UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E ELETRÔNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Luan Zehnder Cancelier

Comparação entre técnicas de classificação de imagens para uso em pequenos satélites

Florianópolis
2025

Luan Zehnder Cancelier

Comparação entre técnicas de classificação de imagens para uso em pequenos satélites

Trabalho de Conclusão de Curso submetido ao curso de Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito parcial para a obtenção do título de bacharel em Engenharia Elétrica.

Orientador(a): Prof. Eduardo Augusto Bezerra, Dr.

Florianópolis
2025

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC. Dados inseridos pelo próprio autor.

Cancelier, Luan Zehnder
Comparação entre técnicas de classificação de imagens para uso em pequenos satélites / Luan Zehnder Cancelier ; orientador, Eduardo Augusto Bezerra, 2025.
66 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Elétrica, Florianópolis, 2025.

Inclui referências.

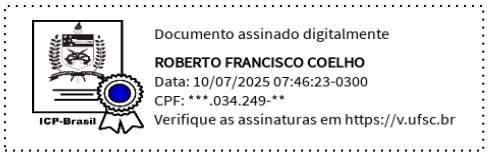
1. Engenharia Elétrica. 2. Processamento de Imagens. 3. CubeSat. 4. Machine Learning. 5. Classificação de Imagens. I. Bezerra, Eduardo Augusto . II. Universidade Federal de Santa Catarina. Graduação em Engenharia Elétrica. III. Título.

Luan Zehnder Cancelier

Comparação entre técnicas de classificação de imagens para uso em pequenos satélites

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do título de “Bacharel em Engenharia Elétrica” e aprovado em sua forma final pelo Curso de Graduação em Engenharia Elétrica.

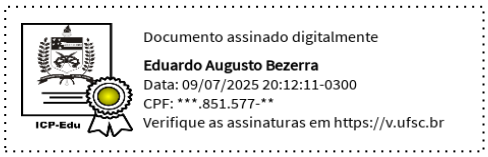
Florianópolis, 02 de julho de 2025.



Prof. Roberto Francisco Coelho, Dr.

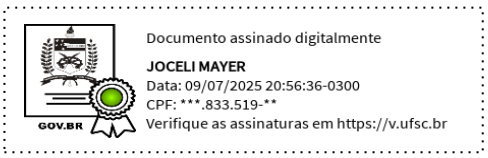
Coordenador do Curso de Graduação em Engenharia Elétrica

Banca examinadora



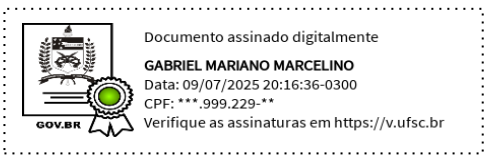
Prof. Eduardo Augusto Bezerra, Dr.

Orientador



Prof. Joceli Mayer, Dr.

Universidade Federal de Santa Catarina



Gabriel Mariano Marcelino, Dr.

Universidade Federal de Santa Catarina

Florianópolis, 2025.

AGRADECIMENTOS

Em primeiro lugar, agradeço aos meus pais, Kestin e Norton, que sempre estiveram ao meu lado em todos os momentos, me apoiando e me inspirando a seguir esse caminho. Muito além do apoio, foram eles que me ensinaram os valores que hoje carrego e que me ajudaram a enfrentar os desafios ao longo da vida. Nada disso teria sido possível sem o exemplo e o amor de vocês.

Também sou grato ao meu irmão, Jonathan, que, além de grande parceiro, foi uma importante referência ao escolher a Engenharia Elétrica. Seguir seus passos foi algo natural, e sua trajetória sempre me motivou a acreditar que eu também era capaz.

Aos amigos que me acompanharam durante a graduação, meu muito obrigado. Dividir essa fase com vocês tornou tudo mais leve e divertido. Foram muitas conversas, desafios, aprendizados e boas risadas. Em especial, agradeço ao João e ao Bruno, com quem tive o privilégio de dividir moradia ao longo de toda a graduação. Obrigado por suportarem meu estresse, por me escutarem nos momentos difíceis e, principalmente, por fazerem parte de tantas memórias boas que vou levar comigo para sempre.

Estendo meus agradecimentos a todos os professores do Departamento de Engenharia Elétrica e Eletrônica e do Centro Tecnológico assim como aos servidores da UFSC, que contribuíram para o meu crescimento acadêmico e pessoal ao longo desses anos. Cada aula, cada conversa e cada desafio vencido somaram para a construção do profissional que estou me tornando.

Por fim, agradeço ao meu orientador, professor Eduardo Augusto Bezerra, pela paciência, orientação, incentivo e confiança durante todo o desenvolvimento deste trabalho. Seu apoio foi essencial para que este projeto se concretizasse.

RESUMO

O presente trabalho realiza uma análise comparativa entre técnicas convencionais de processamento de imagens e redes neurais convolucionais aplicadas à classificação de imagens capturadas por pequenos satélites. Devido às restrições de memória, capacidade de processamento e consumo energético desses dispositivos, torna-se essencial encontrar métodos que sejam eficientes na classificação, ao mesmo tempo que possam ser implementados em sistemas embarcados. Inicialmente, foram aplicadas técnicas tradicionais de processamento, como detecção de bordas, análise de texturas e limiarização. Em seguida, modelos de aprendizado de máquina baseados em redes neurais convolucionais (CNNs) foram treinados, incluindo arquiteturas já existentes e um modelo genérico desenvolvido para fins comparativos. A base de dados utilizada no treinamento foi composta por imagens públicas disponibilizadas pela Estação Espacial Internacional e outras fontes, enquanto o conjunto de testes foi formado majoritariamente por imagens reais obtidas por *CubeSat*. Os métodos foram avaliados quanto ao desempenho de classificação e aos requisitos computacionais, considerando um microcontrolador STM32 como hardware de referência. Os resultados indicaram que os métodos tradicionais, embora menos precisos, apresentaram requisitos computacionais significativamente inferiores. Por outro lado, as CNNs, após a quantização, demonstraram bom desempenho classificatório com viabilidade de execução embarcada. Conclui-se que a escolha da técnica mais adequada depende do equilíbrio entre desempenho e as restrições do sistema embarcado.

Palavras-chave: CubeSat; Classificação de Imagens; Processamento Digital de Imagens; Redes Neurais Convolucionais; Sistemas Embarcados.

ABSTRACT

This work presents a comparative analysis between conventional image processing techniques and convolutional neural networks (CNNs) applied to the classification of images captured by small satellites. Due to the limited memory, processing capacity, and energy consumption of these devices, it is essential to find methods that are both efficient in classification and feasible for implementation in embedded systems. Initially, traditional image processing techniques were applied, such as edge detection, texture analysis, and thresholding. Then, machine learning models based on CNNs were trained, including previously developed architectures and a generic model created for comparative purposes. The training dataset was composed of public images provided by the International Space Station and other sources, while the test dataset consisted mostly of real CubeSat images. The methods were evaluated in terms of classification performance and computational requirements, using an STM32 microcontroller as reference hardware. The results indicated that traditional methods, although less accurate, required significantly fewer computational resources. On the other hand, CNNs, after quantization, demonstrated good classification performance and feasibility for embedded execution. It is concluded that the most suitable technique depends on the balance between performance and system constraints.

Keywords: CubeSat; Image Classification; Digital Image Processing; Convolutional Neural Networks; Embedded Systems.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

OpenCV	Open Source Computer Vision Library
NMS	Non-maximum Suppression
RGB	Red Green Blue
HSV	Hue Saturation Value
LBP	Local Binary Pattern
RNA	Redes Neurais Artificiais
ReLU	Rectified Linear Unit
MLP	Mutilayer Perceptron
CNN	Convolutional Neural Network
FC	Fully Connected
ISS	International Space Station
ESA	European Space Agency
IDE	Integrated Development Environment
ROC	Receiver Operating Characteristic Curve
AUC	Area Under ROC Curve

SUMÁRIO

1 INTRODUÇÃO

No contexto da exploração espacial, os *CubeSat* surgem como uma alternativa compacta e acessível ao se comparar com satélites tradicionais, que possuem não apenas um custo mais elevado, mas também precisam de instalações especiais para o desenvolvimento. Esses satélites em miniatura, caracterizados pelo formato cúbico e dimensões reduzidas, têm se destacado como parceiros valiosos na pesquisa espacial, oferecendo uma abordagem mais acessível em comparação aos satélites de médio e grande porte (SPACE, [s.d.]).

A principal vantagem decorrente das dimensões reduzidas dos *CubeSat* em relação aos satélites convencionais é notável. Esses satélites podem ser integrados como cargas secundárias em missões de maior envergadura, tornando-se uma opção economicamente viável e contribuindo para democratizar as oportunidades de pesquisa na exploração espacial (MANE, 2024).

Portanto, a tecnologia de *CubeSat* não apenas representa uma abordagem mais acessível e compacta, mas também abre caminho para uma participação mais ampla na pesquisa espacial, superando as limitações anteriores associadas a custos elevados e acesso restrito (MANE, 2024).

Os *CubeSat* coletam uma variedade de dados, incluindo imagens, que podem ser aprimoradas por meio de diversas técnicas de processamento. Entre essas técnicas, encontram-se métodos convencionais comumente empregados no processamento de imagens, métodos estes, que focam na extração das *features* da imagem de forma manual, ou seja, através de filtros específicos. Além disso, observa-se uma crescente popularização na adoção de tecnologias mais avançadas, como o aprendizado de máquina em que as features são extraídas das imagens com base em conhecimento prévio, ou seja, de forma automatica. Nesse sentido, destaca-se o uso de redes neurais convolucionais (CNNs), uma abordagem de machine learning que tem se destacado significativamente.

Para ilustrar a aplicação dessas técnicas, podemos considerar a utilização de filtros convolucionais para realçar características específicas em imagens capturadas pelos *CubeSat*. Por exemplo, um filtro de nitidez pode ser empregado para aprimorar bordas e detalhes, enquanto um filtro de suavização pode ser utilizado para reduzir o ruído nas imagens. Já no caso das redes neurais convolucionais, possuem capacidade de aprender padrões complexos e realizar tarefas como identificação de

objetos, classificação de cenas e até mesmo a geração de imagens aprimoradas (TOKI et al., 2024).

O cenário atual da exploração espacial impõe desafios singulares, especialmente no que diz respeito à tecnologia necessária para seu funcionamento pleno. Quando nos deparamos com a tecnologia de satélites CubeSat, novos obstáculos se apresentam, destacando-se as dimensões compactas e os recursos computacionais limitados. Diante desse contexto, surge uma demanda por estratégias que otimizem suas atividades, visando economizar banda de transmissão, dada a restrição das janelas de transmissão.

Estratégias eficazes para a seleção apenas dos dados relevantes para a missão tornam-se imprescindíveis (TOKI et al., 2024).

É importante ressaltar que a transmissão de imagens impróprias pode resultas em custos significativos para a missão, principalmente se tratando dos nanossatélites, a transmissão de dados envolve a ativação do sistema de comunicação o que em satélites com severas limitações energéticas como em CubeSat do tipo 1U por exemplo pode significar uma parcela considerável do budget energético sendo usada, além de utilizar o tempo restrito para envio de dados sem relevância cientifica (TOKI et al., 2024).

Nesse sentido, a classificação a bordo de imagens capturadas antes da transmissão emerge como um aliado crucial para a otimização dos dados. Dessa forma, pretende-se analisar técnicas de machine learning e processamento de imagens com o intuito de classificar de maneira eficiente, isto é, sem sobrecarregar o consumo energético — vital para o funcionamento dos outros subsistemas — e de forma ágil, diferenciando entre imagens terrestres, imagens espaciais e solares. Por meio desse pré-processamento, as imagens desnecessárias poderiam ser descartadas, em tempo real, poupando memória e permitindo a transmissão apenas do essencial.

1.1 OBJETIVOS

Nas próximas seções, são apresentados o objetivo geral e os objetivos específicos deste Trabalho de Conclusão de Curso (TCC).

1.1.1 Objetivo Geral

O objetivo geral deste trabalho se dá na análise de diferentes técnicas de classificação de imagem, incluindo o processamento de imagens convencional, e modelos de aprendizado de máquina. Tendo em vista as limitações computacionais e energéticas, a análise deve contemplar os aspectos computacionais de ambas as técnicas, e os seus compromissos (*trade-off*[1]).

1.1.2 Objetivos Específicos

Para atingir o objetivo geral deste trabalho, os seguintes objetivos específicos foram definidos:

- Estudar e descrever técnicas e ferramentas utilizadas no processamento digital de imagens, com foco em aplicações práticas;
- Analisar os requisitos e as particularidades do processamento digital de imagens no contexto espacial, com ênfase nas limitações impostas por pequenos satélites;
- Estudar e avaliar diferentes estratégias de aprendizado de máquina aplicadas à classificação de imagens, incluindo redes neurais convolucionais;
- Implementar e/ou simular as técnicas selecionadas em ambiente controlado, utilizando bases de dados compostas por imagens similares às capturadas por sensores embarcados em pequenos satélites;
- Avaliar o desempenho das técnicas de classificação quanto à acurácia, tempo de processamento e consumo de recursos computacionais;
- Comparar os resultados obtidos, identificando os principais compromissos (trade-offs) entre desempenho e viabilidade de implementação em plataformas com recursos restritos, como microcontroladores embarcados em pequenos satélites;

[1] Melhorar algum aspecto do projeto, porém, em troca, piorar em outro.

- Validar a viabilidade de execução embarcada das técnicas implementadas por meio de simulações em ambiente de desenvolvimento compatível com a arquitetura de microcontroladores utilizados em pequenos satélites;
- Sugerir diretrizes para a escolha de técnicas de classificação de imagens em missões espaciais com restrições computacionais semelhantes, considerando diferentes cenários de aplicação.

2 REVISÃO BIBLIOGRÁFICA

Na literatura, existem diferentes técnicas que foram utilizadas para classificar e segmentar imagens provindas de pequenos satélites, em (MASKEY, 2020), é apresentada uma arquitetura de rede neural convolucional capaz de fazer uma classificação binária de imagens espaciais entre ruins e boas, que seja compacta o suficiente para caber em um *CubeSat* do tipo 1U. O modelo final, testado em um banco de imagens teste com 30 imagens, atingiu uma acurácia de 90% e F1-Score 0,92 além de se mostrar capaz de reduzir o tempo de operação em 2/3 do que era originalmente considerado antes da implementação.

Em (TOKI et al., 2024), é feita uma pesquisa muito similar, uma arquitetura ainda mais compacta que a CubeSatNet (MASKEY, 2020) é proposta, contendo menos de 10.000 parâmetros, e com o modelo final tendo 15kB sendo capaz de ser implementado em muitos equipamentos de ponta. O modelo foi testado no mesmo banco de dados que (MASKEY, 2020), obtendo uma acurácia de 90% e F1-Score 0,91 provando que mesmo uma rede “rasa” é capaz de classificar as imagens de maneira satisfatória, não sendo necessários grandes custos energéticos ou computacionais.

No campo da segmentação de imagens, (ALEXSEI, 2024) propôs um algoritmo baseado em redes neurais convolucionais, capaz de segmentar imagens priorizando imagens para o *downlink* que possuam visão clara de corpos aquosos e vegetação, ou seja, que não estejam ofuscadas por nuvem ou outros tipos de ruídos. Um F1-Score de 0,85 pode ser obtido na segmentação, com recursos computacionais relativamente baixos sendo utilizados na segmentação.

Por fim, (YAO et al., 2019) propôs um modelo compacto o suficiente para ser implementado em nanossatélites, capaz de segmentar imagens de satélites mais especificamente buscando navios na imagem. O modelo alcançou uma precisão de 80,5% e revocação de 95,9% provando que mesmo aplicações um pouco mais complexas que a proposta neste trabalho, são factíveis nesse contexto.

3 FUNDAMENTAÇÃO TEÓRICA

Esta seção é destinada à apresentação da fundamentação teórica, responsável por embasar o desenvolvimento do trabalho. Serão apresentadas teorias gerais em relação ao processamento digital de imagens, e em relação a técnicas mais modernas de visão computacional, considerando as restrições e requisitos de uma missão espacial do tipo *CubeSat*.

3.1 LIMITAÇÕES DE DOWNLINK DOS CUBESATS

A comunicação com a estação terrestre se prova ser um desafio principalmente pela grande distância, o que gera elevada atenuação do sinal recebido. Além disso, existem limitações impostas pelos nanossatélites, como a área da superfície dos painéis solares são limitadas, a potência disponível para fazer o *downlink* via antena é limitada, assim como o ganho da antena devido a, na maioria dos casos, baixa diretividade das antenas. Embora já existam formas de aumentar a taxa de transmissão dos dados, é uma tarefa difícil alcançar uma relação sinal ruido suficiente para decodificar as imagens (KLEIN; HAWKINS; THORSEN, 2014).

Conforme a equação proposta por Harald T. Friis para o cálculo de rádio enlace no espaço livre

$$P_R(dBm) = P_T(dBm) + G_T(dBi) + G_R(dBi) - L_0(dB) \quad (1)$$

é possível aumentar a potência recebida simplesmente aumentando o ganho da antena de recepção, porém isso significaria uma antena de recepção maior, fazendo o custo crescer exponencialmente, o que vai de encontro com o propósito de *CubeSat*, de fazer missões com baixo orçamento e custo de infraestrutura.

Alternativamente, outras possíveis soluções incluem aumentar a cobertura global da estação terrestre, aumentando a janela de transmissão e aumentar o ganho da antena de transmissão, o que pode ser feito aumentando a sua diretividade com arranjos faseados por exemplo (KLEIN; HAWKINS; THORSEN, 2014).

Porém, todas essas soluções, propõe uma melhora apenas na taxa de transmissão de dados e não no aproveitamento de banda.

3.2 CLASSIFICAÇÃO DE IMAGENS

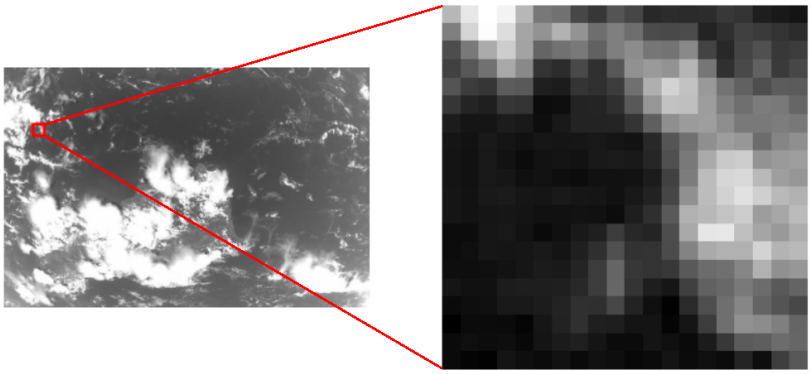
Abaixo serão abordados aspectos básicos e os métodos de processamento de imagens a serem utilizados na tarefa de classificação deste trabalho.

3.2.1 A Imagem Digital

Uma imagem digital pode ser definida como uma projeção em duas dimensões, de um cenário tridimensional, representada por uma função f(x, y), onde x e y indicam as coordenadas de cada elemento, chamados de *pixels*. Cada *pixel* por sua vez, possuirá um valor de intensidade discreto geralmente representado por 1 *byte* (entre [0, 255]). Para imagens coloridas, normalmente três canais estarão presentes, onde cada canal representa uma matriz bidimensional, com intensidades diferentes para a propriedade que o canal representa, por exemplo, em imagens do tipo RGB, o canal R (vermelho) indica a intensidade de vermelho em cada pixel, G de verde e B de azul. Existem imagens com somente um canal, que é o caso das imagens em escala de cinza, muito utilizadas no processamento de sinais, onde cada pixel varia entre o preto (menor intensidade) para o branco (maior intensidade) representando assim escalas de cinza (TYAGI, 2018).

Na figura 1, é ilustrado o caso em que a imagem possui apenas um canal em tons de cinza, se a quantidade de bits reservados por pixel mudar, é possível modificar a quantidade de tons presentes na imagem, se por exemplo ao invés de 8 *bits* fosse alocado somente 1 *bit* por *pixel*, o resultado seria apenas 2 cores, preto e banco dependendo se o valor for 0 ou 1, esse caso é chamado de imagem binária, também muito utilizada no processamento digital de imagens. Conforme a figura 2, uma maior quantidade de *bits* por *pixel* significa uma transição mais suave entre os tons da imagem.

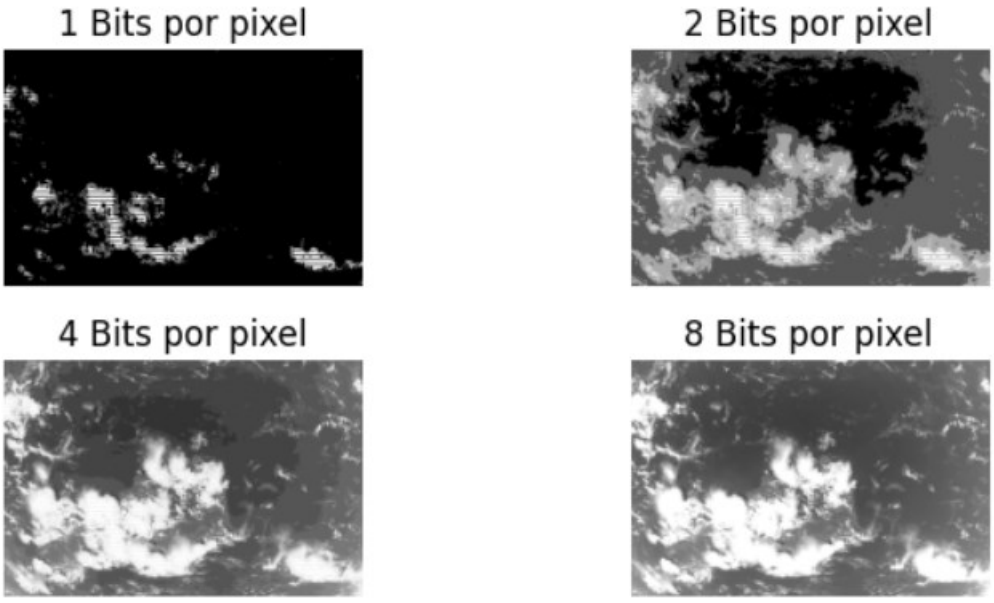
Figura 1 – Imagem Digital Quantizada



204	238	255	247	204	160	156	126	115	135	125	105	108	106	107	116	112	97	94	90
165	225	253	231	192	180	201	179	153	165	153	129	128	129	101	109	121	117	112	108
130	161	196	220	191	147	156	161	144	165	174	159	157	172	123	104	124	132	116	119
122	137	156	194	172	112	107	128	110	137	178	184	180	203	159	110	128	144	117	126
135	119	114	141	139	98	96	113	110	118	155	199	224	212	183	157	144	131	141	140
114	102	98	114	112	85	86	101	102	109	138	181	212	210	186	165	154	163	161	148
96	91	91	96	94	86	93	101	101	102	113	143	176	185	187	175	161	161	164	160
91	89	94	91	88	92	91	92	93	91	89	105	136	155	176	186	182	152	169	176
90	88	93	89	84	89	90	89	94	95	93	103	132	162	195	217	219	179	184	187
90	88	93	92	87	89	86	85	88	88	89	96	122	163	208	226	228	206	187	181
91	89	94	95	92	91	94	92	91	88	93	100	125	177	208	223	234	221	206	200
91	89	91	92	90	88	88	89	88	85	97	109	139	200	205	213	219	186	193	207
85	85	92	93	89	83	87	94	98	119	105	102	130	194	237	238	210	192	183	198
85	87	92	91	92	94	90	98	109	132	111	112	129	168	195	200	200	224	202	206
86	86	87	89	92	95	94	102	118	147	115	112	116	139	152	155	170	200	177	182
91	88	88	96	97	97	98	105	118	148	112	97	92	121	144	149	161	152	139	146
83	86	87	93	93	96	101	110	109	126	101	87	78	106	131	154	170	146	135	127
76	84	84	83	80	92	100	110	98	97	90	90	80	91	118	154	178	174	165	147
83	87	85	86	82	90	95	98	91	84	84	90	87	83	95	116	140	142	152	148
83	79	78	90	86	84	89	83	88	86	82	84	90	84	87	84	102	86	120	137

Fonte: elaborado pelo autor

Figura 2 – Imagem Digital com diferentes resoluções de cor



Fonte: elaborado pelo autor

3.2.2 Modelos de Cores

Modelos de cores são modelos matemáticos utilizados para representar informações relacionadas as cores, separando-as em componentes. Diferentes modelos são usados para diferentes tipos de aplicações, sendo necessário selecionar o melhor modelo para cada situação (KOLKUR et al., 2017). Abaixo serão discutidos apenas os modelos RGB e HSV, que serão utilizados no processamento das imagens, porém existem muitos outros que são uteis em outras tarefas de classificação

O RGB é normalmente o sistema de cor mais utilizado para a representação de cores em imagens digitais, sendo utilizado por computadores e monitores. No RGB, qualquer cor pode ser obtida misturando três cores primárias, o vermelho, verde e azul, então, dependendo da intensidade de cada uma dessas 3 componentes, as cores podem ser formadas (KOLKUR et al., 2017).

O sistema HSV, se aproxima mais de como as pessoas percebem as cores se comparado ao RGB. De forma similar ao RGB, as cores são compostas por 3 componentes, variando de 0 a 1 seus valores. Entre esses, o matiz, que controla o tipo da cor, variando entre todo o espectro desde o vermelho até o magenta retornando para o vermelho novamente, a saturação, que controla se a cor terá tons de cinza, ou se será “pura”, e o valor, que define o brilho da cor (KOLKUR et al., 2017).

Além disso, uma característica muito importante do sistema HSV é que o matiz é invariante a mudanças na iluminação, sendo assim muito útil para ser utilizado em problemas que a cor desempenha um papel fundamental no processamento da imagem (SZABOLCS SERGYAN, 2007).

Como a maioria das imagens são capturadas em RGB para trabalhar com HSV é necessário realizar uma conversão, que pode facilmente ser realizada através das equações (2, 3, 4) através das componentes da imagem RGB (SZABOLCS SERGYAN, 2007).

$$H = \begin{cases} indefinido & se\ \text{máx}(R,G,B) = \text{mín}(R,G,B) \\ \dfrac{60 \cdot (G-B)}{\text{máx}(R,G,B) - \text{mín}(R,G,B)} + 0 & se\ máx(R,G,B) = R\ e\ G \geq B \\ \dfrac{60 \cdot (G-B)}{\text{máx}(R,G,B) - \text{mín}(R,G,B)} + 360 & se\ máx(R,G,B) = R\ e\ G < B \\ \dfrac{60 \cdot (B-R)}{\text{máx}(R,G,B) - \text{mín}(R,G,B)} + 120 & se\ máx(R,G,B) = G \\ \dfrac{60 \cdot (R-G)}{\text{máx}(R,G,B) - \text{mín}(R,G,B)} + 240 & se\ máx(R,G,B) = B \end{cases} \quad (2)$$

$$S = \begin{cases} 0 & se\ \text{máx}(R,G,B) = 0 \\ 1 - \dfrac{\text{mín}(R,G,B)}{\text{máx}(R,G,B)} & se\ máx(R,G,B) > 0 \end{cases} \quad (3)$$

$$V = \text{máx}(R,G,B) \quad (4)$$

3.2.3 Processamento Digital de Imagens

Os principais métodos convencionais utilizados no processamento digital de imagens, assim como a principal biblioteca disponível para as operações necessárias são apresentadas nos tópicos a seguir.

3.2.3.1 Operador de Sobel

O operador de Sobel, proposto por Irwin Sobel e Gary M. Feldman em 1968 (SOBEL, 2014), é uma das técnicas mais utilizadas para a detecção de bordas em imagens digitais. Esse operador calcula uma aproximação da magnitude do gradiente de intensidade em cada pixel da imagem, destacando as áreas onde ocorrem variações bruscas de intensidade (que tipicamente correspondem às bordas). Para isso, são aplicados dois filtros de convolução 3x3 na imagem, um para calcular o gradiente na direção horizontal (5) e outro na direção vertical (6).

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * Imagem \quad (5)$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * Imagem \quad (6)$$

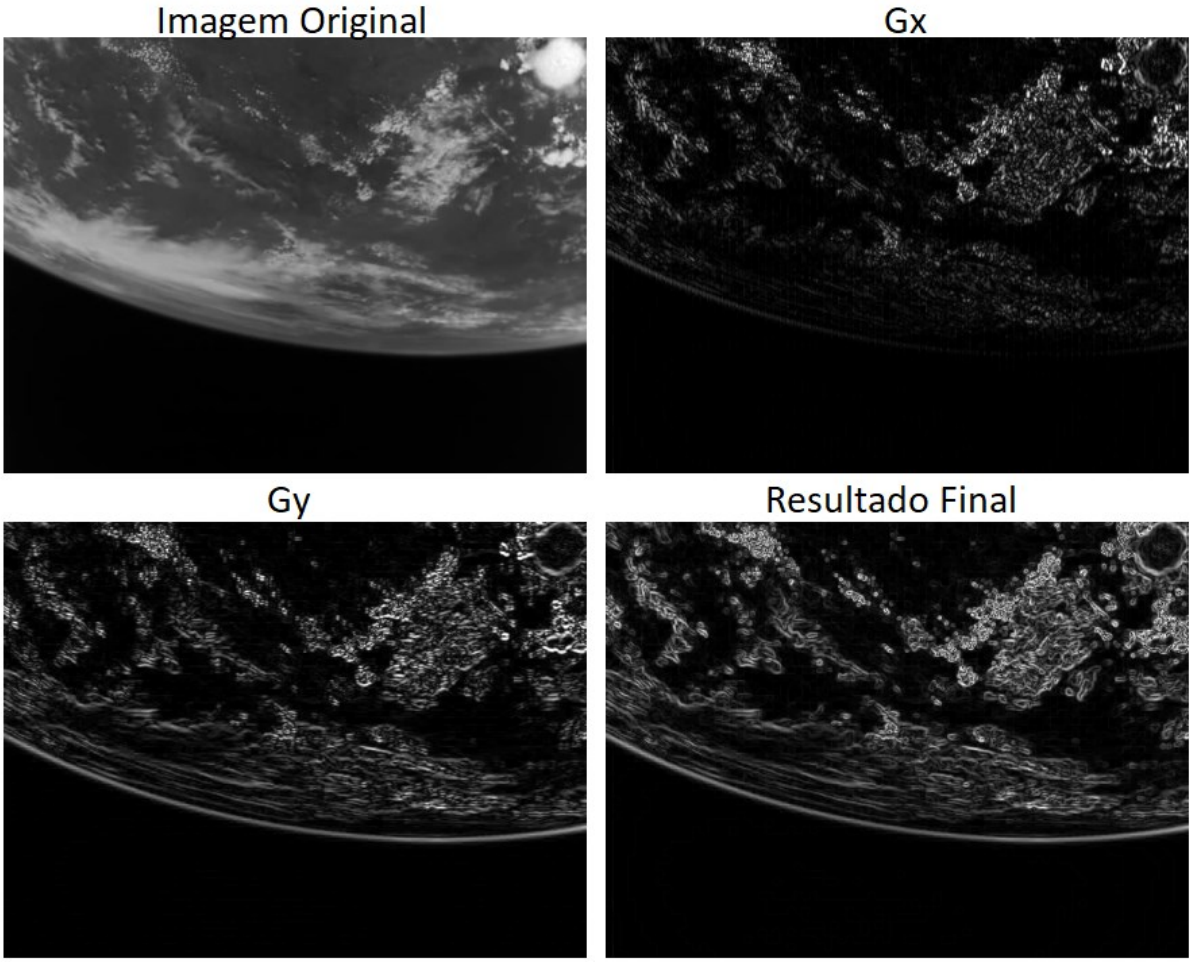
A combinação dos dois gradientes permite estimar a direção e a intensidade da borda (7,8). Essa abordagem é amplamente utilizada devido à sua simplicidade e eficiência computacional (GONZALEZ et al., 2014).

$$G = \sqrt{G_x^2 + G_y^2} \tag{7}$$

$$\theta = tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{8}$$

Na figura 3 é possível ver o resultado de todas as transformações aplicadas na imagem, a seguir, apesar do método ser computacionalmente pouco custoso, algumas outras técnicas de detecção de bordas podem apresentar um desempenho melhor na tarefa de detecção.

Figura 3 – Resultado do operador de Sobel



Fonte: elaborado pelo autor

3.2.3.2 Canny Edge Detector

O detector de bordas de Canny, é um operador proposto por John F. Canny que utiliza alguns passos para obter as bordas de uma imagem. Como o operador é bastante sensível a ruídos, o primeiro passo é suavizar a imagem com um filtro gaussiano. Normalmente é utilizada uma função gaussiana bidimensional (9) para isto (SONG; ZHANG; LIU, 2017).

$$G(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\pi\sigma^2}\right) \tag{9}$$

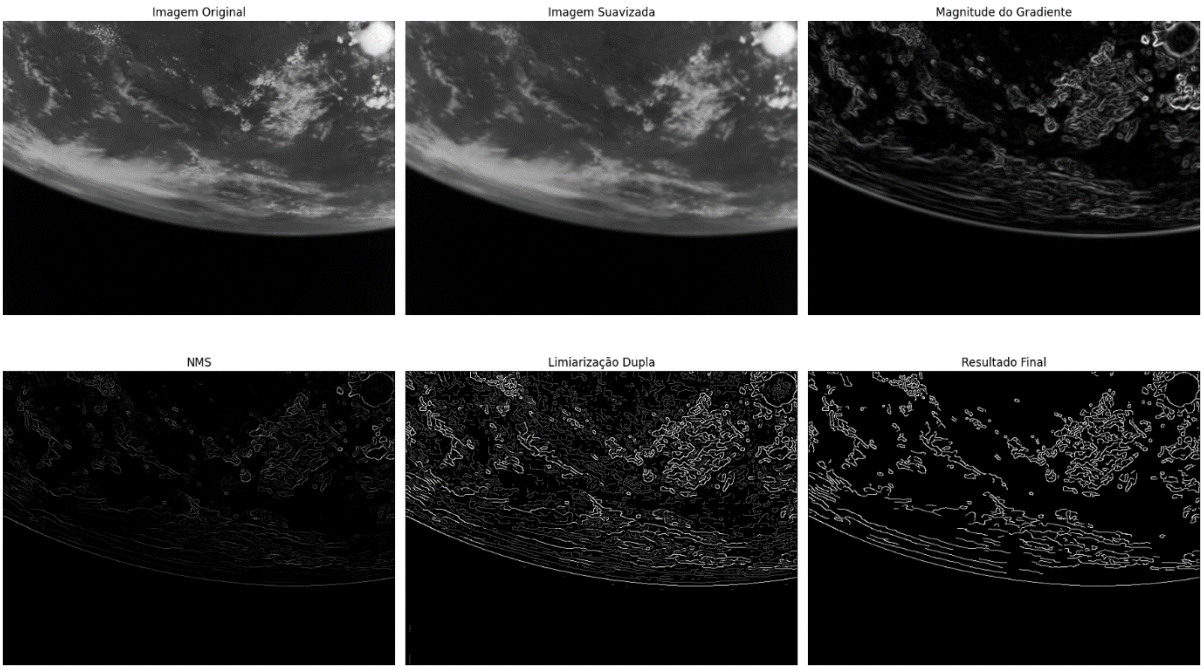
Com a suavização realizada, os gradientes de magnitude e de direção são calculados, (7) e (8) respectivamente, onde G_x e G_y correspondem a convolução da imagem com o operador de Sobel na direção horizontal e vertical respectivamente (OPENCV, [s.d.]).

A seguir, é feito um processo de “afinamento”, das bordas detectadas, chamado de supressão não-máxima (NMS), com o objetivo de detectar *pixels* que não fazem parte da borda. O NMS, para cada *pixel*, usa o gradiente da direção para encontrar os dois *pixels* vizinhos a serem comparados, sendo então interpolados. Se o resultado for maior que o *pixel* analisado, este será suprimido, caso contrário é considerado como uma possível borda (RONG et al., 2014).

Finalmente, na última etapa é feita uma limiarização dupla, onde os *pixels* com magnitude do gradiente maior que o limiar superior são considerados como pontos de borda, os *pixels* com magnitude menor que o limiar inferior são suprimidos, e os *pixels* entre os dois limiares, por meio de uma limiarização por histerese são verificados para ver se possuem algum vizinho considerado como ponto de borda, se possuir, o pixel automaticamente é considerado um ponto de borda (SONG; ZHANG; LIU, 2017).

Todo o processo passo a passo do algoritmo pode ser observado na figura 4.

Figura 4 – Detector de Bordas de Canny



Fonte: elaborado pelo autor

3.2.3.3 Limiarização

Uma área do processamento de imagens muito utilizada na segmentação de imagens é a limiarização, que consiste em estabelecer limiares para os valores dos *pixels* e separar regiões baseado nesses limiares.

3.2.3.3.1 Limiar fixo para tons de cinza

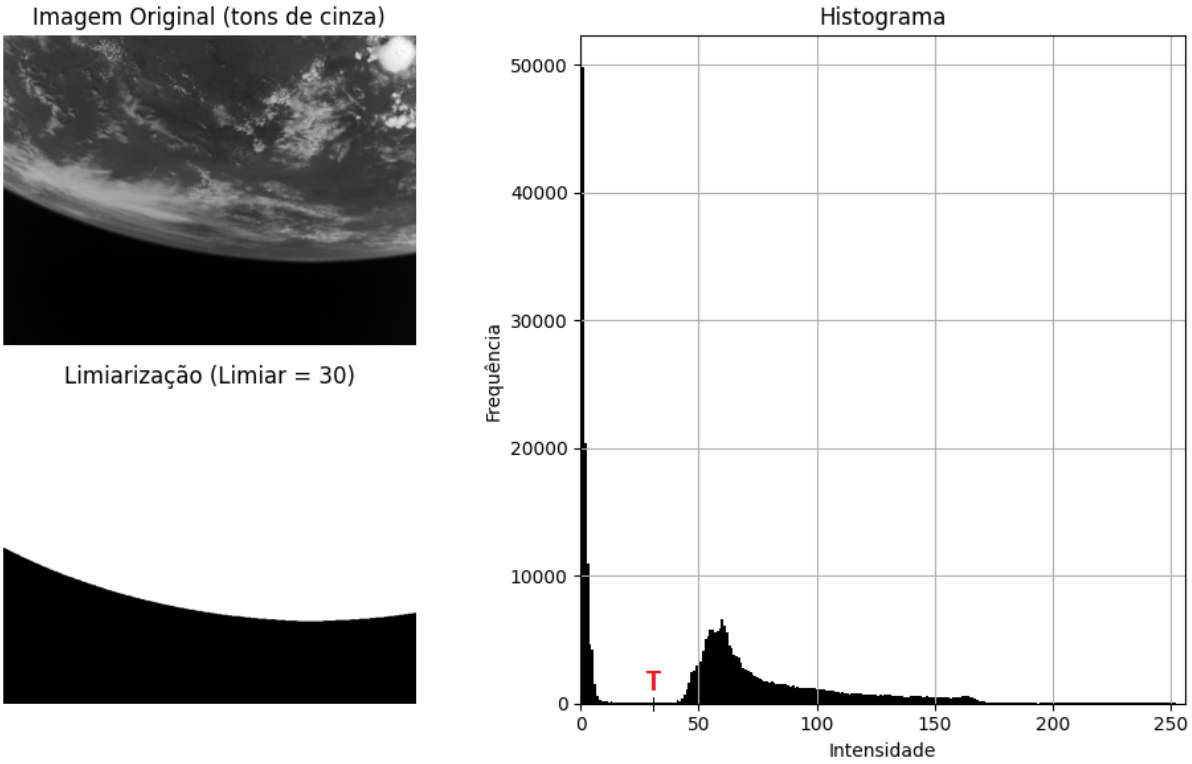
A técnica de limiarização de uma imagem mais simples disponível, se trata de realizar uma análise no histograma de intensidade dos *pixels*, de forma a definir um limiar que separa a imagem em duas regiões. O resultado dessa separação seria uma imagem binária com intensidade 1 ou 0 (255 ou 0) a depender de qual grupo a imagem pertence. Para essa técnica ser efetiva, é necessário que o fundo e o objeto tenham duas intensidades agrupadas de forma dominante (GONZALEZ; WOODS; MASTERS, 2009).

A modelagem para isso pode ser descrita por (10) (GONZALEZ; WOODS; MASTERS, 2009) com a figura 5 mostrando um exemplo de aplicação onde através

do histograma da imagem, foi definido um limiar que separa o fundo do objeto em análise.

$$g(x,y) = \begin{cases} 1 & se\ f(x,y) > T \\ 0 & se\ f(x,y) \leq T \end{cases} \quad (10)$$

Figura 5 – Limiar Fixo para tons de cinza



Fonte: elaborado pelo autor

É possível ainda estender essa técnica aumentando o número de limiares, e atribuindo valores para estes intervalos (11), sendo possível segmentar mais de uma seção da imagem (GONZALEZ; WOODS; MASTERS, 2009).

$$g(x,y) = \begin{cases} a & se\ f(x,y) > T_2 \\ b & se\ T_1 < f(x,y) \leq T_2 \\ c & se\ f(x,y) \leq T_1 \end{cases} \quad (11)$$

3.2.3.3.2 Limiar fixo para detecção de cores

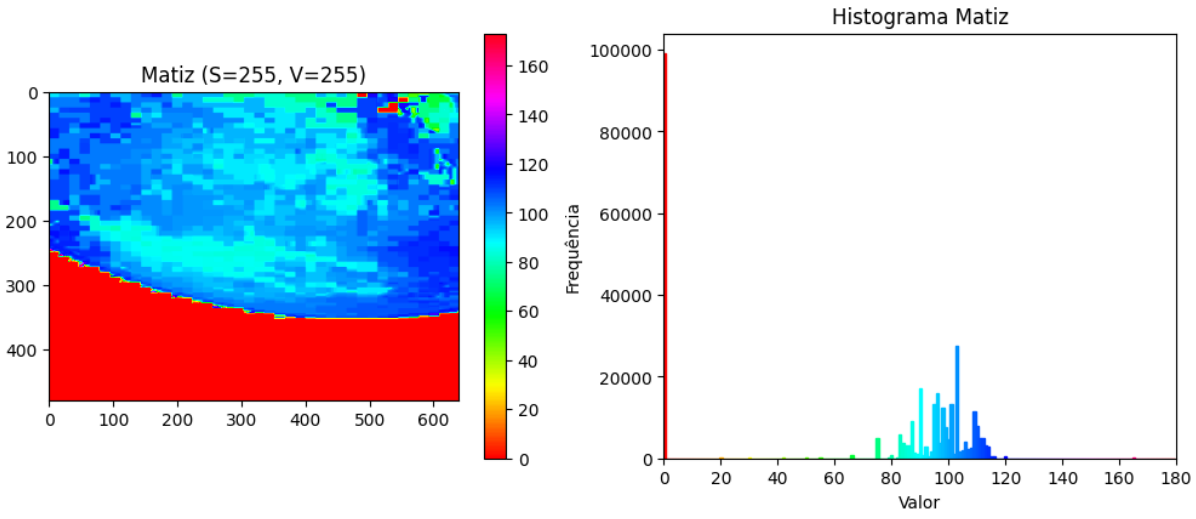
Grande parte das técnicas utilizadas na segmentação de imagens em tons de cinza podem ser estendidas para imagens coloridas, se a análise for feita para cada

componente da imagem individualmente, os resultados podem ser combinados para assim obter uma segmentação final. Um problema a ser solucionado, é que cada modelo de cor possui as suas vantagens e desvantagens, com nenhum modelo claramente se sobressaindo sobre o outro, sendo assim é necessária a escolha cuidadosa de um modelo para cada caso (CHENG et al., 2001).

Apesar do RGB ser o modelo de cor mais utilizado tanto na aquisição de imagens, como na exibição delas, a aplicação deste modelo na segmentação de imagens via cores é muito restrita, isso considerando que uma simples mudança na intensidade de uma cor (efeitos de iluminação), irá fazer todas as três componentes da cor mudarem também, sendo assim muito difícil encontrar intervalos de cores que delimitem um objeto justamente por essa interdependência entre os três canais (CHENG et al., 2001).

Como a no sistema HSV o matiz é invariante a saturação (SZABOLCS SERGYAN, 2007), a iluminação da imagem vai ter uma menor influencia na detecção das cores. Na figura 6 é possível observar através da projeção do matiz da imagem, e do histograma do matiz, que com apenas um canal é possível inferir a cor “pura” presente na imagem (sem adição de branco ou preto), sendo mais fácil definir os limiares de detecção das cores devido à baixa variação do matiz com a iluminação da imagem. Devido à essas características, esse será o modelo de cor utilizado para a limiarização de cores das imagens.

Figura 6 – Análise do matiz de uma imagem



Fonte: elaborado pelo autor

3.2.3.3.3 Limiarização adaptativa através do método de Otsu

Em 1979, Nobuyuki Otsu propôs um método para limiarizar imagens de forma automática, ou seja, sem utilizar nenhum parâmetro, através da maximização da variância entre classes. Assumindo que as intensidades dos *pixels* são diferentes para diferentes classes, o algoritmo retorna o limiar que melhor separa essas classes, baseado em um histograma de tons de cinza (OTSU, 1979).

Conforme formulado por Otsu, dado os valores inteiros representando as L diferentes intensidades dos *pixels* da imagem $\{0,1,2,\dots,L-1\}$, n_i representando a quantidade de *pixels* com intensidade i e N o somatório de *pixels* total da imagem, é possível obter a distribuição de probabilidade dos tons de cinza da imagem (12).

$$p(i) = \frac{n_i}{N} \tag{12}$$

Supondo que seja estabelecido um limiar de valor t, a classe 1 conterá pixels com intensidades de 0 a t, e a classe 2 de $t+1$ a $L-1$. Assim, a probabilidade acumulada de um *pixel* pertencer a classe 1 ou a classe 2 é dada por (13) e (14)

$$\omega_1(t) = \sum_{i=0}^{t} p(i) \tag{13}$$

$$\omega_2(t) = \sum_{i=t+1}^{L-1} p(i) \tag{14}$$

As médias de intensidade das classes 1 e 2, dadas por (15) e (16) respectivamente, e a média global (17) podem ser utilizadas para calcular a dispersão das intensidades de cada classe em relação a média global, chamada de variância entre classes (18).

$$\mu_1(t) = \frac{\sum_{i=0}^{t} i \cdot p(i)}{\omega_1(t)} \tag{15}$$

$$\mu_2(t) = \frac{\sum_{i=t+1}^{L-1} i \cdot p(i)}{\omega_2(t)} \tag{16}$$

$$\mu = \sum_{i=0}^{L-1} i \cdot p(i) \tag{17}$$

$$\sigma_b^2(t) = \omega_1(t) \cdot (\mu_1(t) - \mu)^2 + \omega_2(t) \cdot (\mu_2(t) - \mu)^2 \tag{18}$$

Como o objetivo do método de Otsu é maximizar a variância entre classes para encontrar o limiar ótimo, o problema agora se resume como um problema de otimização (19) com o limiar ótimo dado como t^*.

$$\sigma_b^2(t^*) = \max_{0 \le t \le L-1} \sigma_b^2(t) \tag{19}$$

3.2.3.4 Análise de texturas

Existem diferentes técnicas para a análise de texturas, entre elas, a LBP (*Local Binary Pattern*) se destaca devido a sua simplicidade computacional e principalmente robustez na análise para alterações causadas na imagem em tons de cinza (PIETIKÄINEN, 2010).

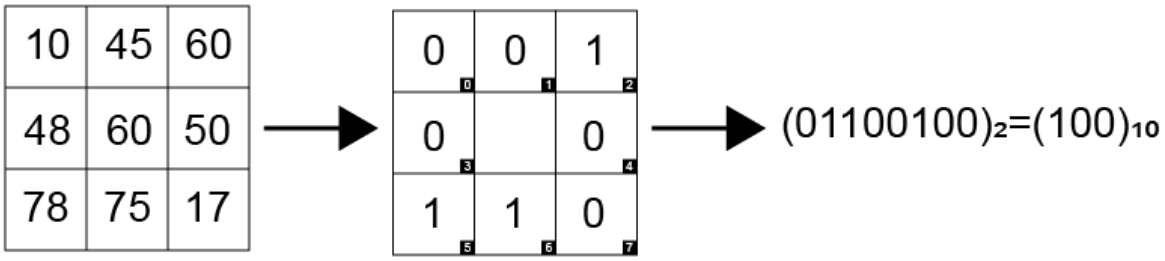
O operador original LBP, é calculado analisando todos os pixels da imagem, e seus 8 vizinhos, se a intensidade destes for maior ou igual ao *pixel* central, o *pixel* do vizinho assume o valor binário 1, caso contrário, assume o valor 0. Após isso, é retornado um valor binário com 8 bits, e o valor decimal resultante desse binário será o valor da unidade de textura (OJALA; PIETIKÄINEN; HARWOOD, 1996).

A figura 7 mostra um exemplo de uma janela 3x3 *pixels* de uma imagem sendo utilizada para o cálculo da unidade de textura, onde o valor final, em decimal, pode ser dado por (20) e (21) com g_p representando a intensidade do *pixel* vizinho na posição p, g_c a intensidade do *pixel* central e P o número de vizinhos que no caso de uma janela 3x3, são 8.

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c) \cdot 2^p \tag{20}$$

$$s(x) = \begin{cases} 1, & x \ge 0 \\ 0, & x < 0 \end{cases} \tag{21}$$

Figura 7 – Processo da LBP



Fonte: elaborado pelo autor

Existem algumas extensões para o método original de cálculo da LBP, com a LBP uniforme se destacando. Essa técnica tem como vantagem retornar um descritor que é invariante à rotação. O método é inspirado na ideia no fato de que, experimentalmente, verificou-se que cerca de 90% dos padrões encontrados nas LBP com 8 vizinhos são padrões uniformes. Padrões podem ser definidos como uniformes se o número binário encontrado no final da binarização possuir no máximo duas transições de 0 para 1 ou 1 para 0. Por exemplo, padrões como 11010100 (5 transições), 01001000 (4 transições) são considerados não uniformes, enquanto padrões como 00011000 (2 transições) e 00000000 (0 transições) são considerados uniformes. Dessa forma, o histograma resultante possuirá bem menos classes se comparado ao método tradicional, já que seria uma para cada padrão uniforme mais uma classe destinada a todos os padrões que não são uniformes, sendo computacionalmente mais eficiente (PIETIKÄINEN, 2010).

3.2.4 Aprendizado de Máquina

O aprendizado de máquina, ou *machine learning*, pode ser definido como a ciência que foca no desenvolvimento de algoritmos e modelos que permitem que os computadores aprendam a partir de dados. Em vez de serem explicitamente programados para realizar uma tarefa específica, esses sistemas são capazes de identificar padrões e tomar decisões baseadas em informações previamente adquiridas (GÉRON, 2019).

Um exemplo de aplicação seriam os filtros de spam de e-mails, que, em um contexto de programação tradicional, o fluxo de trabalho normalmente envolve a análise do problema e a codificação das regras que regem a sua solução, nesse caso,

critérios como palavras-chave poderiam ser definidas para categorizar o email como spam ou não. Em contrapartida, na abordagem de aprendizado de máquina, um modelo seria desenvolvido para executar uma tarefa específica (classificação de emails) com base em um conjunto de dados representativo do problema. Este modelo é capaz de identificar padrões relevantes que podem ser indicadores importantes se um email é spam, sem a necessidade de pré-definir regras (GÉRON, 2019).

Ainda no contexto do exemplo, alterações nos padrões dos emails spam, como variação nas palavras utilizadas, pode resultar em falha no código, classificando erroneamente emails. Em contrapartida, um modelo de aprendizado de máquina pode ser continuamente aprimorado através da realimentação com novos dados, permitindo uma adaptação do modelo a mudanças (GÉRON, 2019).

3.2.4.1 Aprendizado Supervisionado

Os algoritmos de aprendizado supervisionado, consistem no treinamento de modelos com base em dados rotulados, onde os exemplos de treinamento irão possuir as *features* (características que descrevem o dado de entrada) assim como um rótulo ou valor de saída associado a esse exemplo (GOODFELLOW; BENGIO; COURVILLE, 2016).

3.2.4.2 Aprendizado Não Supervisionado

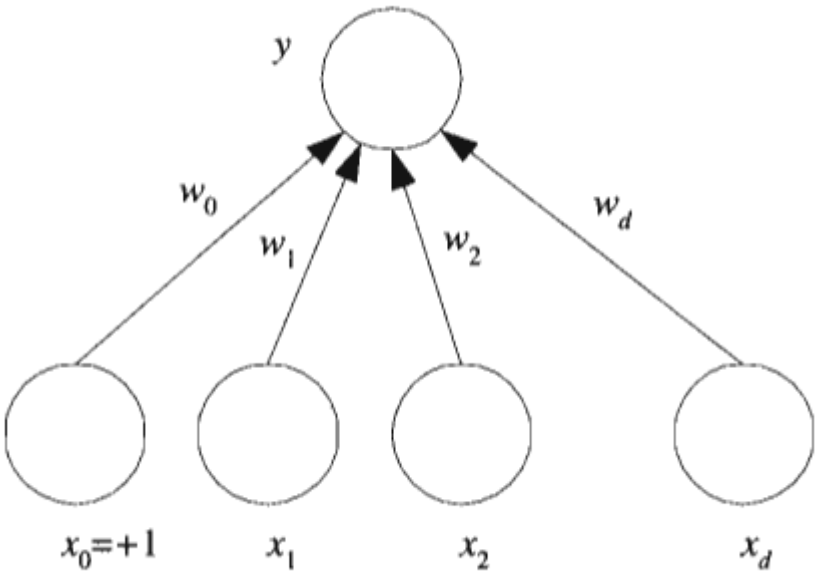
Para os algoritmos de treinamento de modelos de aprendizado não supervisionado, são fornecidos exemplos de treinamento que contém somente as *features* dos exemplos, ou seja, os dados não são rotulados ou não possuem valores de saída. O objetivo é encontrar propriedades que possam ser uteis nesses dados, como por exemplo dividi-los em *clusters* (GOODFELLOW; BENGIO; COURVILLE, 2016).

3.2.4.3 Redes Neurais Artificiais (RNA)

As Redes Neurais Artificiais (RNA) têm sua origem inspirada no funcionamento do cérebro humano, onde os neurônios são representados pelos chamados

perceptrons, um elemento básico de processamento que tem como entrada dados externos ou saídas de outros *perceptrons* conectados. Cada entrada tem associada a si um peso, que compõe a saída do *perceptron* após ser realizada uma soma ponderada entre todas as entradas. Na figura 8 está ilustrada a implementação mais simples possível de um *perceptron*, com y representando a saída calculada por (22), w_j os pesos das entradas, e x_j as entradas. Ainda, é adicionado um valor de viés ou *bias* (w_0), para ser possível generalizar mais o modelo (ALPAYDIN, 2020).

Figura 8 – Estrutura básica de um perceptron



Fonte: (ALPAYDIN, 2020)

$$y = f\left(\sum_{j=1}^{d} w_j x_j + w_0\right) \tag{22}$$

Após realizada a soma ponderada, o resultado é utilizado como argumento para uma função não linear, conhecida como função de ativação. Algumas funções de ativação mais utilizadas incluem o ReLU e a função sigmoide (WATT; REZA BORHANI; AGGELOS KONSTANTINOS KATSAGGELOS, 2016).

Se a ideia de unidades básicas de processamento for expandida para várias camadas, é possível construir estruturas capazes de resolver problemas mais complexos, as chamadas MLPs (*Multilayer Perceptron*), constituídas pela camada de entrada, camadas intermediarias que são chamadas de camadas ocultas e uma camada de saída. Para os casos em que uma RNA possui várias camadas ocultas, essa rede é comumente chamada de rede neural profunda (GÉRON, 2019).

3.2.4.4 Redes Neurais Convolucionais (CNN)

As redes neurais convolucionais (CNNs), são redes neurais especializadas para lidar com dados que apresentem estrutura de grade, como imagens por exemplo. Essa estrutura foi popularizada por LeCun et al. (1989) sendo amplamente utilizada para tarefas de reconhecimento de imagem. Como o próprio nome diz, a rede utiliza uma operação chamada de convolução ao invés da multiplicação de matrizes convencional utilizada em RNAs tradicionais (GOODFELLOW; BENGIO; COURVILLE, 2016).

Diferente das redes neurais convencionais, que analisam cada *pixel* de forma individual, ignorando dessa forma uma propriedade importante das imagens que é a correlação forte de pixels próximos, as CNN se aproveitam dessa propriedade. Para isso, são utilizadas as chamadas camadas convolucionais, onde cada neurônio (unidade convolucional), se conecta a um campo receptivo (pequena região de entrada), gerando na saída um mapa de ativação, capaz de detectar padrões em diferentes regiões da imagem (BISHOP, 2006).

As camadas das CNNs incluem camadas de convolução, camadas de *pooling* e camadas totalmente conectadas.

3.2.4.4.1 Camadas Convolucionais

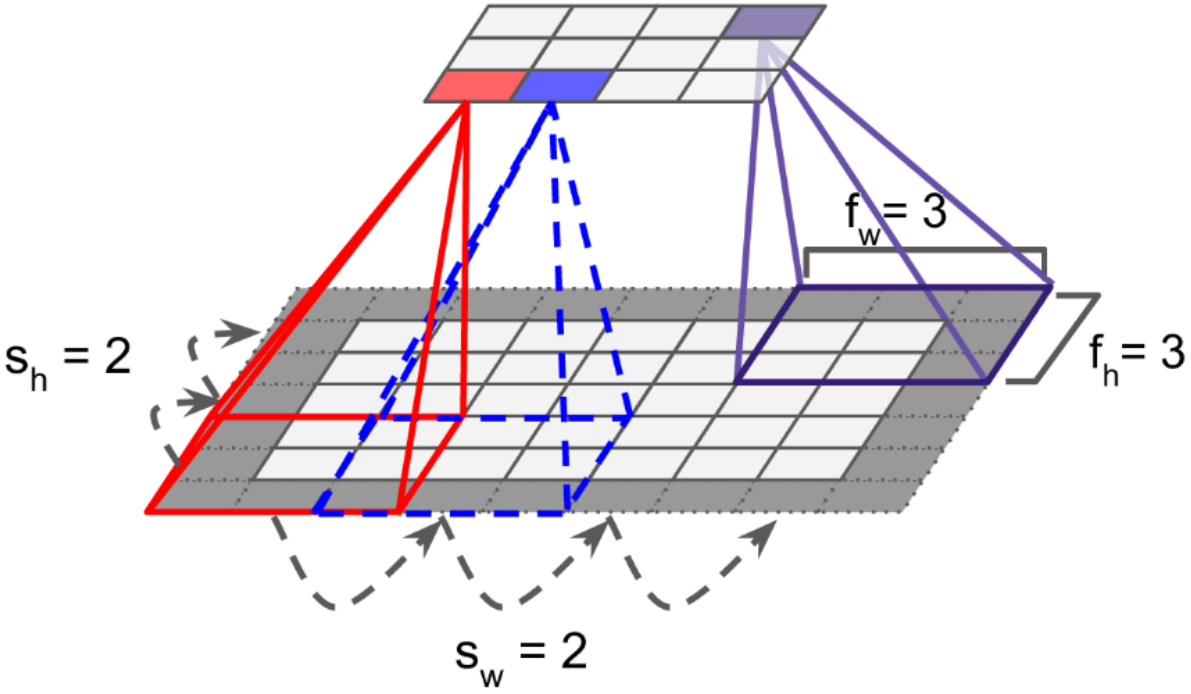
A camada convolucional é um dos blocos mais importante para o funcionamento das CNNs, sendo composta por um conjunto de filtros, também chamados de *kernels*, aplicados na entrada para extrair características das imagens. Para isso, os filtros são deslizados pela entrada de forma a multiplicar ponto a ponto ambas as matrizes, o resultado disso é chamado de mapa de ativação ou *feature map*. Os pesos dos filtros inicialmente são randomizados para conforme o treinamento, serem ajustados (TAYE, 2023).

Nos *pixels* das bordas da imagem, um problema surge, o filtro não é capaz de levar em conta alguns dos *pixels* vizinhos pelo fato de não existirem vizinhos nesse caso, para contornar esse problema, uma solução possível seria considerar o valor desses *pixels* como 0, técnica essa chamada de *zero padding*. Uma outra característica muito importante que é possível ser ajustada, é chamada de *stride*, que seria o passo em que o filtro percorre a imagem, ou seja, o número de *pixels* que são "saltados" durante a convolução. Essa operação é muito importante pois conforme o *stride* é aumentado, a saída da convolução terá uma redução no seu tamanho com o seu tamanho sendo dado por (23). Na *figura 9* é possível visualizar uma convolução em 2 dimensões considerando *zero padding*, e um *stride* arbitrário (MURPHY, 2022).

$$O = \left(\frac{x_h + 2p_h - f_h + s_h}{s_h}\right) \times \left(\frac{x_w + 2p_w - f_w + s_w}{s_w}\right) \tag{23}$$

Em (23) a entrada possui dimensões $x_h \times x_w$, o filtro $f_h \times f_w$ com um valor de *stride* na vertical de s_h e s_w na horizontal e zero padding com valor de p_h e p_w de cada lado.

Figura 9 – Convolução de uma imagem com stride 2 e um filtro 3x3



Fonte: Adaptado de (GÉRON, 2019)

3.2.4.4.2 Camadas de Pooling

A camada de *pooling*, tem como objetivo diminuir a dimensionalidade dos *feature maps*, enquanto tenta manter as suas características mais importantes, isso é feito para diminuir o uso de memória, o esforço computacional e o número de parâmetros do modelo a ser treinado. Da mesma forma que as camadas convolucionais, é necessário definir um *padding* e um *stride*, assim como uma função para agregar os *pixels*. Diferentemente da convolução, o operador de *pooling* não possuirá pesos servindo apenas para agregar os *pixels*, duas das funções mais utilizadas são o *max pooling*, que escolhe o pixel com maior intensidade dentro da região analisada, e o *average pooling*, que computa a média dos pixels da região (GÉRON, 2019).

3.2.4.4.3 Camadas totalmente conectadas

As camadas totalmente conectadas (FC) são responsáveis por compilar todas as características que foram extraídas nas outras camadas, para isso, o *feature map*

é primeiramente achatado em um vetor unidimensional para assim ser conectado as camadas totalmente conectadas, que tem consistem basicamente na mesma ideia das MLPs. Caso a última camada da CNN seja uma FC, esta tipicamente será responsável por atribuir uma classe para a imagem através da sua função de ativação (YAMASHITA et al., 2018).

3.2.4.5 Conjuntos de Dados

Durante o processo de treinamento de um modelo de classificação, normalmente o banco de dados ou *dataset*, é dividido em 3 conjuntos. O conjunto de treinamento tem como objetivo treinar o modelo, ou seja, ajustar os parâmetros do modelo, enquanto o conjunto de validação avalia o modelo durante o treinamento, como o conjunto de validação não é visto durante a etapa de treinamento, serve como uma métrica para determinar qual o modelo com menor erro de validação. Por fim, o conjunto de testes é utilizado para testar o modelo após o seu treinamento já ter sido finalizado, para a estimativa da performance real do modelo ser mais fiel o possível, esse conjunto necessita ser estatisticamente independente, sem ter seus dados vazados para o treinamento (XU; GOODACRE, 2018).

Entre os métodos mais utilizados na divisão entre dados de treinamento e validação, se destacam o *hold-out validation*, que consiste em dividir de forma fixa os dados em validação e treinamento, ou seja, o conjunto de validação permanece o mesmo até o final do treino e a *k-fold cross-validation* que divide todo o conjunto em k grupos, e em cada iteração, um grupo diferente é selecionado como conjunto de validação, e o restante se torna o conjunto de treinamento, essa técnica tem como principal vantagem que nenhum dado será ignorado durante a etapa de treinamento, tendo um aproveitamento maior do *dataset* (KOHAVI, 1995).

3.2.4.5.1 Data Augmentation

Existem algumas técnicas para tentar diminuir o *overfitting*[2] em modelos treinados, assim como melhorar o aproveitamento do *dataset* para quantidades

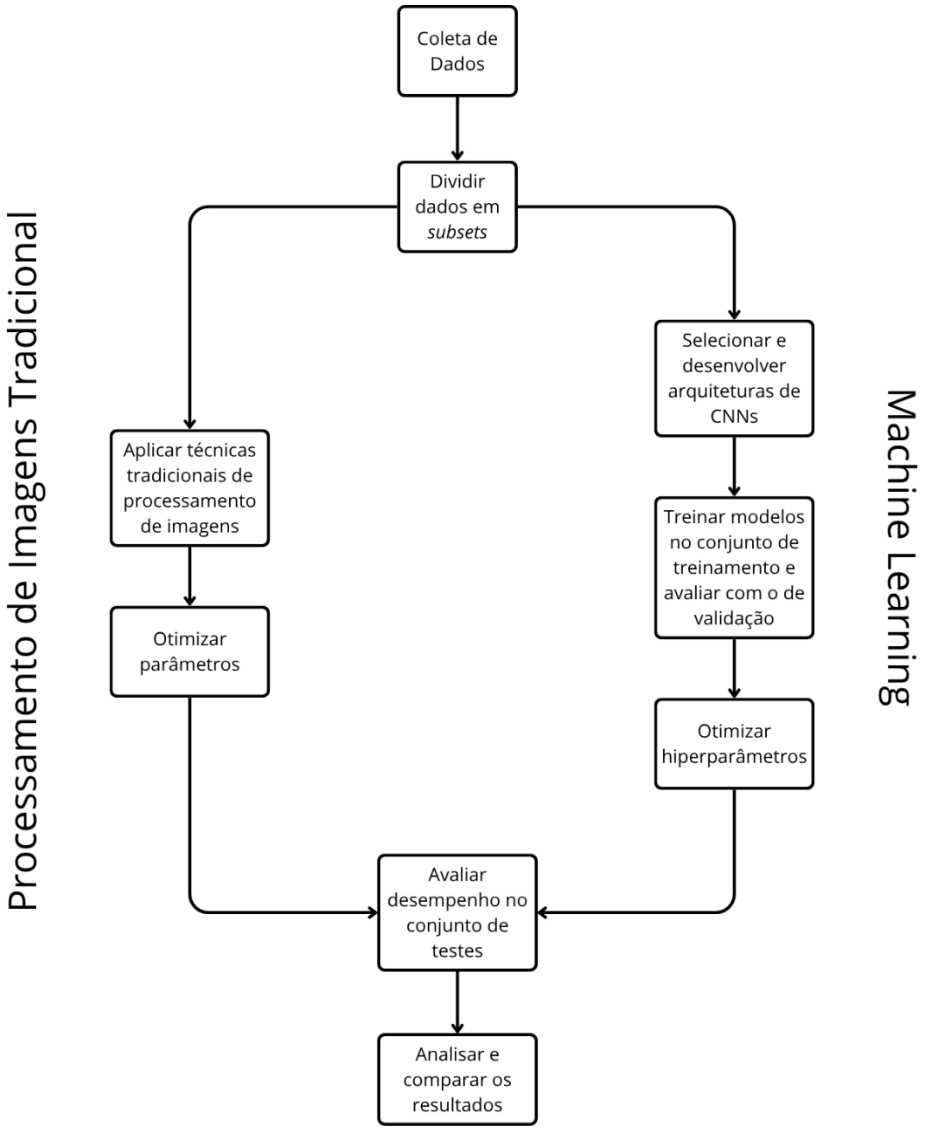
[2] Quando o modelo treinado se ajusta muito bem aos dados de treinamento, porém se prova ineficaz em previsões com novos dados.

pequenas de dados. Entre as técnicas disponíveis, destaca-se o *data augmentation*, que, no contexto de imagens, utiliza diversas transformações para diversificar o conjunto de imagens existente. Transformações amplamente aceitas na literatura incluem cortes, translações, rotações, alterações na paleta de cores e distorções (PEREZ; WANG, 2017).

4 METODOLOGIA

A metodologia de trabalho do projeto pode ser dividida em algumas etapas, conforme a figura 10. Primeiramente o banco de imagens a ser utilizado para as simulações é adquirido, para posteriormente ser dividido em subconjuntos de treinamento, validação e testes. Após, são desenvolvidos os experimentos utilizando técnicas de processamento digital de imagens tradicionais, assim como os modelos de aprendizado de máquina. Ambos são avaliados utilizando o subconjunto de testes, para evitar o *overfitting* no caso dos modelos de machine learning, e manter a mesma métrica de avaliação entre ambos.

Figura 10 – Metodologia de trabalho



Fonte: elaborado pelo autor

4.1 AQUISIÇÃO DO BANCO DE DADOS

A aquisição do banco de imagens de missões de *CubeSat* é uma tarefa muito difícil, isto devido à falta de disponibilidade de imagens deste domínio em específico na *internet*, sendo assim, uma alternativa para essa escassez seria a busca de imagens de outros tipos de satélites similares.

Em (MASKEY, 2020), o autor buscando resolver um problema muito similar ao deste trabalho, criou um banco de dados de imagens 100x100 separadas em imagens ruins e boas (seção 3.2) de satélites, contendo imagens da ISS, de balões atmosféricos, do satélite Sentinel 3A disponibilizado pela ESA e de foguetes amadores. Após isso, o autor aplicou *data augmentation*, para expandir o banco de dados, resultando em 30.000 imagens “boas” e 30.000 “ruins”.

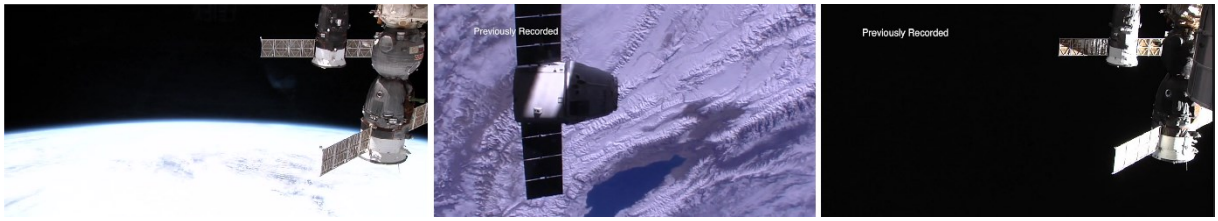
Além deste banco de imagens, foram adicionadas mais algumas imagens retiradas de vídeos transmitidos pela ISS, por meio da extração de seus *frames*, essas imagens disponibilizadas publicamente via o Kaggle [27], foram então filtradas para remover imagens que possam conter textos, e objetos que possam confundir o modelo, como por exemplo partes da ISS a mostra (figura 11).

Ademais, foram adicionadas algumas imagens também do catálogo *NASA Visible Earth* o qual contêm diversas imagens e animações referentes ao planeta, provindas de mais de 100 satélites. Para isso, foi utilizada a biblioteca *Beautiful Soup* do *Python*, com o objetivo de fazer uma *web scraping* e retornar imagens para o banco de dados.

Por fim, foram adquiridas imagens obtidas pelos *CubeSat*, que por somarem uma menor parcela serão utilizadas apenas no teste posteriormente. As imagens foram obtidas a partir da internet, ao procurar por imagens especificas neste domínio no google imagens, assim como através das imagens disponibilizadas pelo projeto BIRDS liderado pela universidade Japonesa *Kyutech* [28].

Na tabela 1 estão dispostas as fontes para aquisição dos dados, assim como a quantidade de imagens disponíveis após as filtragens serem feitas.

Figura 11 – Exemplos de imagens da ISS a serem filtradas



Fonte: elaborado pelo autor

Tabela 1 – Fontes de aquisição de dados

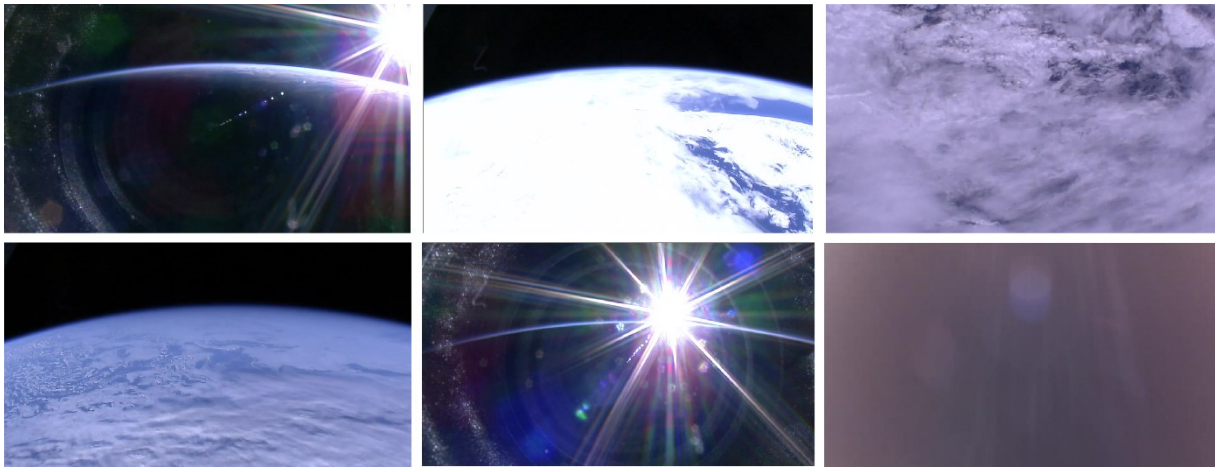
Fonte	**Quantidade**
(MASKEY, 2020)	60.000
ISS	6.380
NASA Visible Earth	667
BIRDS-3	156
BIRDS-4	78
Google imagens	26

Fonte: elaborado pelo autor

4.2 DIVISÃO DOS DADOS EM SUBCONJUNTOS

A fim de realizar a divisão em subconjuntos, primeiramente é necessário classificar as imagens entre as classes disponíveis, neste caso, entre ruim e boa. Foram usados diversos critérios para classificar uma imagem como boa, se destacando que, a imagem não deve ser de um espaço vazio, deve ser do horizonte ou com a terra completa no campo de visão, a saturação deve estar apropriada e a imagem não pode estar completamente coberta por nuvens. É importante ressaltar, que apesar de o banco de dados utilizado por (MASKEY, 2020), já possuir as imagens classificadas, algumas imagens não seguem os critérios estabelecidos, por isso foi feita uma reclassificação para atender as especificações do projeto, além da classificação feita nos novos dados adicionados.

Figura 12 – Exemplos de imagens classificadas como ruins



Fonte: elaborado pelo autor

Como a grande maioria das imagens de satélite adquiridas não são especificas de *CubeSat* (vide tabela 1), um problema que pode ocorrer é o de data shift[3]. Apesar de todas as imagens corresponderem a imagens do espaço, existirá uma diferença entre imagens retiradas, por exemplo, de satélites de grande porte, ou da ISS, e imagens retiradas por *CubeSat* com resolução limitadas, vibrações maiores e diferenças de orbita. Essa diferença na hora de implementar o modelo pode causar confusões no classificador, para minimizar esse efeito, a alternativa utilizada será de utilizar a grande maioria das imagens retiradas por *CubeSat*, no conjunto de testes, o que garantirá que não haja vazamento de dados para o treinamento, assim como providenciará uma métrica mais realista para o desempenho real do modelo.

Devido a dependência das técnicas de processamento de imagens tradicionais em parâmetros ajustados manualmente, isto é, que não são ajustados por uma função de perda, estes parâmetros serão ajustados especificamente para maximizar o desempenho no conjunto de testes, já que uma generalização maior neste caso não seria necessária devido a especificidade do problema de classificação limitado a imagens de *CubeSat*.

O conjunto de validação será definido usando o método de validação *hold-out*, com o conjunto de validação representando 20% (utilizando divisão estratificada) das imagens de treinamento. Já para o conjunto de testes, as imagens especificas de

[3] Data shift se refere a diferença entre os dados utilizados para o treinamento, e os dados utilizados durante a implantação do modelo de classificação em si.

CubeSat são reservadas para o teste do modelo treinado. A quantidade de imagens disponível ao final da divisão pode ser observada na tabela 2.

Tabela 2 – Subconjuntos do banco de dados

Subconjunto	**Bom**	**Ruim**
Treinamento	23.948	29.690
Validação	5.987	7.422
Teste	96	164

Fonte: elaborado pelo autor

4.3 PROCESSAMENTO DE IMAGENS TRADICIONAL

Para o desenvolvimento de classificadores usando puramente processamento tradicional de imagens, foram testadas várias técnicas e combinações diferentes destas técnicas, ajustando os parâmetros e as arquiteturas baseado em observações empíricas feitas no desempenho de cada classificador.

Cada classificador possuirá uma ou mais verificações que devem ser bem-sucedidas (em conjunção) para classificar a imagem como boa, entre os métodos aplicados estão limiarização de cores, detecção de bordas utilizando os operadores de Sobel e Canny, limiarização para imagens em tons de cinza, limiarização adaptativa baseada no método de Otsu e análise de texturas. Todas essas operações foram realizadas com o suporte da biblioteca *OpenCV*.

4.3.1 OpenCV

O *OpenCV*, pode ser definido como uma biblioteca *Open Source* destinada para aplicações de visão computacional e aprendizado de máquina. A biblioteca conta com mais de 2500 algoritmos otimizados tanto de técnicas convencionais de processamento de imagens, como técnicas do estado da arte, facilitando assim o desenvolvimento de aplicações comerciais que utilizem visão computacional. Além disso, o *OpenCV* é compatível com Linux, Android, Windows e Mac OS, com suporte para C++, Python, Java e Matlab, sendo assim, amplamente utilizado em grupos de pesquisa, empresas e órgãos governamentais, com grandes empresas utilizando a

biblioteca de forma direta como, Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda e Toyota (“About OpenCV”, 2018).

4.3.2 Verificação por limiarização de histograma

Na limiarização de cores, inspirado na ideia de que imagens da terra terão cores predominantes como azul e verde, é realizada uma verificação para ver qual a porcentagem de *pixels*, em relação ao total, possui valores entre um intervalo ajustável no modelo HSV, se essa proporção estiver acima de um limiar, a imagem passa nessa verificação, correspondendo a um indício forte de pertencer a uma imagem da terra.

Um limiarizador simples pode ser usado também para detectar nas imagens em escala de cinza, pixels que ultrapassem uma intensidade alta, se a proporção de pixels que ultrapassem esse limiar for alta, isto pode significar que há uma forte saturação na imagem, podendo corresponder a uma imagem ruim.

Na limiarização adaptativa de Otsu, são buscadas regiões de *pixels* correlacionadas, como o retorno dessa função é uma imagem binária correspondendo a limiarização da imagem, a proporção de *pixels* resultante dessa imagem, será utilizada para separar imagens que tenham grandes regiões correlacionadas (imagens do horizonte, imagens completas da terra) e regiões sem grande correlação de *pixels* (espaço vazio).

4.3.3 Verificação de bordas

A detecção de bordas é um forte indicativo que a intensidade do *pixel* vária bruscamente entre *pixels* vizinhos, como em imagens muito escuras, ou imagens do espaço vazio a tendencia é menos bordas serem encontradas, essa verificação pode ser muito útil para definir se a imagem é boa ou não. Por isso, foram utilizados dois tipos diferentes de detectores de bordas para avaliar o desempenho de cada imagem, nesse quesito, o operador de Sobel, e o *Canny Edge Detector*. Se a quantidade de *pixels* de borda for maior que um limite estabelecido, a imagem passará nesta verificação.

4.3.4 Verificação de texturas

Baseado na percepção de que imagens de satélite com maior complexidade e diversidade de texturas tem maior probabilidade de representar uma captura da superfície terrestre, para esta verificação será calculada a LBP da imagem. Após o histograma ser retornado pela LBP, é feito um cálculo da entropia dessa medida, com o objetivo de tentar representar numericamente a diversidade de texturas da amostra. Novamente, se a diversidade de textura atingir um valor absoluto maior que o limiar definido, a verificação considerará a imagem como sendo boa.

4.4 MACHINE LEARNING

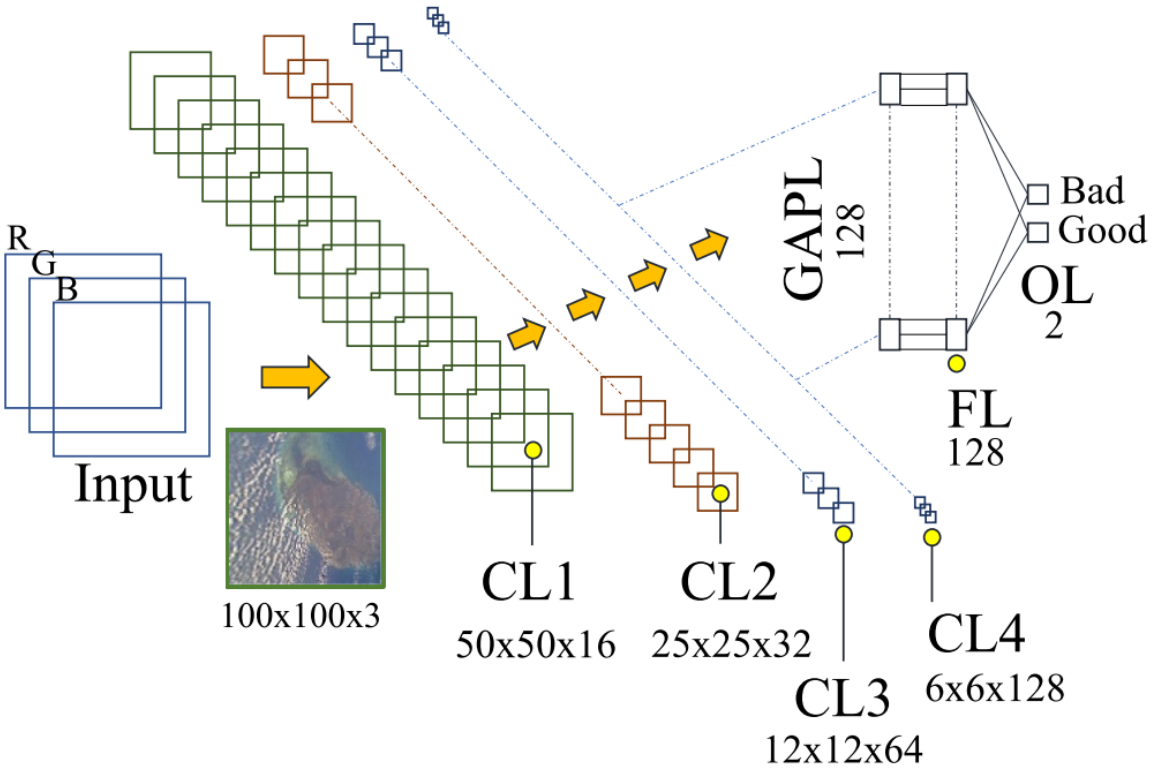
4.4.1 Arquiteturas

Com o objetivo de encontrar o melhor custo-benefício de processamento e eficiência, foram testadas diversas arquiteturas de CNNs, otimizando os hiperparâmetros para realizar a comparação do melhor modelo encontrado em cada arquitetura. Para todos os casos, o tamanho das imagens de entrada é de 100x100, devido a limitação imposta pelos bancos de dados utilizados. Todas as redes foram treinadas com base nas bibliotecas Keras e TensorFlow.

4.4.1.1 CubesatNetV2

Em (MASKEY; CHO, 2020), os autores propõem a criação de uma CNN customizada que seja leve o suficiente para poder ser implantada em um *CubeSat* do tipo 1U, esta rede tem como objetivo realizar a classificação binária das imagens entre ruim e boa, de forma similar à proposta por este trabalho, porém utilizando imagens diferentes, e critérios diferentes para dividir as imagens entre as duas classes. Assim, a arquitetura que obteve melhor desempenho, criada pelos autores, será utilizada para medir o desempenho nos novos dados de teste e treinamento.

Figura 13 – Arquitetura da CNN CubesatNetV2



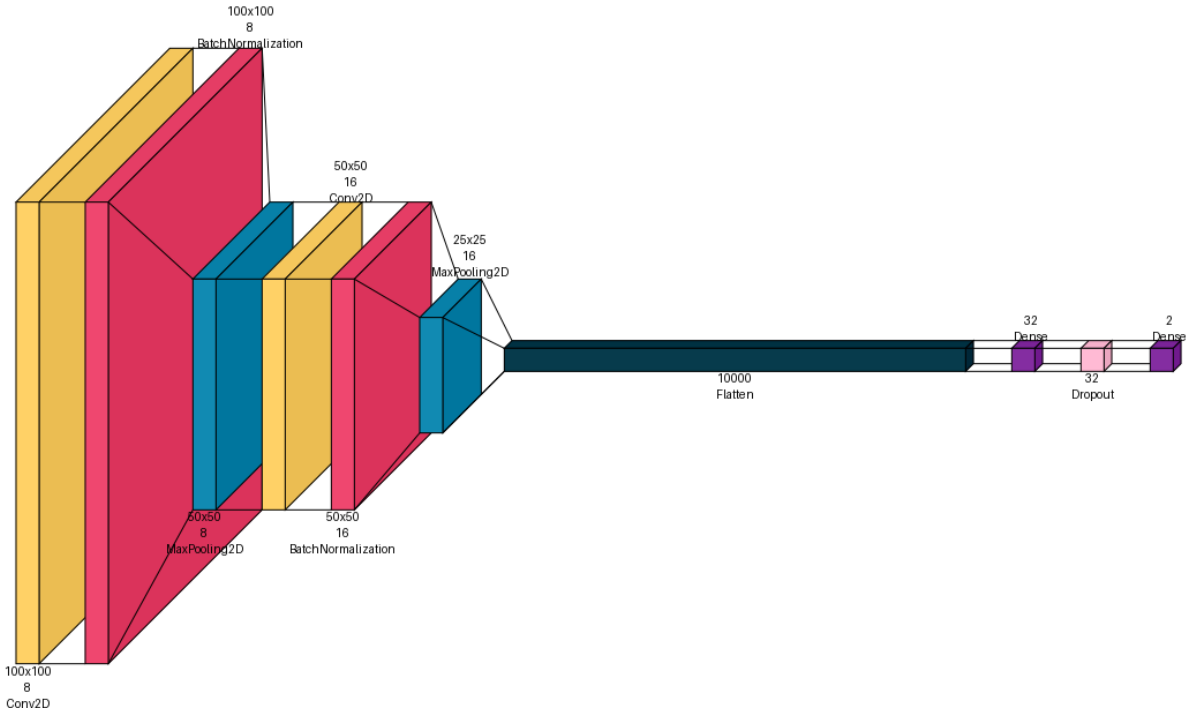
Fonte: (MASKEY; CHO, 2020)

Conforme a figura 13, a rede é bastante simplificada, com 4 camadas convolucionais, uma camada de *global average pooling* (GAPL) e uma camada totalmente conectando, totalizando 97.698 parametros treináveis na rede (MASKEY; CHO, 2020). Os autores proporam outras variações dessa rede porém a que obteve o melhor desempenho foi a versão 2, ao mesmo tempo que possuindo um valor muito baixo de parâmetros.

4.4.1.2 CNN genérica

Para fins comparativos, foi criada uma arquitetura genérica e compacta de CNN (figura 14), contendo 321.538 parâmetros treináveis, recebendo como entrada a imagem RGB 100x100 e saída a probabilidade de pertencer a classe bom ou ruim.

Figura 14 – Arquitetura da CNN Customizada



Fonte: elaborado pelo autor

A rede pode ser dividida em duas camadas convolucionais, com a primeira realizando uma operação de convolução com regularização do tipo L2 (penalizando pesos grandes, e consequentemente ajudando a controlar o *overfitting*) e função de ativação ReLU. São aplicados 8 filtros, também com o objetivo de reduzir o tamanho do modelo e generalizá-lo mais. O objetivo desta primeira camada é obter os atributos de baixo nível da imagem, como texturas e bordas. É feito um *batch normalization* e uma operação de *max pooling* para reduzir o tamanho da saída e estabilizar o treinamento.

A segunda camada convolucional segue o mesmo princípio, com a operação de convolução possuindo o mesmo tipo de função de ativação e de normalização. A diferença está no número de filtros que subiu para 16. O objetivo desta camada é detectar os atributos de nível mais alto ou mais complexos da imagem. Normalmente é feito *batch normalization* e *max pooling*.

Finalmente, os *feature maps* 2D são achatados em um vetor unidimensional para assim passarem pela camada totalmente conectada para assim retornar a probabilidade das classes na saida através de uma ativação do tipo *softmax*. É importante ressaltar que para evitar overfitting, foi utilizada técnica de *dropout* que

randomicamente coloca a entrada do neurônio como sendo 0, forçando o modelo a aprender a generalizar melhor.

4.4.1.3 MobileNet

A *MobileNet* é uma arquitetura de CNN desenvolvida pelo Google, pensada para ser aplicada em dispositivos cujo *hardware* seja limitado, como celulares e dispositivos embarcados, tornando uma opção viavel para *CubeSat* que não sejam ultralimitados em sua capacidade de computação. Ao longo do tempo este modelo foi aprimorado e foram desenvolvidas diferentes versões, podendo ser dividida em *MobileNetV1*, *MobileNetV2* e *MobileNetV3*. Neste trabalho serão utilizadas as duas primeiras versão para avaliar o seu desempenho.

A *MobileNetV1* (tabela 3) é baseada no conceito de convoluções separadas em profundidade (*depthwise separable convolutions*) que aplica apena um filtro por canal de entrada, e de convoluções *pointwise* que utilizam filtros 1x1. Além disso, é adicionado um parâmetro que permite controlar a largura do modelo, para cada camada, esse multiplicador é aplicado na entrada e saída podendo reduzir significativamente o tamanho do modelo em troca de uma perda de performance (HOWARD et al., 2017).

Tabela 3 – Arquitetura MobileNetV1

Tipo/*Stride*	**Dimensões do filtro**	**Dimensões do *Input***
Conv / s2	3 x 3 x 3 x 32	224 x 224 x 3
Conv dw / s1	3 x 3 x 32 dw	112 x 112 x 32
Conv / s1	1 x 1 x 32 x 64	112 x 112 x 32
Conv dw / s2	3 x 3 x 64 dw	112 x 112 x 64
Conv / s1	1 x 1 x 64 x 128	56 x 56 x 64
Conv dw / s1	3 x 3 x 128 dw	56 x 56 x 128
Conv / s1	1 x 1 x 128 x 128	56 x 56 x 128
Conv dw / s2	3 x 3 x 128 dw	56 x 56 x 128
Conv / s1	1 x 1 x 128 x 256	28 x 28 x 128

Conv dw / s1	3 x 3 x 256 dw	28 x 28 x 256
Conv / s1	1 x 1 x 256 x 256	28 x 28 x 256
Conv dw / s2	3 x 3 x 256 dw	28 x 28 x 256
Conv / s1	1 x 1 x 256 x 512	14 x 14 x 256
5x Conv dw / s1	3 x 3 x 512 dw	14 x 14 x 512
5x Conv / s1	1 x 1 x 512 x 512	14 x 14 x 512
Conv dw / s2	3 x 3 x 512 dw	14 x 14 x 512
Conv / s1	1 x 1 x 512 x 1024	7 x 7 x 512
Conv dw / s2	3 x 3 x 1024 dw	7 x 7 x 1024
Conv / s1	1 x 1 x 1024 x 1024	7 x 7 x 1024
Avg Pool / s11	Pool 7 x 7	7 x 7 x 1024
FC / s1	1024 x 1000	1 x 1 x 1024
Softmax / s1	Classifier	1 x 1 x 1000

Fonte: (HOWARD et al., 2017)

Tabela 4 – Arquitetura MobileNetV2

Input	**Operação**	**t**	**c**	**n**	**s**
224² x 3	conv2d	-	32	1	2
112² x 32	bottleneck	1	16	1	1
112² x 16	bottleneck	6	24	2	2
56² x 24	bottleneck	6	32	3	2
28² x 32	bottleneck	6	64	4	2
14² x 64	bottleneck	6	96	3	1
14² x 96	bottleneck	6	160	3	2
7² x 160	bottleneck	6	320	1	1
7² x 320	conv2d 1 x 1	-	1280	1	1
7² x 1280	avgpool 7 x 7	-	-	1	-
1 x 1 x 1280	conv2d 1 x 1	-	k	-	

Fonte: (SANDLER et al., [s.d.])

Já para a *MobileNetV2* (tabela 4) é apresentada uma nova inovação, o bloco de convolução invertida com conexões residuais o qual permite uma redução grande nos parâmetros do modelo.

Ambos os modelos são pre treinados, o que significa que previamente foram treinados em conjuntos de dados muito grandes, conferindo-os uma percepção de formas e caracteristicas gerais das imagens, assim esses conhecimentos prévios serão usados para classificar as imagens de satélite, sendo necessário apenas o treinamento das camadas mais superficiais, processo esse conhecido como *transfer learning*.

4.4.1.4 EfficientNet

A *EfficientNet* é a última arquitetura de CNN a ser testada, desenvolvida por um time de pesquisadores do Google AI, possui duas versões a *EfficientNetV1* e *EfficientNetV2* com o último prometendo entregar mais desempenho e menores tempos de treinamento.

O primeiro modelo desenvolvido pode ser dividido entre níveis de B0 a B7, onde B0 é a versão base do modelo, mais eficiente computacionalmente, e B1 a B7 representam versões escaladas do modelo base com cada uma elevando a complexidade através do *Compound Scaling* que otimiza o uso de recursos alterando a largura, resolução de entrada e profundidade da rede.

Para a *EfficientNetV2* o conceito de *progressive learning* é utilizado, servindo para as imagens de entrada terem suas resoluções ajustadas durante o treinamento, além do conceito de *Fused-MBConv* que melhora a eficiencia do modelo através da combinação de convoluções. A rede pode ser dividida conforme a sua proporção, entre B0 a B3 e S, M, L (pequeno, médio, grande) de forma similar a *EfficientNetV1*.

Da mesma forma que nas *MobileNet*, será utilizado *transfer learning* para treinar as camadas superficiais da rede e aproveitar o conhecimento já aprendido.

4.4.2 Treinamento

Durante a fase de treinamento, o mesmo método foi empregado para todas as arquiteturas. A função de perda escolhida foi a entropia cruzada categórica esparsa

(*sparse_categorical_crossentropy*), e o otimizador utilizado foi o Adam, que, além de ser computacionalmente eficiente, adapta a taxa de aprendizado para auxiliar na velocidade da aprendizagem.

Utilizando a função *compute_class_weight* da biblioteca *Scikit-learn*, foram calculados pesos para ambas as classes, a fim de evitar que a classe mais representativa gerasse algum viés durante o treinamento, sendo essa abordagem especialmente útil para conjuntos de treinamento desequilibrados.

O treinamento foi realizado durante 100 épocas, com a validação sendo feita via validação *hold-out*, destinando-se 20% dos dados para a validação. O tamanho das imagens utilizadas no treinamento foi de 100x100, com o objetivo de acelerar o processo.

4.4.3 Quantização

Após a otimização dos hiperparâmetros de treinamento, e treinamento efetivo dos modelos finais, um processo muito importante para a implantação foi feito, chamado de quantização. A quantização é responsável por reduzir o consumo computacional e o tamanho dos modelos, convertendo os seus pesos e ativações de ponto flutuante para representações numéricas de menos precisão.

Para a abordagem do trabalho, foi utilizada uma quantização do tipo *Integer-Only*, em que os pesos e ativações são convertidos para inteiros de 8 bits (uint8), garantindo além de um modelo mais compacto, um modelo com maior compatibilidade já que não exige processadores capazes de lidar com pontos flutuantes para ser implementado. Esse processo foi realizado com a biblioteca *TensorFlow Lite*, compatível com muitos *hardwares* especializados.

Após a otimização dos hiperparâmetros e o treinamento dos modelos finais, foi realizada uma etapa fundamental para a implantação: a quantização. Esse processo visa reduzir o consumo computacional e o tamanho dos modelos, convertendo seus pesos e ativações de ponto flutuante para representações numéricas de menor precisão.

Neste trabalho, foi adotada a quantização do tipo *Integer-Only*, na qual os pesos e ativações são convertidos para inteiros de 8 bits (uint8). Isso resulta em modelos mais compactos e com maior compatibilidade, já que não exigem

processadores com suporte a operações em ponto flutuante para sua execução. A quantização foi realizada utilizando a biblioteca *TensorFlow Lite*, que é compatível com diversos *hardware* especializados.

4.5 SIMULAÇÕES E TESTES

Na etapa final do projeto, foram realizadas diversas simulações e testes com o objetivo de estimar a aplicabilidade e o desempenho das soluções propostas. Como o foco do trabalho é a comparação entre diferentes alternativas para a classificação de imagens, as simulações foram restritas ao ambiente à nível de *software*. Assim, informações relacionadas à prototipagem em um processador real, como o consumo de energia durante a execução, não foram abordadas. No entanto, foi possível prever métricas como o uso máximo de memória durante a execução e a acurácia na fase de testes.

Segundo Maskey (2020), a escolha de um microcontrolador (MCU) para aplicações espaciais deve considerar critérios como compatibilidade, consumo energético e disponibilidade no mercado. Com base nesses critérios, optou-se por utilizar um microcontrolador da família STM32, amplamente adotado em projetos de pequenos satélites (ver Tabela 3). O modelo escolhido para as simulações foi o STM32F469NI, devido à sua eficiência energética, 2 MB de memória flash, 384 kB de RAM, suporte nativo a modelos quantizados no formato *tflite*, além de contar com ferramentas especializadas para desenvolvimento e testes que serão exploradas nas próximas seções.

Tabela 5 - Microprocessadores da família STM32 usados em *CubeSat*

Cubesat	**Tipo**	**Desenvolvedor**	**Lançamento**	**MCU**
ESTCube-1	1U	Uni. of Tartu	2013	STM32F1
SNUSAT-1/1b	2U	Seoul National Uni.	2017	STM32F4
FOX-1C	1U	AMSAT	2018	STM32L1
PW-Sat2	2U	Warsaw Uni. Of Tech	2018	STM32F1
SNUSAT-2	3U	Seoul National Uni.	2018	STM32F4

PicSat	3U	Observatoire de Paris	2018	STM32F3
KrakSat	1U	AGH Uni. of Sci. And Tech., SatRevolution S.A.	2019	STM32xx
M6P	6U	Nanoavionics	2019	STM32H7

Fonte: (MASKEY, 2020)

4.5.1 Processamento de Imagens Tradicional

O procedimento de testes seguiu um fluxo simples. Primeiramente, as combinações de verificações foram otimizadas com o objetivo de alcançar a melhor acurácia balanceada no conjunto de testes, conforme descrito na Seção 3.3, utilizando bibliotecas de processamento de imagens em *Python*, como a *OpenCV*. Em seguida, o desempenho final foi avaliado nesse mesmo conjunto de testes. O procedimento foi repetido para diferentes resoluções de imagem, com o intuito de mensurar perdas ou ganhos de desempenho conforme as alterações de tamanho.

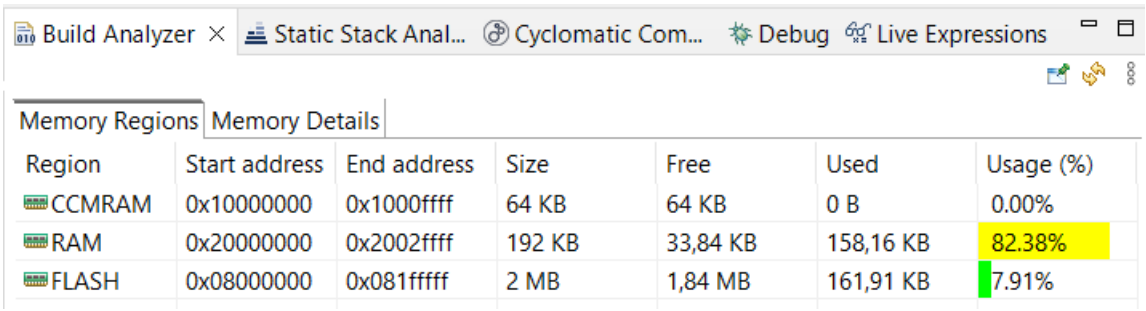
Após essa etapa, foi utilizada a *IDE* personalizada desenvolvida pela *STMicroelectronics* — fabricante dos microprocessadores escolhidos para os testes —, baseada na *IDE* Eclipse/CDT. Em conjunto com ela, empregou-se o software de emulação *QEMU*, responsável por emular e virtualizar o microcontrolador utilizado, permitindo o teste do código e a observação de informações relevantes durante sua execução.

As funções previamente utilizadas a partir da biblioteca *OpenCV* foram reescritas em linguagem C. Essa reescrita teve como objetivo reduzir o uso de memória, uma vez que, ao evitar o carregamento da biblioteca inteira, o consumo é significativamente menor. Com o auxílio da *IDE* e do *QEMU*, foi possível simular o pico de uso de memória durante a execução de cada algoritmo, obtendo assim os requisitos de memória *flash* (não volátil) e memória RAM. A memória RAM foi dividida em 2 tipos nos testes, a memória estática, isto é, a memória responsável por armazenar as variáveis estáticas declaradas, e a memória *Stack*, responsável por armazenar variáveis locais usadas dentro de funções e chamadas. Ainda é possível citar a memória *Heap*, porém como nessa implementação não foi utilizada alocação dinâmica de memória, não foi necessário medir o seu uso.

Essas informações podem ser facilmente obtidas por meio da própria *IDE* durante as simulações. A memória *flash* e a memória RAM estática são reportadas

automaticamente pelo *Build Analyzer* (Figura 16) após a emulação. De acordo com a documentação do fabricante, a memória *Stack* é alocada de forma decrescente na RAM — isto é, a partir do endereço mais alto em direção ao mais baixo. Dessa forma, torna-se possível visualizar, *byte* a *byte*, quais regiões da memória foram alteradas (inicialmente todos os *bytes* possuem valor hexadecimal 00), permitindo o cálculo da quantidade efetiva de memória *Stack* utilizada.

Figura 15 – Exemplo Build Analyzer



Fonte: elaborado pelo autor

4.5.2 Machine Learning

Assim como a *STMicroelectronics* oferece uma *IDE* voltada ao desenvolvimento de *software* para seus produtos, a fabricante também disponibiliza um conjunto de ferramentas específicas para o desenvolvimento e a implantação de redes neurais em seus microcontroladores. Como os testes deste trabalho são realizados com um microprocessador da linha STM32, foi utilizada a plataforma *ST Edge AI Developer Cloud*, desenvolvida para otimizar e realizar o *benchmark* de modelos de aprendizado de máquina nesses dispositivos.

O processo consiste em basicamente fazer upload do modelo em um dos formatos aceitos, nesse caso, como a quantização já foi realizada previamente com o auxílio da biblioteca TFlite, este foi o formato utilizado. Após isso, a plataforma fará automaticamente uma otimização para a MCU ou MPU escolhida, sendo possível escolher qual será o foco da otimização, entre menor custo de memória e tempo de inferência menor, para todos os modelos, a otimização foi ajustada para balancear entre os dois critérios. Com a otimização concluída, o benchmark do modelo final é feito na nuvem no modelo de microprocessador selecionado, retornando algumas

informações úteis, como requisitos de memória e a duração em milissegundos considerando os ciclos da CPU (“ST Edge AI Developer Cloud”, 2025).

O processo consiste, basicamente, no envio do modelo treinado em um dos formatos aceitos pela plataforma. Neste caso, como a quantização já havia sido realizada previamente utilizando a biblioteca *TensorFlow Lite*, o modelo foi carregado diretamente nesse formato. A plataforma então realiza, de forma automática, uma otimização voltada ao microcontrolador (MCU) ou processador (MPU) selecionado. Durante essa etapa, é possível configurar o foco da otimização, priorizando menor consumo de memória, menor tempo de inferência, ou um equilíbrio entre ambos. Neste trabalho, todos os modelos foram otimizados com a opção de balanceamento entre memória e desempenho. Concluída a otimização, o benchmark do modelo é executado na nuvem, utilizando o microcontrolador escolhido como referência, sendo este um dos motivos pela escolha do modelo especifico, estando este disponível para simulação em nuvem. Como resultado, são fornecidas métricas úteis, como os requisitos de memória (RAM e *Flash*) e a duração da inferência em milissegundos, com base nos ciclos da CPU (“ST Edge AI Developer Cloud”, 2025).

A ferramenta ainda possui a funcionalidade de gerar automaticamente o código necessário para implementar a rede neural no *hardware* alvo. No entanto, como o escopo deste trabalho limita-se à comparação entre métodos de classificação de imagens, essa funcionalidade não foi utilizada.

5 RESULTADOS

Os resultados das simulações realizadas com combinações de verificações, conforme exposto na seção 3.3, com os dados relacionados aos requisitos de memória para diferentes tamanhos de imagem sendo dispostos na tabela 4 e os dados relacionados ao desempenho do método no banco de imagens de teste na tabela 5.

Tabela 6 – Requisitos de memória Flash e RAM para os métodos de processamento tradicional de imagens

Método	Tamanho	Var. Globais (RAM)	Stack (RAM)	Total (RAM)	Flash
Sobel	100x100	11,72kB	0,141kB	11,72kB	44,53kB
	150x150	23,92kB	0,141kB	23,92kB	93,36kB
	200x200	41,02kB	0,141kB	41,02kB	161,72kB
Sobel + Saturação	100x100	11,72kB	0,141kB	11,861kB	44,7kB
	150x150	23,92kB	0,141kB	24,061kB	93,53kB
	200x200	41,02kB	0,141kB	41,161kB	161,89kB
Canny	100x100	80,05kB	0,199kB	80,249kB	48,41kB
	150x150	177,71kB	0,199kB	177,909kB	97,24kB
	200x200	314,43kB	0,199kB	314,629kB	165,6kB
Canny + Saturação	100x100	80,09kB	0,5078kB	80,5978kB	52,64k
	150x150	177,71kB	0,5078kB	178,2178kB	97,42kB
	200x200	314,43kB	0,5078kB	314,9378kB	165,78kB
Otsu	100x100	30,83kB	1,16kB	31,99kB	44,11kB
	150x150	67,45kB	1,16kB	68,61kB	92,94kB
	200x200	118,72kB	1,16kB	119,88kB	161,3kB
Otsu + Saturação	100x100	30,83kB	1,16kB	31,99kB	44,28kB
	150x150	67,45kB	1,16kB	68,61kB	93,11kB
	200x200	118,72kB	1,16kB	119,88kB	161,47kB
LBP	100x100	21,45kB	1,14kB	22,59kB	44,39kB
	150x150	45,86kB	1,14kB	47kB	93,22kB
	200x200	80,04kB	1,14kB	81,18kB	161,58kB
LBP + Saturação	100x100	21,45kB	1,14kB	22,59kB	44,56kB
	150x150	45,86kB	1,14kB	47kB	93,39kB
	200x200	80,04kB	1,14kB	81,18kB	161,75kB
Cores	100x100	31,21kB	0,122kB	31,332kB	42,79kB

	150x150	67,83kB	0,122kB	67,952kB	91,62kB
	200x200	119,1kB	0,122kB	119,222kB	159,98kB
Cores + Saturação	100x100	44,73kB	0,145kB	44,875kB	40,98kB
	150x150	89,8kB	0,145kB	89,945kB	93,55kB
	200x200	158,16kB	0,145kB	158,305kB	161,91kB

Fonte: elaborado pelo autor

Tabela 7 – Performance de classificação para os métodos de processamento tradicional de imagens

Método	Tamanho	Acurácia	Acurácia Balanceada	Precisão	Revocação	F1-Score
Sobel	100x100	0,700	0,758	0,553	0,979	0,707
	150x150	0,727	0,777	0,578	0,969	0,724
	200x200	0,731	0,774	0,584	0,938	0,720
Sobel + Saturação	100x100	0,769	0,774	0,655	0,792	0,717
	150x150	0,781	0,781	0,676	0,781	0,725
	200x200	0,773	0,770	0,670	0,760	0,712
Canny	100x100	0,604	0,684	0,482	0,990	0,648
	150x150	0,615	0,689	0,489	0,969	0,650
	200x200	0,638	0,705	0,505	0,958	0,662
Canny + Saturação	100x100	0,708	0,729	0,574	0,813	0,672
	150x150	0,715	0,733	0,583	0,802	0,675
	200x200	0,738	0,752	0,611	0,802	0,694
Otsu	100x100	0,750	0,748	0,640	0,740	0,686
	150x150	0,754	0,751	0,645	0,740	0,689
	200x200	0,742	0,737	0,633	0,719	0,673
Otsu + Saturação	100x100	0,731	0,696	0,659	0,563	0,607
	150x150	0,735	0,699	0,667	0,563	0,610
	200x200	0,719	0,680	0,646	0,531	0,583
LBP	100x100	0,508	0,603	0,427	0,969	0,592
	150x150	0,558	0,634	0,452	0,927	0,608
	200x200	0,581	0,648	0,465	0,906	0,615
LBP + Saturação	100x100	0,777	0,758	0,702	0,688	0,695
	150x150	0,777	0,758	0,702	0,688	0,695
	200x200	0,777	0,758	0,702	0,688	0,695

Cores	100x100	0,715	0,748	0,895	0,622	0,734
	150x150	0,715	0,748	0,895	0,622	0,734
	200x200	0,712	0,745	0,894	0,616	0,729
Cores + Saturação	100x100	0,762	0,781	0,892	0,707	0,789
	150x150	0,762	0,781	0,892	0,707	0,789
	200x200	0,758	0,778	0,891	0,701	0,785

Fonte: elaborado pelo autor

A análise das tabelas permite identificar que apesar do tamanho da imagem ter um grande impacto nos requisitos de memória, isso não se traduz no desempenho de classificação, ou seja, para os tamanhos de imagem testados, o tamanho da imagem não necessariamente impacta o desempenho dos métodos. Isso pode ser explicado também pelo fato de todas as inspeções serem feitas com valores normalizados, significando que os valores a serem comparados com o limiar, são tomados com base no número total de *pixels* da imagem.

Todos os métodos testados foram também testados em conjunto com uma inspeção de saturação, pois se constatou empiricamente que na maioria dos casos as verificações tinham dificuldade de identificar quando uma imagem estava muito saturada, identificando incorretamente uma imagem como boa, da mesma forma imagens muito escuras eram consideradas boas também. Essa verificação se mostrou eficaz em melhorar a performance de predição sem um sacrifício significativo de memória.

A seguir, os testes de hardware e performance de classificação (tabelas 6 e 7 respectivamente), foram realizados. É importante ressaltar que alguns modelos não puderam ter a duração de predição medida, pois os requisitos de Flash e RAM ultrapassam o limite máximo do MCU escolhido, o teste poderia ser realizado em outro hardware com maior capacidade de memória, porém para manter o padrão de comparação, os casos em que isso aconteceu não tiveram a duração média medida.

Tabela 8 – Requisitos e performance das CNNs treinadas

Modelo	MACC	Duração (ms)	Tamanho Pesos	Tamanho Ativações	Flash	RAM
cubesatnet	38575696	1950	381,63 KiB	201,87 KiB	397,18 KiB	205,33 KiB
cubesatnet_quant	38337268	427,9	96,12 KiB	56,63 KiB	131,63 KiB	61,16 KiB

efficientnetb0	117149040	11330	15,22 MiB	2 MiB	15,3 MiB	2,07 MiB
efficientnetb0_quant	104744968	2793	3,89 MiB	559,31 KiB	4,17 MiB	765,99 KiB
efficientnetb1	173999472	-	24,74 MiB	1,84 MiB	24,85 MiB	1,93 MiB
efficientnetb1_quant	156941140	1710	6,31 MiB	641,81 KiB	6,7 MiB	737,53 KiB
efficientnet2b0	190318592	-	22,24 MiB	710,54 KiB	22,33 MiB	782,92 KiB
efficientnet2b0_quant	180717396	-	5,69 MiB	256,11 KiB	6,05 MiB	333,67 KiB
efficientnet2b1	259591552	-	26,04 MiB	779,04 KiB	26,15 MiB	867,08 KiB
efficientnet2b1_quant	247334384	-	6,66 MiB	265,55 KiB	7,08 MiB	360,09 KiB
generico	5840184	574,0	1,23 MiB	360,37 KiB	1,24 MiB	364,22 KiB
generico_quant	5780156	242,0	314,17 KiB	131,94 KiB	349,03 KiB	136,16 KiB
mobilenet	108159264	7784	12,2 MiB	834,42 KiB	12,23 MiB	851,14 KiB
mobilenet_quant	106272772	1662	3,08 MiB	248,66 KiB	3,18 MiB	264,99 KiB
mobilenet_alpha05	28905635	2167	3,11 MiB	489,43 KiB	3,14 MiB	506,16 KiB
mobilenet_alpha05_quant	27992388	345,0	811,1 KiB	130,22 KiB	885,72 KiB	146,55 KiB
mobilenet2	78197016	6412	8,43 MiB	1,12 MiB	8,48 MiB	1,15 MiB
mobilenet2_quant	75578044	1228	2,16 MiB	320,98 KiB	2,3 MiB	351,13 KiB
mobilenet2_alpha05	25329848	2528	2,6 MiB	699,5 KiB	2,65 MiB	730,34 KiB
mobilenet2_alpha05_quant	23931772	371,3	692,18 KiB	174,8 KiB	804,71 KiB	204,9 KiB

Fonte: elaborado pelo autor

Tabela 9 – Performance de classificação das CNNs treinadas

Modelo	**Acurácia**	**Acurácia Balanceada**	**Precisão**	**Revocação**	**F1-Score**
cubesatnet	0,731	0,709	0,638	0,625	0,632
cubesatnet_quant	0,746	0,719	0,670	0,615	0,641
efficientnetb0	0,796	0,763	0,772	0,635	0,697
efficientnetb0_quant	0,796	0,769	0,753	0,667	0,707
efficientnetb1	0,842	0,838	0,767	0,823	0,794
efficientnetb1_quant	0,735	0,645	0,935	0,302	0,457
efficientnet2b0	0,781	0,733	0,791	0,552	0,650
efficientnet2b0_quant	0,792	0,753	0,784	0,604	0,682
efficientnet2b1	0,831	0,792	0,861	0,646	0,738
efficientnet2b1_quant	0,812	0,753	0,927	0,531	0,675
generico	0,746	0,693	0,734	0,490	0,588
generico_quant	0,619	0,521	0,452	0,146	0,220
mobilenet	0,838	0,837	0,755	0,833	0,792

mobilenet_quant	0,781	0,781	0,676	0,781	0,725
mobilenet_alpha05	0,858	0,848	0,804	0,813	0,808
mobilenet_alpha05_quant	0,869	0,873	0,787	0,885	0,833
mobilenet2	0,854	0,819	0,892	0,688	0,776
mobilenet2_quant	0,858	0,829	0,873	0,719	0,789
mobilenet2_alpha05	0,831	0,795	0,851	0,656	0,741
mobilenet2_alpha05_quant	0,827	0,824	0,743	0,813	0,776

Fonte: elaborado pelo autor

A análise das tabelas mostra que, as versões quantizadas dos modelos, se comparadas com a versão original, não apresentam queda significativa de performance de predição, e em alguns casos até mesmo melhora a performance, a única exceção foi o modelo genérico, que por possuir um número baixo de parâmetros se comparado aos outros modelos, pode sofrer um impacto maior da quantização de números flutuantes para inteiros, perdendo assim a precisão. Isso pode ser evidenciado pelo MACC (*Multiply-and-accumulate complexity*) que é uma usada para indicar a complexidade, do ponto de vista do processamento, de cada modelo, onde se comparado ao dos outros modelos, o do modelo genérico é muito menor.

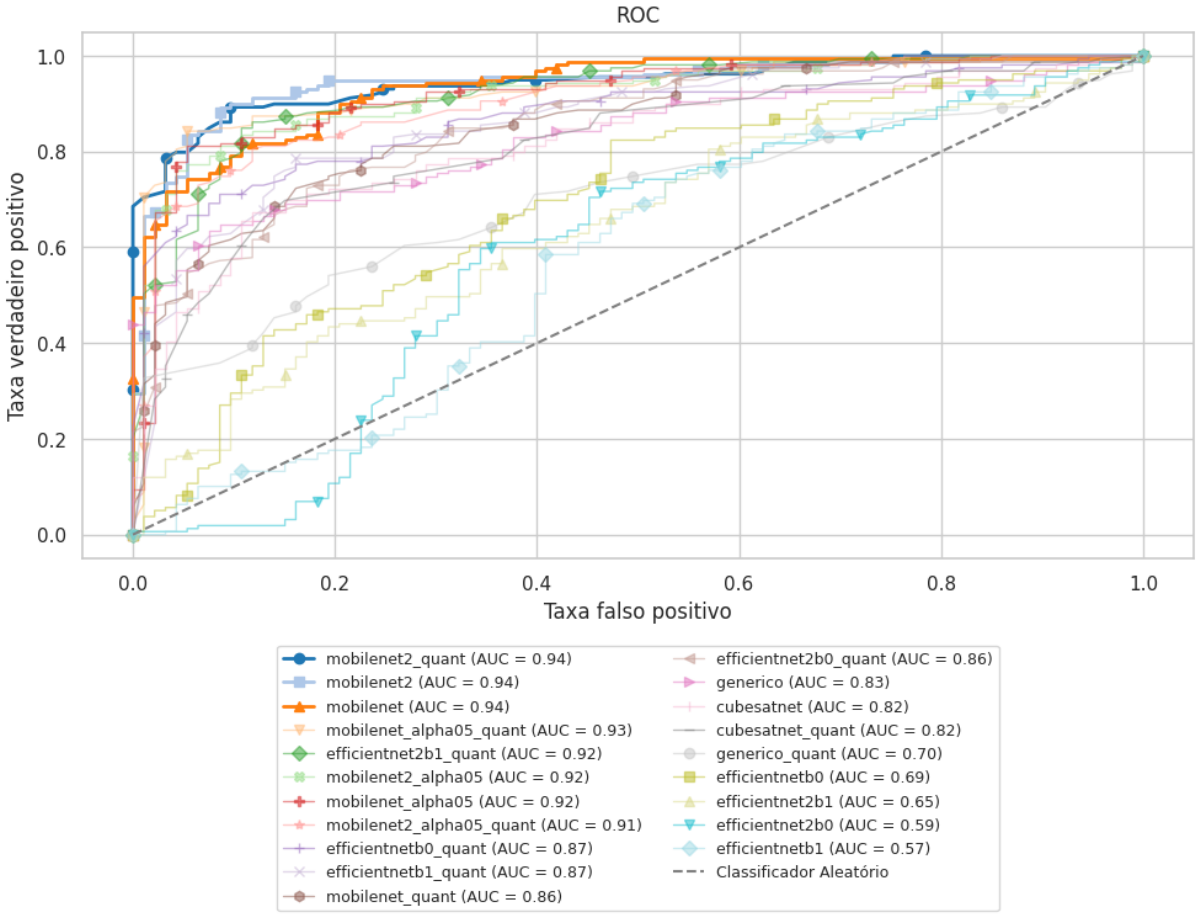
Ainda analisando a complexidade dos modelos, é possível observar que nem sempre um modelo maior significa uma performance de predição melhor, podendo significar que modelos como os da *efficientnet* por exemplo, estão possivelmente sofrendo *overfitting*, e modelos mais simples são mais eficazes para essa tarefa de classificação sendo capazes de generalizar de forma mais satisfatória.

Um modelo que se mostrou bastante balanceado em suas predições foi o *mobilenet* com alpha de 0,5 tanto em sua versão original, quanto quantizada, além disso se provando ser capaz de rodar na MCU testada sem a necessidade de adição de memória externa e com tempos de predição médios relativamente baixos.

Por fim, outra métrica muito útil que pode ser utilizada para medir e projetar o desempenho dos modelos, é a curva ROC (*Receiver Operating Characteristic Curve*). Nela é possível observar o desempenho conforme o limiar de discriminação do modelo entre as duas opções de rótulo. Na figura 18, foram ilustradas todas as curvas ROC para todos os modelos treinados, a partir dos objetivos do projeto é possível selecionar o limiar que atinja uma taxa de verdadeiros positivos e/ou falsos positivos ótima.

Na figura 18, estão destacados os modelos com maior AUC (*Area Under ROC Curve*), sendo assim, os modelos que mais se aproximam de um modelo de previsão perfeito.

Figura 16 – Curvas ROC dos modelos treinados



Fonte: elaborado pelo autor

6 CONCLUSÃO E TRABALHOS FUTUROS

6.1 CONCLUSÃO

Este trabalho teve como principal objetivo propor e comparar diferentes métodos para realizar a classificação binária de imagens provindas de pequenos satélites. Na seção 4 foram expostos os dados obtidos a partir de simulações realizadas em *hardware* emulado, onde para os métodos de processamento de imagem tradicionais, se destaca a inspeção de cores em conjunto com o filtro de saturação, obtendo uma acurácia balanceada de 0,781 e F1-Score de 0,789 com um custo de *hardware* relativamente baixo se comparado com os outros modelos.

Já para os modelos de CNNs treinados, se destacam os da família mobilenet, em especial o “mobilenet_alpha05_quant” podendo obter 0,833 de acurácia balanceada e 0,885 de F1-Score com uma duração média de previsão menor que 1 segundo e requisitos de memória relativamente baixos se comparado com os outros modelos.

A escolha de qual a melhor técnica, depende fundamentalmente das características da missão, por exemplo, missões com limitações severas de *hardware* como *CubeSat* do tipo 1U, podem exigir sacrifícios de na acurácia para comportarem o *software*, preferindo assim técnicas tradicionais de processamento digital, ou modelos leves e quantizados. Em missões com fins cientificas, onde a qualidade das imagens é imprescindível para tirar alguma conclusão e é preferível descartar algumas imagens boas a enviar imagens ruins para a estação terrestre, a revocação é uma boa métrica a ser analisada. No caso contrário, onde imagens boas não podem ser descartadas, e ruido na transmissão não tem grande importância, a precisão é um bom indicativo de qual técnica escolher. Uma alternativa mais balanceada seria analisar o F1-Score. Na figura 18, é possível escolher um limiar para os modelos de *machine learning* que maximizem essa métrica escolhida, otimizando mais ainda o modelo para a tarefa escolhida.

6.2 TRABALHOS FUTUROS

O desenvolvimento deste trabalho abre a possibilidade para novas pesquisas no tema serem desenvolvidas, entre as possíveis melhorias para o projeto é possível citar:

- Aquisição de novas imagens no domínio de pequenos satélites para assim melhorar a qualidade do banco de dados;
- Juntar processamento de imagens tradicional com os modelos treinados para tentar melhor o desempenho de previsão;
- Realizar testes em *hardware* físico para validar os tempos de previsão, potência consumida durante o pico de processamento;
- Aplicar algum dos modelos propostos em uma aplicação real para medir o desempenho em um ambiente que não seja controlado.

REFERÊNCIAS

[1] KLEIN, J.; HAWKINS, J.; THORSEN, D. Improving cubesat downlink capacity with active phased array antennas. **ScholarWorks-UA (University of Alaska Fairbanks)**, 1 mar. 2014.

[2] TYAGI, V. Understanding Digital Image Processing. 13 set. 2018.

[3] **About OpenCV**. Disponível em: <https://opencv.org/about/>.

[4] SONG, R.; ZHANG, Z.; LIU, H. Edge connection based Canny edge detection algorithm. **Pattern Recognition and Image Analysis**, v. 27, n. 4, p. 740–747, out. 2017.

[5] RONG, W. et al. **An improved Canny edge detection algorithm**. Disponível em: <https://ieeexplore.ieee.org/document/6885761>. Acesso em: 5 dez. 2020.

[6] OPENCV. **OpenCV: Canny Edge Detection**. Disponível em: <https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html>.

[7] GONZALEZ, C. I. et al. An improved sobel edge detection method based on generalized type-2 fuzzy logic. **Soft Computing**, v. 20, n. 2, p. 773–784, 16 dez. 2014.

[8] KOLKUR, S. et al. Human Skin Detection Using RGB, HSV and YCbCr Color Models. **Proceedings of the International Conference on Communication and Signal Processing 2016 (ICCASP 2016)**, 2017.

[9] SZABOLCS SERGYAN. **Color content-based image classification**. Disponível em: <https://www.researchgate.net/publication/229005517_Color_content-based_image_classification>. Acesso em: 13 out. 2024.

[10] GONZALEZ, R. C.; WOODS, R. E.; MASTERS, B. R. Digital Image Processing, Third Edition. **Journal of Biomedical Optics**, v. 14, n. 2, p. 029901, 2009.

[11] CHENG, H. D. et al. Color image segmentation: advances and prospects. **Pattern Recognition**, v. 34, n. 12, p. 2259–2281, 1 dez. 2001.

[12] OTSU, N. A Threshold Selection Method from Gray-Level Histograms. **IEEE Transactions on Systems, Man, and Cybernetics**, v. 9, n. 1, p. 62–66, jan. 1979.

[13] PIETIKÄINEN, M. Local Binary Patterns. **Scholarpedia**, v. 5, n. 3, p. 9775, 2010.

[14] OJALA, T.; PIETIKÄINEN, M.; HARWOOD, D. A comparative study of texture measures with classification based on featured distributions. **Pattern Recognition**, v. 29, n. 1, p. 51–59, jan. 1996.

[15] GÉRON, A. **Hands-on machine learning with Scikit-Learn and TensorFlow concepts, tools, and techniques to build intelligent systems**. 2. ed. [s.l.] O'Reilly Media, Inc., 2019.

[16] GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. Cambridge, Massachusetts: The MIT Press, 2016.

[17] ALPAYDIN, E. **Introduction to machine learning**. Cambridge, Massachusetts: The Mit Press, 2020.

[18] WATT, J.; REZA BORHANI; AGGELOS KONSTANTINOS KATSAGGELOS**. Machine learning refined : foundations, algorithms, and applications**. Cambridge, United Kingdom: Cambridge University Press, 2016.

[19] BISHOP, C. M. **Pattern Recognition and Machine Learning**. [s.l.] Springer, 2006.

[20] MURPHY, K. P. **Probabilistic machine learning: an introduction**. Cambridge: MIT Press, 2022.

[21] TAYE, M. M. Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions. **Computation**, v. 11, n. 3, p. 52, 6 mar. 2023.

[22] YAMASHITA, R. et al. Convolutional Neural networks: an Overview and Application in Radiology. **Insights into Imaging**, v. 9, n. 4, p. 611–629, 22 jun. 2018.

[23] XU, Y.; GOODACRE, R. On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the Generalization Performance of Supervised Learning. **Journal of Analysis and Testing**, v. 2, n. 3, p. 249–262, jul. 2018.

[24] KOHAVI, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. **International Joint Conference on Artificial Intelligence**, v. 2, p. 1137–1143, 20 ago. 1995.

[25] PEREZ, L.; WANG, J. **The Effectiveness of Data Augmentation in Image Classification using Deep Learning**. Disponível em: <https://arxiv.org/abs/1712.04621v1>.

[26] MASKEY, A. Data for: CubeSatNet: Ultralight Convolutional Neural Network Designed for On-orbit Binary Image Classification on a 1U CubeSat. **Mendeley Data**, v. 1, 16 set. 2020.

[27] PRISE6. **Earth from ISS HDEV experiment**. Disponível em: <https://www.kaggle.com/datasets/prise6/earth-from-iss-hdev-experiment>. Acesso em: 1 dez. 2024.

[28] POOJA LEPCHA. Image Dataset taken by 1U CubeSat Constellation in the Low Earth Orbit. **Mendeley Data**, v. 2, 27 fev. 2023.

[29] MASKEY, A.; CHO, M. CubeSatNet: Ultralight Convolutional Neural Network designed for on-orbit binary image classification on a 1U CubeSat. **Engineering Applications of Artificial Intelligence**, v. 96, p. 103952, nov. 2020.

[30] HOWARD, A. et al. **MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications**. [s.l: s.n.]. Disponível em: <https://arxiv.org/pdf/1704.04861>.

[31] SANDLER, M. et al. **MobileNetV2: Inverted Residuals and Linear Bottlenecks**. [s.l: s.n.]. Disponível em: <https://arxiv.org/pdf/1801.04381>.

[32] **ST Edge AI Developer Cloud**. Disponível em: <https://stedgeai-dc.st.com/documentation>. Acesso em: 8 abr. 2025.

[33] SOBEL, I. **An Isotropic 3x3 Image Gradient Operator**. Disponível em: <https://www.researchgate.net/publication/239398674_An_Isotropic_3x3_Image_Gradient_Operator>.

[34] MANE, S. Theoretical Overview on CubeSat Technology. **ResearchGate**, v. 12, n. 1, p. 1107–1113, 23 jan. 2024.

[35] SPACE, A. **10 Advantages of CubeSats vs. Conventional Satellites**. Disponível em: <https://info.alen.space/advantages-of-cubesats-vs-conventional-satellites>.

[36] TOKI, S. A. et al. **A Shallow Convolutional Neural Network for Onboard CubeSat Image Classification**. Disponível em: <https://www.researchgate.net/publication/387903361_A_Shallow_Convolutional_Neural_Network_for_Onboard_CubeSat_Image_Classification>. Acesso em: 5 jul. 2025.

[37] ALEXSEI, K. **Onboard Data Prioritization using Image Classification and Segmentation for Nanosatellites**. Disponível em: <https://kyutech.repo.nii.ac.jp/records/2001044>. Acesso em: 5 jul. 2025.

[38] YAO, Y. et al. On-Board Ship Detection in Micro-Nano Satellite Based on Deep Learning and COTS Component. **Remote Sensing**, v. 11, n. 7, p. 762–762, 29 mar. 2019.