. O primeiro diz respeito à conversão das imagens originais da escala RGB para a escala de cinza. Nesse processo, três canais de cores foram redimensionados em apenas um, o que pode ter diminuído significantemente a quantidade de informações de cada pixel a cada filtro. O segundo fator leva em consideração à estratégia adotada para a análise das imagens. Em vez de analisá-las por inteiro, o processo de classificação considerou cada pixel individualmente, fora do contexto da foto, além de redefinir o hiperplano e suas margens a cada iteração a partir de apenas um pixel dos 414.720 (parcela de treino de cada imagem) em vez de, por exemplo, considerar cada amostra como uma imagem em si.

Outro ponto a ser considerado é a qualidade das fotos obtidas para treinar o modelo. Como discutido, as fotos obtidas do Google Earth tinham distância do solo de 600 m, além de terem qualidade insuficiente para a análise deste trabalho, pois com baixa resolução, a dificuldade em definir regiões de fronteira em uma foto aumenta. Assim, uma possibilidade seria registrar as fotos a partir de uma câmera hiperespectral acoplada em um *drone*, cuja capacidade de captura compreende dezenas de bandas do espectro eletromagnético, fornecendo muito mais informações do que as imagens em RGB devido à alta resolução espectral (PERERA; PREMACHANDRA; KAWANAKA, 2023). Dessa forma, observando o grau de refletância do ambiente analisado a uma distância menor do solo, é possível distinguir cada um dos componentes de uma imagem em uma determinada banda do espectro eletromagnético. Com isso, problemas enfrentados quanto à semelhança de intensidades de brilho (como no caso de sombra de construções,

intensidade de brilho de florestas e de rios) poderiam ser resolvidos. A Figura 32 mostra um exemplo da diferença espectral na análise do solo e de plantações de batata, chá e repolho para diferentes bandas. É evidente a diferença desses quatro elementos entre as bandas de 738 e 834 nm.

Figura 33 - Composição espectral de imagens de solo e plantações de batata, chá e repolho.



Fonte: Compilação do autor[45] .

Por outro lado, apesar da performance do modelo não ter atingido o nível de excelência esperado, foi possível implementá-lo com sucesso no módulo embarcado. Assim, a conversão de arquitetura do binário do modelo, além de toda a estrutura de conteinerização foram executadas de maneira efetiva.

Com isso, sendo possível aplicar o modelo em um sistema embarcado, torna-se mais próxima a proposta de implementar o módulo embarcado com o modelo de ML em um *drone* para que o processo das imagens aconteça logo após a captura

45 Imagem retirada de PERERA, C. J.; PREMACHANDRA, C.; KAWANAKA, H. Comparison of Light Weight Hyperspectral Camera Spectral Signatures with Field Spectral Signatures for Agricultural Applications, 2023, com edição.

delas. Ainda, sendo possível implementar uma câmera hiperespectral para realizar a captura das imagens, após o processamento delas e a análise da melhor banda espectral, espera-se que a performance do modelo de ML seja aperfeiçoada e sua exatidão aumentada. Portanto, o objetivo final de analisar uma mesma área geográfica ao longo do tempo terá maior precisão e assim, maior veracidade dos dados em estudo.

APÊNDICE A – CÓDIGO COMENTADO DA OBTENÇÃO DO MODELO DE *MACHINE LEARNING*

```
import numpy as np
import cv2
import pandas as pd
import pickle
import os
from skimage.filters import roberts, sobel_h, sobel_v, prewitt_h, prewitt_v
from scipy import ndimage as nd
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.utils import compute_class_weight
from skimage.segmentation import chan_vese
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import classification_report

#Filtro Gabor, que resulta em 48 versões de filtro com diferentes perfis de
intensidade e foco

def filtro_Gabor(imagem):
    df = pd.DataFrame()
    index = 1
    nucleos = []
    for theta in range(1,4):
        theta = theta / 4 * np.pi
        for sigma in (1,3):
            for lamda in np.arange(0, np.pi, np.pi/4):
                for gama in (0.05, 0.5):
                    gabor_index = 'Gabor' + str(index)
                    tamanho_nucleo = 2
                    nucleo = cv2.getGaborKernel((tamanho_nucleo,tamanho_nucleo),
sigma, theta, lamda, gama, 0, ktype=cv2.CV_32F)
                    nucleos.append(nucleo)
                    imagem_filtrada = cv2.filter2D(imagem, cv2.CV_8UC3, nucleo)
                    imagem_redimensionada = imagem_filtrada.reshape(-1)
                    df_red = pd.DataFrame(imagem_redimensionada, columns =
[gabor_index])
                    df = pd.concat([df, df_red], axis = 1)
                    index += 1
    return df

#Função que retorna como atributos de cada pixel o próprio valor de cada um deles

def Original(imagem):
    original_redimensionado = imagem.reshape(-1)
    df_red = pd.DataFrame(original_redimensionado, columns = ['Pixels Originais'])
```

```
    return df_red

#Filtro Canny, detecta bordas

def Canny_Edge(imagem):
    magnitude = cv2.Canny(imagem, 100, 200)
    #Padronizando os dados entre 0 e 255
    magnitude = magnitude * 255/np.max(magnitude)
    #Arredondando os dados
    magnitude = np.round(magnitude)
    #Convertendo para inteiro
    magnitude = magnitude.astype(int)
    canny_redimensionado = magnitude.reshape(-1)
    df_red = pd.DataFrame(canny_redimensionado, columns = ['Canny Edge'])

    return df_red

#Filtro Roberts, detecta bordas

def Roberts(imagem):
    magnitude = roberts(imagem)
    #Padronizando os dados entre 0 e 255
    magnitude *= 255/np.max(magnitude)
    #Arredondando os dados
    magnitude = np.round(magnitude)
    #Convertendo para inteiro
    magnitude = magnitude.astype(int)
    roberts_redimensionado = magnitude.reshape(-1)
    df_red = pd.DataFrame(roberts_redimensionado, columns = ['Roberts'])

    return df_red

#Filtro Sobel, detecta bordas

def Sobel(imagem):
    #Calculando o gradiente Sobel para cada direção (horizontal e vertical)
    sobel_x = sobel_h(imagem)
    sobel_y = sobel_v(imagem)
    #Padronizando os dados entre 0 e 255
    magnitude = np.sqrt(sobel_x**2 + sobel_y**2)
    magnitude *= 255/np.max(magnitude)
    #Arredondando os dados
    magnitude = np.round(magnitude)
    #Convertendo para inteiro
    magnitude = magnitude.astype(int)
    sobel_redimensionado = magnitude.reshape(-1)
    df_red = pd.DataFrame(sobel_redimensionado, columns = ['Sobel'])
```

```
    return df_red

#Filtro Prewitt, detecta bordas

def Prewitt(imagem):
    #Calculando o gradiente Prewitt para cada direção (horizontal e vertical)
    prewitt_x = prewitt_h(imagem)
    prewitt_y = prewitt_v(imagem)
    #Padronizando os dados entre 0 e 255
    magnitude = np.sqrt(prewitt_x**2 + prewitt_y**2)
    magnitude *= 255/np.max(magnitude)
    #Arredondando os dados
    magnitude = np.round(magnitude)
    #Convertendo para inteiro
    magnitude = magnitude.astype(int)
    prewitt_redimensionado = magnitude.reshape(-1)
    df_red = pd.DataFrame(prewitt_redimensionado, columns = ['Prewitt'])

    return df_red

#Filtro Gaussiano, aplica desfoque nas imagens

def Gaussiana(imagem, sigma):
    imagem_gaussiana = nd.gaussian_filter(imagem, sigma=sigma)
    gaussiana_redimensionada = imagem_gaussiana.reshape(-1)
    df_red = pd.DataFrame(gaussiana_redimensionada, columns =
[f'Gaussiana_s{sigma}'])

    return df_red

#Filtro Mediana, reduz ruído

def Mediana(imagem):
    imagem_mediana = nd.median_filter(imagem, size=3)
    mediana_redimensionada = imagem_mediana.reshape(-1)
    df_red = pd.DataFrame(mediana_redimensionada, columns = ['Mediana'])

    return df_red

#Filtro Chan-Vese, detecta regiões da imagem

def Chan_Vese(imagem):
    imagem_chan_vese = chan_vese(imagem, mu=0.25, lambda1=1, lambda2=1,
tol=1e-3, max_num_iter=200, dt=0.5, init_level_set="checkerboard",
extended_output=True)
    array_chan_vese = np.array(imagem_chan_vese[1])
    #Normalizando os dados
```

```
    array_chan_vese *= 255/np.max(array_chan_vese)
    #Arredondando os dados
    array_chan_vese = np.round(array_chan_vese)
    #Convertendo para inteiro
    array_chan_vese = array_chan_vese.astype(int)
    chan_vese_redimensionada = array_chan_vese.reshape(-1)
    df_red = pd.DataFrame(chan_vese_redimensionada, columns = ['Chan_Vese'])

    return df_red

#Função que aplica os filtros a cada imagem, e organiza seus pixeis em um
dataframe de dimensão
#518.400 (arranjos de pixeis) por 56 (quantidade de atributos de cada arranjo), que
será anexado
#em um dicionário cuja chave é sua imagem

def definir_filtros(caminho_imagens):
    #Definindo o dicionário que terá todos os dataframes de cada imagem
    dict_features = {}

    i = 1

    for imagem_treino in sorted(os.listdir(caminho_imagens)):

        #Definindo o dataframe final de cada imagem
        dados_todos_filtros = pd.DataFrame()
        #Garantindo que os dataframes parciais de cada filtro sejam reiniciados a cada
loop
        df_gabors = pd.DataFrame()
        df_original = pd.DataFrame()
        df_canny = pd.DataFrame()
        df_roberts = pd.DataFrame()
        df_sobel = pd.DataFrame()
        df_prewitt = pd.DataFrame()
        df_gaussiana_s3 = pd.DataFrame()
        df_gaussiana_s7 = pd.DataFrame()
        df_mediana = pd.DataFrame()
        df_chan_vese = pd.DataFrame()

        #Lendo a imagem
        imagem = cv2.imread(caminho_imagens + imagem_treino)

        #Garantindo que tenha o tamanho de 960x540 px
        imagem = cv2.resize(imagem, (960, 540))

        #Como a biblioteca OpenCV lê as imagens na escala BGR, elas são
primeiramente convertidas
        #em RGB, e depois em escala de cinza
```

```
        imagem_RGB = cv2.cvtColor(imagem, cv2.COLOR_BGR2RGB)
        imagem_tons_cinza = cv2.cvtColor(imagem_RGB, cv2.COLOR_RGB2GRAY)

        #Aplicando os filtros para cada imagem
        df_gabors = filtro_Gabor(imagem_tons_cinza)
        df_original = Original(imagem_tons_cinza)
        df_canny = Canny_Edge(imagem_tons_cinza)
        df_roberts = Roberts(imagem_tons_cinza)
        df_sobel = Sobel(imagem_tons_cinza)
        df_prewitt = Prewitt(imagem_tons_cinza)
        df_gaussiana_s3 = Gaussiana(imagem_tons_cinza, 3)
        df_gaussiana_s7 = Gaussiana(imagem_tons_cinza, 7)
        df_mediana = Mediana(imagem_tons_cinza)
        df_chan_vese = Chan_Vese(imagem_tons_cinza)

        #Reunindo todos os dataframes de cada filtro um outro dataframe com todos
os atributos
        #de cada filtro alinhados horizontalmente
        dados_todos_filtros = pd.concat([df_gabors, df_original, df_canny, df_roberts,
df_sobel, df_prewitt, df_gaussiana_s3, df_gaussiana_s7, df_mediana,
df_chan_vese], axis=1)

        #Preenchendo o dicionário com o dataframe anterior e sua respectiva imagem
        dict_features.update({f'Imagem_{i}' : dados_todos_filtros})

        i += 1

    return dict_features

#Função que recebe cada imagem de referência e organiza seus pixeis em um
dataframe de dimensão
#518.400 (pixeis) por 1 (atributo de classificação), que será anexado em um
dicionário cuja
#chave é sua imagem

def definir_referencia(caminho_imagens):
    #Definindo o dicionário que terá todos os dataframes de cada imagem
    dict_ref = {}

    j = 1

    for imagem_referencia in sorted(os.listdir(caminho_imagens)):

        #Garantindo que o dataframe final de cada imagem seja reiniciado a cada loop
        dados_categorizadas = pd.DataFrame()

        #Lendo a imagem
        imagem = cv2.imread(caminho_imagens + imagem_referencia)
```

```
        #Garantindo que tenha o tamanho de 960x540 px
        imagem = cv2.resize(imagem, (960, 540))

        #Como a biblioteca OpenCV lê as imagens na escala BGR, elas são
primeiramente convertidas
        #em RGB, e depois em escala de cinza, ainda que as imagens de referência já
estejam na
        #escala de cinza
        imagem_RGB = cv2.cvtColor(imagem, cv2.COLOR_BGR2RGB)
        imagem_tons_cinza = cv2.cvtColor(imagem_RGB, cv2.COLOR_RGB2GRAY)

        #Redimensionando cada imagem para ter apenas uma dimensão
        dados_imagem_ref = imagem_tons_cinza.reshape(-1)

        #Organizando os dados de cada imagem em um dataframe
        dados_categorizadas = pd.DataFrame(dados_imagem_ref, columns =
['Imagens de Referência'])

        #Preenchendo o dicionário com o dataframe anterior e sua respectiva imagem
        dict_ref.update({f'Imagem_Ref_{j}':dados_categorizadas})

        j += 1

    return dict_ref

#Função que divide cada dataframe de X e y em grupos de treino e teste para que
então o modelo seja
#treinado em etapas, ou seja, a partir dos dados de cada imagem para economizar
armazenamento de RAM

def divisao_treino_teste(X, y, model, i, todos_X_teste, todos_y_teste):

    #Divide os valores de X e y em grupos de treino e teste. A parcela de teste é de
20%
    #Além disso, "random_state" garante a reprodutibilidade
    X_treino, X_teste, y_treino, y_teste = train_test_split(X, y, test_size=0.2,
random_state=1)

    #Convertendo o dataframe em arranjos NumPy
    X_treino = X_treino.to_numpy()
    y_treino = y_treino.to_numpy().ravel()

    #Treinando o modelo em partes (dados de uma imagem por vez)
    #As classes correspondem às classificações:
    #29: Vegetação Rasteira e Arbustos
    #76: Sem Vegetação
    #150: Vegetação de Floresta
```

```
    model.partial_fit(X_treino, y_treino, classes=[29, 76, 150])

    #Convertendo o dataframe em arranjos NumPy
    X_teste = X_teste.to_numpy()
    y_teste = y_teste.to_numpy().ravel()


    #Concatenando todos os dados de teste de todas as fotos até o momento
    #para que o modelo resultante de todas as iterações até então avalie todas
    #as informações de teste até o momento
    if i == 1:
        todos_y_teste = y_teste
        todos_X_teste = X_teste
    else:
        todos_y_teste = np.append(todos_y_teste, y_teste)
        todos_X_teste = np.append(todos_X_teste, X_teste, axis=0)

    #Obtendo a exatidão do modelo
    print(f"Exatidão depois da Imagem {i} = {model.score(todos_X_teste,
todos_y_teste)}")

    #Realizando a previsão de resposta do modelo frente a todos os dados de teste
de X até então
    previsao = model.predict(todos_X_teste)

    #Obtendo o relatório de classificação, o qual retorna valores como precisão e
recall
    print(f"Relatório de Classificação depois da Imagem {i}: \
n{metrics.classification_report(todos_y_teste, previsao)}\n")

    return model, todos_X_teste, todos_y_teste


##########################################################

#Função principal

if __name__ == "__main__":

    #Definindo dataframes para captar os dados das imagens
    todos_original = pd.DataFrame()
    todos_categorizadas = pd.DataFrame()
    todos_X_teste = pd.DataFrame()
    todos_y_teste = []
    todos_y_teste = np.array(todos_y_teste)

    #Definindo os caminhos para encontrar as fotos
    caminho_imagens_original = '/home/user/Área de Trabalho/Originais/'
```

```
    caminho_imagens_categorizadas = '/home/user/Área de Trabalho/Referencia/'


    #Transformando os dados em listas para facilitar a iteração
    dados_original = list(definir_filtros(caminho_imagens_original).values())
    dados_categorizadas =
list(definir_referencia(caminho_imagens_categorizadas).values())


    #Iteração de cada foto pelo treino e teste do modelo
    for i, (X, y) in enumerate(zip(dados_original, dados_categorizadas), 1):

        if i == 1:

            #Peso das classes sendo definidos
            peso = compute_class_weight(class_weight='balanced',
classes=np.unique(y), y=y.to_numpy().ravel())

            modelo_inicial = lambda peso, y : SGDClassifier(loss='hinge', shuffle=True,
random_state=1, warm_start=True, learning_rate='adaptive', eta0 = 0.01,
class_weight=dict(zip(np.unique(y), peso)), average=True)

            modelo_pronto, todos_X_teste, todos_y_teste = divisao_treino_teste(X, y,
modelo_inicial(peso, y), i, todos_X_teste, todos_y_teste)

        else:

            #Peso das classes sendo definidos
            peso = compute_class_weight(class_weight='balanced',
classes=np.unique(y), y=y.to_numpy().ravel())

            modelo_pronto.set_params(**{'class_weight' : dict(zip(np.unique(y), peso))})

            modelo_pronto, todos_X_teste, todos_y_teste = divisao_treino_teste(X, y,
modelo_pronto, i, todos_X_teste, todos_y_teste)


    #Calculando a matriz de confusão final
    previsao = modelo_pronto.predict(todos_X_teste)

    print(f'\nMatriz de Confusão: \n{metrics.confusion_matrix(todos_y_teste, previsao,
labels=[29, 76, 150])}')


    #Salvando o modelo treinado
    caminho_modelo = "/home/user/Área de Trabalho/Modelo_ML"
    pickle.dump(modelo_pronto, open(caminho_modelo, 'wb'))
```

REFERÊNCIAS

. Biomas brasileiros. 1. ed. São Paulo: Oficina de Textos, 2016. E-book. Disponível em: https://plataforma.bvirtual.com.br. Acesso em: 23 out. 2023.

. Secas na Amazônia: causas e consequências. 1. ed. São Paulo: Oficina de Textos, 2013. E-book. Disponível em: https://plataforma.bvirtual.com.br. Acesso em: 23 out. 2023.

1.4. Support Vector Machines. scikit learn, 2023-b. Disponível em: https://scikit-learn.org/stable/modules/svm.html. Acesso em: 20 nov. 2023.

AKAL, Orhan; BARBU, Adrian. Learning Chan-Vese, Florida State University, 2019. Disponível em: https://ani.stat.fsu.edu/~abarbu/papers/2019-LearnCV-ICIP.pdf. Acesso em: 19 nov. 2023.

ALSHEHRI, M.; OUADOU, A.; SCOTT, G. Deforestation Detection in the Brazilian Amazon Using Transformer-based Networks, 2023 IEEE Conference on Artificial Intelligence (CAI), Santa Clara, CA, USA, 2023, pp. 292-293, doi: 10.1109/CAI54212.2023.00130. Disponível em: https://ieeexplore.ieee.org/document/10194966. Acesso em: 29 nov. 2023.

AMORIM, Antonio; POLASTRI, Maria Julia. Evaluation of Edge Detection Filters Applied to Corroded Steel Sheets, 2020. International Journal of Science and Research (IJSR). 5. 898-902. Disponível em: https://www.researchgate.net/publication/342820766_Evaluation_of_Edge_Detection_Filters_Applied_to_Corroded_Steel_Sheets. Acesso em: 23 nov. 2023.

Arco do desmatamento. IPAM Amazônia. Disponível em: https://ipam.org.br/glossario/arco-do-desmatamento/. Acesso em: 26 out. 2023.

BARTÍK, M.; PICHLOVÁ, D.; KUBÁTOVÁ, H. Hardware-software co-design: A practical course for future embedded engineers, 2016 5th Mediterranean Conference on Embedded Computing (MECO), Bar, Montenegro, 2016, pp. 347-350, doi: 10.1109/MECO.2016.7525779. Disponível em: https://ieeexplore.ieee.org/document/7525779. Acesso em: 29 nov. 2023.

BHATTIPROLU, S. 63 - Image Segmentation using traditional machine learning Part1 - FeatureExtraction [Video], 2019. Disponível em: https://www.youtube.com/watch?v=6yW31TT6-wA&list=PLZsOBAyNTZwYHBIlu_PUO19M7aHMgwBJr&index=68. Acesso em: 29 out. 2023.

BECKER, Bertha Koiffmann; STENNER, Claudio. Um futuro para a Amazônia. 1. ed. São Paulo: Oficina de Textos, 2008. E-book. Disponível em: https://plataforma.bvirtual.com.br. Acesso em: 24 out. 2023.

CATAPANG, J. K. Optimizing Speed and Accuracy Trade-off in Machine Learning Models via Stochastic Gradient Descent Approximation, 2022 9th International Conference on Soft Computing & Machine Intelligence (ISCMI), Toronto, ON, Canada, 2022, pp. 124-128, doi: 10.1109/ISCMI56532.2022.10068476. Disponível em: https://ieeexplore.ieee.org/document/10068476. Acesso em: 21 nov. 2023.

CHUGH, Roger Singh et al. A Comparative Analysis of Classifiers for Image Classification, 2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence), Noida, India, 2020, pp. 248-253, doi: 10.1109/Confluence47617.2020.9058042. Disponível em: https://ieeexplore.ieee.org/document/9058042. Acesso em: 19 nov. 2023.

Computador em Módulo NXP® i.MX 8: Apalis iMX8. Toradex, 2023-a. Disponível em: https://www.toradex.com/pt-br/computer-on-modules/apalis-arm-family/nxp-imx-8. Acesso em: 29 nov. 2023.

DING, Lijun; GOSHTASBY, Ardeshir. On the Canny edge detector, Pattern Recognition, Volume 34, Issue 3, 2001, Pages 721-725, ISSN 0031-3203, https://doi.org/10.1016/S0031-3203(00)00023-6 (https://www.sciencedirect.com/science/article/pii/S0031320300000236)

FISHER, R. et al. Roberts Cross Edge Detector, 2003-a, The University of Edinburgh. Disponível em: https://homepages.inf.ed.ac.uk/rbf/HIPR2/roberts.htm. Acesso em: 18 nov. 2023.

FISHER, R. et al. Sobel Edge Detector, 2003-b, The University of Edinburgh. Disponível em: https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm. Acesso em: 18 nov. 2023.

FISHER, R. et al. Gaussian Smoothing, 2003-c, The University of Edinburgh. Disponível em: https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm. Acesso em: 18 nov. 2023.

FISHER, R. et al. Median Filter, 2003-d, The University of Edinburgh. Disponível em: https://homepages.inf.ed.ac.uk/rbf/HIPR2/median.htm. Acesso em: 18 nov. 2023.

Fronteira do desmatamento na Amazônia avançou entre 2018 e 2019, afirma estudo. O Globo, 2019. Disponível em: https://oglobo.globo.com/brasil/fronteira-do-desmatamento-na-amazonia-avancou-entre-2018-2019-afirma-estudo-1-24141480. Acesso em: 27 out. 2023.

GOOGLE MACHINE LEARNING. Classificação: acurácia, 2023. Disponível em: https://developers.google.com/machine-learning/crash-course/classification/accuracy?hl=pt-br#:~:text=Informally%2C%20accuracy%20is%20the%20fraction,predictions%20Total%20number%20of%20predictions. Acesso em: 21 nov. 2023.

GRILLI, Mariana. São Félix do Xingu é o município que mais emite CO2 no Brasil. Globo Rural, 04 mar. 2021. Disponível em: https://globorural.globo.com/Noticias/Sustentabilidade/noticia/2021/03/sao-felix-do-xingu-e-o-municipio-que-mais-emite-co2-no-brasil.html. Acesso em: 27 out. 2023.

Image Classification and Analysis. Government of Canada, 2023. Disponível em: https://natural-resources.canada.ca/maps-tools-and-publications/satellite-imagery-and-air-photos/tutorial-fundamentals-remote-sensing/image-interpretation-analysis/image-classification-and-analysis/9361. Acesso em: 17 nov. 2023.

Ixora Carrier Board. Toradex, 2023-b. Disponível em: https://www.toradex.com/pt-br/products/carrier-board/ixora-carrier-board. Acesso em: 30 nov. 2023.

KANDEL, Ibrahem; CASTELLI, Mauro; POPOVIČ, Aleš. Comparative Study of First Order Optimizers for Image Classification Using Convolutional Neural Networks on Histopathology Images, National Library of Medicine, 2020. Disponível em: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8321140/. Acesso em: 28 nov. 2023.

KULKARNI, Ajay; CHONG, Deri; BATARSEH, Feras A. Data Democracy, At the Nexus of Artificial Intelligence, Software Development, and Knowledge Engineering,

2020, Pages 83-106. Disponível em: https://www.sciencedirect.com/science/article/pii/B9780128183663000058. Acesso em: 21 nov. 2023.

MEHROTRA R., NAMUDURI K.R., RANGANATHAN N. Gabor filter-based edge detection, Pattern Recognition, Volume 25, Issue 12, 1992, Pages 1479-1494, ISSN 0031-3203, https://doi.org/10.1016/0031-3203(92)90121-X. (https://www.sciencedirect.com/science/article/pii/003132039290121X)

NIXON, Mark; AGUADO, Alberto. Gaussian Filtering, University of Southampton, 2002. Disponível em: https://www.southampton.ac.uk/~msn/book/new_demo/gaussian/. Acesso em: 18 nov. 2023.

O que é machine learning?. IBM, 2023. Disponível em: https://www.ibm.com/br-pt/topics/machine-learning. Acesso em: 16 nov. 2023.

pandas.DataFrame. pandas, 2023. Disponível em: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html. Acesso em: 17 nov. 2023.

PERERA, C. J.; PREMACHANDRA, C.; KAWANAKA, H. Comparison of Light Weight Hyperspectral Camera Spectral Signatures with Field Spectral Signatures for Agricultural Applications, 2023 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 2023, pp. 1-3, doi: 10.1109/ICCE56470.2023.10043396. Disponível em: https://ieeexplore.ieee.org/document/10043396. Acesso em: 29 nov. 2023.

PISL, J. et al. Classification of Tropical Deforestation Drivers with Machine Learning and Satellite Image Time Series, IGARSS 2023 - 2023 IEEE International Geoscience and Remote Sensing Symposium, Pasadena, CA, USA, 2023, pp. 911-914, doi: 10.1109/IGARSS52108.2023.10281472. Disponível em: https://ieeexplore.ieee.org/document/10281472. Acesso em: 29 nov. 2023.

SAHU, Chandan Kumar; SHAMAR, Maitrey. HINGE LOSS IN SUPPORT VECTOR MACHINES, School of Computer Sciences, National Institute of Science Education and Research, Bhubaneshwar, Homi Bhabha National Institute, 2023. Disponível em: https://www.niser.ac.in/~smishra/teach/cs460/23cs460/lectures/lec11.pdf. Acesso em: 28 nov. 2023.

sklearn.linear_model.SGDClassifier. scikit learn, 2023-a. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html. Acesso em: 20 nov. 2023.

SHAHAJAD M., GAMBHIR D. and GANDHI R. Features extraction for classification of brain tumor MRI images using support vector machine, 2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence), Noida, India, 2021, pp. 767-772, doi: 10.1109/Confluence51648.2021.9377111. Disponível em: https://ieeexplore.ieee.org/document/9377111. Acesso em: 17 nov. 2023.

SHIJIN KUMAR P. S.; DHARUN, V. S. Extraction of Texture Features using GLCM and Shape Features using Connected Regions, Int. J. Eng. Technol, vol. 8, no. 6, pp. 29262930, 2016. Disponível em: https://www.enggjournals.com/ijet/docs/IJET16-08-06-254.pdf. Acesso em: 17 nov. 2023.

The University of Auckland. Prewitt filter, 2023. Disponível em: https://www.cs.auckland.ac.nz/courses%0B/compsci373s1c/PatricesLectures/Prewitt_2up.pdf. Acesso em 18 nov, 2023.

Torizon OS Technical Overview. Toradex, 2023-c. Disponível em: https://developer.toradex.com/torizon/torizoncore/torizoncore-technical-overview/. Acesso em: 30 nov. 2023.

Universidade Federal Fluminense (UFF). Canny: detecção de borda, 2023. Disponível em: http://profs.ic.uff.br/~aconci/canny.pdf. Acesso em: 18 nov. 2023.

Use containers to Build, Share and Run your applications. Docker, 2023. Disponível em: https://www.docker.com/resources/what-container/. Acesso em: 30 nov. 2023.

ZANOTTA, D. C. et al. Automatic Methodology for Mass Detection of Past Deforestation in Brazilian Amazon, IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium, Yokohama, Japan, 2019, pp. 6610-6613, doi: 10.1109/IGARSS.2019.8898606. Disponível em: https://ieeexplore.ieee.org/document/8898606. Acesso em: 29 nov. 2023.