
UNIVERSIDADE PAULISTA

RAFAEL BASTO CANUDO
THIAGO MURADIAN SIMÃO

ATIVIDADES PRÁTICAS SUPERVISIONADAS

# Desenvolvimento de um Sistema para Análise e Processamento de Imagens de Satélite para Identificação de Desmatamento Na Amazônia

São Paulo
2025

RAFAEL BASTO CANUDO
THIAGO MURADIAN SIMÃO

# ATIVIDADES PRÁTICAS SUPERVISIONADAS
## Desenvolvimento de um Sistema para Análise e Processamento de Imagens de Satélite para Identificação de Desmatamento Na Amazônia

Trabalho de Atividades Práticas Supervisionadas apresentado ao curso de Ciência da Computação da UNIVERSIDADE PAULISTA no 4° semestre

Orientadora: Kelly Costa

São Paulo
2025

# Sumário

## 1. OBJETIVO DO TRABALHO

A análise de performance de algoritmos aplicada ao geoprocessamento de imagens da Amazônia, incluindo estados como Amazonas, Pará, Acre e Mato Grosso, e do cerrado central, abrangendo áreas em Goiás e Minas Gerais, configura-se como um campo de pesquisa rico e promissor, apresentando um significativo potencial de impacto. O desmatamento nessas regiões críticas não apenas compromete a biodiversidade global, mas também altera o clima do planeta de forma alarmante. O monitoramento contínuo da floresta e do cerrado por meio de imagens de satélite é essencial para a fiscalização e combate a atividades ilegais.

Nesse cenário, o desenvolvimento de algoritmos eficientes para a análise de grandes volumes de dados geográficos é fundamental, possibilitando a tomada de decisões rápidas e precisas. O objetivo deste trabalho é incentivar a pesquisa e o desenvolvimento de soluções inovadoras para enfrentar um dos maiores desafios ambientais contemporâneos, com ênfase na detecção e classificação de áreas desmatadas na Amazônia e no cerrado central.

O grupo de pesquisa irá explorar diversas técnicas de aprendizado de máquina e Deep learning, aplicando-as a um conjunto de dados de imagens de alta resolução, visando otimizar a performance dos algoritmos e contribuir para a preservação da floresta e do cerrado.

## 2. INTRODUÇÃO

Em um mundo cada vez mais digital, o uso de IAs, em específico as LLMs (Large Language Models ou, em tradução livre, Grandes Modelos de linguagem) como o GPT da OpenAI e o Gemini do Google, se tornou comum. Segundo uma pesquisa da GetApp (2023), 25% dos profissionais utilizam diariamente algum modelo de IA e 53% utilizam algumas vezes na semana. Porém, o ChatGPT e o Gemini além de serem chats de texto, também são capazes de interpretar imagens, áudios e vídeos em versões pagas, sendo assim aplicadas técnicas diversas para o aprendizado da máquina (Machine Learning) e aprendizado profundo (Deep Learning).

A compreensão de um panorama histórico da evolução do estudo matemático e computacional em Inteligência Artificial permite compreender as bases que levaram à criação de diversas técnicas modernas de Machine Learning e, consecutivamente, Deep Learning.

## 2.1 UM BREVE RESUMO DA HISTÓRIA DO MACHINE LEARNING

O debate moderno sobre máquinas inteligentes começa, de fato, com Alan Turing. Em 1950, no texto *Computing Machinery and Intelligence*, Turing propôs o experimento que hoje conhecemos como Teste de Turing: uma máquina que consiga manter uma conversa que não possa ser diferenciada daquela de um humano seria, para todos os efeitos práticos, inteligente.

Na década de 1950 consolidou-se um movimento de pesquisadores interessados em transformar essa questão em trabalho concreto. Em 1956, durante a Conferência de Dartmouth, John McCarthy e outros formalizaram o termo *Artificial Intelligence* e deram início a um esforço organizado para construir programas capazes de raciocinar, representar conhecimento e resolver problemas. Surgiram linguagens e ferramentas específicas, e uma geração de projetos focou em formalizar o conhecimento humano por meio de regras e inferência lógica.

Esse período é marcado pela chamada IA simbólica. A ideia era direta: codificar o saber em regras e deixar que a máquina as aplicasse. Sistemas especialistas ilustram bem essa proposta. Sobre o projeto DENDRAL, Russell e Norvig (2013, p. 38) registram que:

> “Ele foi desenvolvido em Stanford, onde Ed Feigenbaum (um antigo aluno de Herbert Simon), Bruce Buchanan (filósofo transformado em cientista da computação) e Joshua Lederberg (geneticista laureado com um Prêmio Nobel) formaram uma equipe para resolver o problema de inferir a estrutura molecular a partir das informações fornecidas por um espectrômetro de massa.”

DENDRAL e projetos como MYCIN mostraram que era possível automatizar raciocínios de especialistas. Mas havia um custo alto: criar e manter bases de regras exigia muito trabalho e não garantia a generalização. As soluções funcionavam bem em domínios restritos, mas fragilizaram-se quando os problemas ficaram menos bem definidos. Esse desgaste contribuiu para o chamado “inverno da IA”, quando expectativas e financiamentos recuaram.

A reação veio com mudança de foco: em vez de codificar regras, por que não fazer os sistemas aprenderem a partir de dados? Assim emergiu o aprendizado de máquina, que apoia-se em estatística, probabilidade e otimização. Mitchell (1997)

resume bem essa virada: trata-se do estudo de algoritmos que melhoram seu desempenho por meio da experiência, sem a necessidade de instruções explícitas para cada tarefa. Em termos práticos, passou-se a ajustar modelos a exemplos em vez de abastecê-los com regras fixas.

Nas décadas seguintes, técnicas como árvores de decisão, SVMs e redes neurais tornaram-se ferramentas de uso corrente. O aprendizado supervisionado, o não supervisionado e o aprendizado por reforço ganharam formas e aplicações concretas. E então veio o salto computacional: GPUs, grandes conjuntos de dados e bibliotecas eficientes permitiram treinar redes muito maiores. O resultado foi o Deep Learning, que, a partir de trabalhos como AlexNet, mostrou ganhos substanciais em visão computacional, e, depois, com os Transformers, redefiniu a forma de trabalhar com a linguagem.

À partir dos anos 2000, as IAs e modelos de Deep Learning começaram a ser integrados ao cotidiano da sociedade. Os estudos de processamento de linguagem natural levaram a criação de assistentes virtuais como a Siri da Apple e a Alexa da Amazon. Carros autônomos se tornaram uma realidade, mesmo que não sejam de acesso à população no geral.

> “Em 2012, a Google deu mais um passo em seus sistemas de IA. Consolidando tecnologias em desenvolvimento desde 2006 em deep learning, ela conseguiu treinar um algoritmo para reconhecer gatinhos em vídeos do YouTube” (BARBOSA, BEZERRA, 2020, p.6)

Em 2022, houve o lançamento da LLM mais conhecida, o ChatGPT, um chatbot de IA generativa que revolucionou a forma como o público em geral teve acesso a um modelo de IA avançado de forma acessivel, porém levantou questionamentos sobre plágio e o avanço de modelos generativos de texto, imagem e até vídeo.

## 3. FUNDAMENTOS DAS PRINCIPAIS TÉCNICAS DE ANÁLISE DE IMAGENS

De forma resumida, é correto dizer que a análise e classificação de imagens se encaixam em dois tipos, baseados conforme seu funcionamento com rótulos. Classificar como Supervisionado ou Não-Supervisionado indica como ele se comporta com informações externas a imagem, no caso, sendo essas informações chamadas de rótulos.

## 3.1 TÉCNICAS CLÁSSICAS DE ANÁLISE DE IMAGENS

Antes das técnicas modernas de Deep Learning, técnicas de Machine Learning eram mais utilizadas para identificação de padrões em imagens (principalmente em reconhecimento facial), sendo estas técnicas separadas em três categorias principais, segundo Alfalou et al. (2018):

- Abordagens de características locais
- Abordagens de aprendizagem de subespaço
- Abordagens de filtros de correlação

Cada categoria busca identificar padrões por meio de comparação na própria imagem ou usando uma referência externa, porém, essas peculiaridades também criam pontos fracos, como por exemplo, a dificuldade em lidar com mudanças de iluminação e direção do rosto. A maioria das técnicas clássicas de Machine Learning são Não-Supervisionadas pois surgiram em um período que não se tinha grandes bases de dados rotuladas de facil acesso.

### 3.1.1 ABORDAGENS DE CARACTERÍSTICAS LOCAIS

Esses métodos ”por meio de informações de orientação local, histogramas, propriedades geométricas e correlação” (ALFALOU ET AL. , 2018, p.4, tradução nossa) são os mais utilizados para reconhecimento facial e reconhecimento de objetos ou pessoas por conta das suas capacidades de identificar padrões na própria imagem, ou seja, de forma Não-Supervisionada.

As principais abordagens são Histograma de Gradiente Orientado (Histogram of Oriented Gradients ou HOG), Padrões Binários Locais (Local Binary Patterns ou LBP) e Transformação de Características Invariantes à Escala (Scale-Invariant Feature Transform ou SIFT).

> “Quando se fala de Histograma de Gradientes Orientados (Histogram of Oriented Gradients - HOG), a idéia básica é que tanto as formas quanto as aparências dos objetos contidos em uma imagem podem ser muito bem caracterizadas a partir da distribuição local das intensidades dos gradiente” (COSTA, COUTO, 2008, p.8)

Portanto, ele funciona através da análise da variação na escala de cinza para identificar as bordas e as direções que elas seguem. Por conta do jeito que funciona, esse método não é o ideal se houver pouca variação de luz na imagem ou se houver variação na pose do rosto analisado.

Já o LBP funciona à partir da comparação da variação da escala de cinza de pixels em volta de um pixel central, sendo portanto a comparação de oito pixels em relação a um pixel. Os dois métodos se utilizam da mesma métrica, ou seja, a variação da escala de cinza na imagem adaptada para preto e branco, porém o HOG é focado em delimitar as bordas e as direções dessas bordas da imagem, já o LBP busca mapear uma parte da imagem pixel a pixel para identificar um padrão, ou um número para cada pixel, e assim gerar um mapa de padrões locais que posteriormente pode ser aplicado na imagem.

"Desenvolvido por David Lowe, o SIFT é conhecido por sua invariância ao dimensionamento e rotação e por sua alta robustez a mudanças de iluminação e ruído" (SOUZA, 2024, p.13). Esse tipo de técnica analisa a imagem em busca de pontos-chaves que são invariáveis na imagem, podendo assim serem identificados em outros ângulos e outros níveis de iluminação.

### 3.1.2 ABORDAGENS DE APRENDIZAGEM DE SUBESPAÇO

A redução da dimensionalidade da imagem para um subespaço mantendo somente as informações essenciais é o princípio dessas abordagens, sendo Análise de Componentes Principais (Principal Component Analysis ou PCA) e Análise Discriminante Linear (Linear Discriminant Analysis ou LDA) as mais relevantes nessa categoria.

A PCA é uma técnica Não-Supervisionada que diminui dados de imagens, principalmente de alta resolução, da melhor forma tentando encontrar as características fundamentais da imagem, perdendo assim, pequenos detalhes da imagem. Já a LDA utiliza de classes, ou rótulos, para fazer um processo parecido de redução da imagem em subespaço, porém por ser um uma técnica Supervisionada, o foco está em reconhecer padrões em comparação às classes de referência.

### 3.1.3 ABORDAGENS DE FILTROS DE CORRELAÇÃO

Por fim, nesses tipos de abordagem a maioria são Supervisionadas por conta do seu método de utilizar um modelo como referência para varrer a imagem em busca de correlação. O caso clássico disso é o Filtro Casado (Matched Filter ou MF) que tem como modelo uma imagem ou objeto e varre a imagem de busca de correlação. Entretanto, por usar uma imagem estática como referência, essa técnica se complica com mudanças de ângulos, de luminescência e qualidade da imagem.

Um aprimoramento desse método é o Filtro de Mínima Soma do Erro Quadrático (Minimum Output Sum of Squared Error ou MOSSE) que usa varias imagens do objeto para criação de filtro que gera um mapa de resposta que o pico máximo indica a posição do objeto na imagem.

## 3.2 TÉCNICAS MODERNAS DE DEEP LEARNING

Deep Learning é uma subárea de estudo de Machine Learning “dedicada ao estudo e desenvolvimento de máquinas que podem aprender (às vezes com o objetivo de eventualmente atingir inteligência artificial geral).” (TRASK, 2019, p.10, tradução nossa). As técnicas modernas de Deep Learning são focadas em arquiteturas com várias camadas compostas de neurônios artificiais interconectados e cada um desses neurônios possuem seus pesos e seus “bias”, que são parâmetros para variação no treinamento do modelo.

### 3.2.1 REDES NEURAIS CONVULOCIONAIS (CNNs)

As Redes Neurais Convolucionais (Convolutional Neural Network ou CNNs) são as arquiteturas mais conhecidas em técnicas de Deep Learning e servindo como base para outras arquiteturas

Seu funcionamento baseia-se na capacidade de extrair automaticamente características espaciais de uma imagem por meio de operações de convolução, evitando a necessidade de engenharia manual de atributos. Segundo LeCun et al. (2015, p. 436, tradução nossa), “O aprendizado profundo permite que modelos computacionais compostos por múltiplas camadas de processamento aprendam representações de dados com múltiplos níveis de abstração”. Cada camada convolucional aplica filtros convolucionais (kernels), que são pequenas matrizes de pesos capazes de detectar padrões locais, como bordas, cantos, texturas e formas, gerando mapas de ativação que representam a presença desses padrões na imagem.

Durante o treinamento, os pesos dos filtros e os parâmetros de bias são ajustados automaticamente por algoritmos de otimização, como o backpropagation, permitindo que a rede aprenda a reconhecer padrões complexos de forma hierárquica, das características mais simples às mais abstratas.

Além das camadas convolucionais, as CNNs geralmente apresentam camadas de pooling, responsáveis por reduzir a dimensionalidade espacial dos mapas de ativação, e camadas totalmente conectadas, que realizam a classificação final ou outras tarefas específicas. A combinação dessas camadas permite que as CNNs sejam altamente eficazes em tarefas de reconhecimento de objetos, classificação de imagens e detecção de padrões visuais, sendo amplamente utilizadas em aplicações como reconhecimento facial, análise médica de imagens e veículos autônomos.

Dentre as principais arquiteturas, destaca-se a LeNet-5, proposta por LeCun et al. (1998), considerada pioneira no reconhecimento de dígitos manuscritos. Esta arquitetura introduziu a estrutura de múltiplas camadas convolucionais seguidas de pooling e camadas totalmente conectadas, servindo de referência para o desenvolvimento de redes mais profundas. Outro exemplo é a AlexNet que utilizou camadas convolucionais profundas, ReLU (serve para introduzir não linearidade ao modelo gerando maior diversidade no treinamento) como função de ativação e técnicas de regularização, como dropout, demonstrando significativamente melhores resultados em classificação de imagens quando comparada às redes anteriores.

### 3.2.2 REDES DE SEGMENTAÇÃO E ARQUITETURAS ENCODER-DECODER

As redes de segmentação são arquiteturas de deep learning projetadas para classificar cada pixel de uma imagem, permitindo uma compreensão detalhada das estruturas presentes. Diferente das CNNs usadas para classificação global de imagens, essas redes realizam a segmentação de regiões específicas, sendo essenciais em aplicações como análise médica, reconhecimento de objetos e processamento de imagens geoespaciais.

Um dos paradigmas mais utilizados em redes de segmentação é a arquitetura encoder-decoder, em que o encoder extrai características hierárquicas da imagem de entrada e o decoder reconstrói a segmentação pixel a pixel, preservando informações espaciais importantes (BADRINARAYAN; KENDALL; CIPOLLA, 2017). Entre essas arquiteturas, a U-Net, proposta por Ronneberger, Fischer e Brox (2015), destaca-se por sua eficiência e precisão mesmo quando treinada com bases de dados relativamente pequenas. A U-Net combina um caminho de contração, responsável por extrair as características mais relevantes da imagem, e um caminho

de expansão, que reconstrói a segmentação mantendo detalhes espaciais. Um elemento central da U-Net são as conexões de salto (skip connections), que conectam camadas correspondentes do encoder e do decoder, permitindo que informações finas do encoder sejam reutilizadas no decoder para melhorar a precisão da segmentação (RONNEBERGER; FISCHER; BROX, 2015).

Originalmente desenvolvida para segmentação de imagens biomédicas, a U-Net tem sido aplicada com sucesso em outras áreas, como reconhecimento de objetos em imagens naturais e segmentação de características geográficas em imagens de satélite (CHEN et al., 2018).

### 3.2.3 REDES GENERATIVAS

As redes generativas são uma classe de arquiteturas de deep learning projetadas para gerar novos dados a partir de distribuições aprendidas, sendo amplamente utilizadas em imagens, áudio e texto. Entre os principais tipos de redes generativas estão os autoencoders e as Generative Adversarial Networks (GANs). Os autoencoders são compostos por um encoder, que reduz a dimensionalidade dos dados de entrada para um espaço latente, e um decoder, que reconstrói os dados originais a partir dessa representação comprimida. Essa abordagem permite que a rede aprenda a codificar características essenciais do conjunto de dados, sendo útil para tarefas como compressão de imagens, remoção de ruído e geração de variações de dados existentes (GOODFELLOW; BENGIO; COURVILLE, 2016).

Já as GANs, introduzidas por Goodfellow et al. (2014, citado em TRASK, 2019), consistem em duas redes treinadas de forma adversarial: um gerador, que cria novos dados, e um discriminador, que avalia se os dados são reais ou gerados. O treinamento dessas redes cria um processo de competição, no qual o gerador melhora progressivamente a qualidade das amostras geradas, enquanto o discriminador se torna mais preciso em distinguir dados reais de falsos. Esse princípio tem permitido avanços notáveis na geração de imagens realistas, criação de conteúdo sintético para treinamento de modelos e até na transferência de estilos visuais.

Os autoencoders variacionais (VAE) são uma extensão que combina a capacidade de reconstrução dos autoencoders com uma abordagem probabilística,

permitindo a geração de novas amostras mostrando diretamente o espaço latente. Essa técnica é particularmente útil em aplicações médicas, onde imagens geradas a partir de pequenas variações no espaço latente podem auxiliar no treinamento de modelos de segmentação e detecção de anomalias. De modo geral, redes generativas, tanto GANs quanto autoencoders, exemplificam como o deep learning evoluiu para além da classificação e segmentação, explorando a capacidade de modelar e criar dados complexos a partir de representações latentes aprendidas (TRASK, 2019; GOODFELLOW; BENGIO; COURVILLE, 2016).

A aplicação dessas redes vai além da geração de imagens; elas são utilizadas para aumentar conjuntos de dados em cenários de escassez de informação, melhorar diagnósticos médicos através da simulação de imagens raras, criar arte digital e até gerar modelos preditivos em simulações científicas. Combinadas com outras arquiteturas, como CNNs e U-Nets, GANs e autoencoders ampliam significativamente o potencial de aprendizado e generalização dos sistemas de visão computacional, consolidando-se como ferramentas essenciais no ecossistema de deep learning moderno.

## 4. METODOLOGIA APLICADA

Após a análise das técnicas, o grupo de pesquisa constatou que as redes de segmentação e encoder-decoder encaixam melhor para a produção de um sistema de processamento de imagens. Alguns pontos que levaram a essa decisão foram a capacidade de supervisionamento do sistema, o controle do aprendizado e a aplicação de imagens de satélites. A rede escolhida foi a U-net por seu modelo de aprendizado ter sido criado para segmentação pixel a pixel, sendo assim ideal para análise de imagens com somente duas respostas possíveis, ou seja, análise binária (floresta ou não floresta).

Por ser do tipo encoder-decoder, a entrada dela recebe uma imagem de satélite + a *mask* demarcando desmatamento ou não dessa imagem, sendo do tipo GeoTIFF (imagens de alta resolução de satélites). O modelo aprende à partir da tentativa de criar uma *mask* predita para a imagem e a comparação com a *mask* real, gerando a quantidade de *loss* e ajustando para a próxima tentativa desse processo, sendo chamado de época.

Outros parâmetros utilizados foram *Pixel Accuracy, IoU* e *Dice.* O *Pixel Accuracy* consiste na divisão da quantidade de acertos dos *pixels* da *mask* predita com a quantidade total de *pixels* da *mask* real. Esse parâmetro não é tão confiável se as imagens possuírem grandes áreas de floresta preservada, pois o modelo aprende a tentar todos os pixels como floresta preservada para garantir um valor alto. Esse peso foi utilizado por conta do dataset utilizado para treinar o modelo ter imagens diversas.

O *IoU* é um peso que utiliza a divisão da intersecção e a união da *mask* predita e a real. Esse peso segue a lógica do parâmetro anterior, porém o refina se baseando em quanto realmente o modelo acertou em relação ao todo. Por fim, o *Dice* compara as máscaras diretamente, utilizando a intersecção das *masks* pela soma total dos *pixels* de ambas.

Para a inferência do sistema, ele seleciona uma imagem aleatória do dataset sem pegar a *mask* equivalente e tenta predizer onde está desmatado gerando uma máscara que ao fim é mostrada ao usuário junto da mask e da imagem real.
O Dataset utilizado possui entre 350 a 500 imagens + masks catalogando áreas como florestas e não florestas. Além das imagens de satélite da Amazônia, também foram utilizadas imagens da Mata Atlântica.

O link a seguir é um vídeo com a apresentação do sistema, sua funcionalidades e um teste e validação da análise de imagens e criação de *masks* de desmatamento:
https://drive.google.com/drive/folders/12RFDCY2wVZlefERHSeLy0033MwyqwqPH

## 5. CÓDIGO DO SISTEMA

### 5.1 Dataset.py

```
import torch
from torch.utils.data import Dataset
import numpy as np
import rasterio

class GeoTiffDataset(Dataset):
```

```
            def     __init__(self,     image_paths,     mask_paths,
transform=None):
        self.image_paths = image_paths
        self.mask_paths = mask_paths
        self.transform = transform

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        with rasterio.open(self.image_paths[idx]) as src:
            img = src.read().astype(np.float32)

        # Máscara 1-banda
        with rasterio.open(self.mask_paths[idx]) as msk:
            mask = msk.read(1).astype(np.int64)

        # Isso é utilizado para N-bandas (RGB, NIR, etc.)
          if img.max() > 1.0: # Evita normalizar se já estiver
normalizado, poupa tempo
            for c in range(img.shape[0]):
                min_val = img[c].min()
                max_val = img[c].max()
                      img[c] = (img[c] - min_val) / (max_val -
min_val + 1e-8)

        # Converte máscara para binária
        mask = np.where(mask > 0, 1, 0)

                return  torch.tensor(img,  dtype=torch.float32),
torch.tensor(mask, dtype=torch.long)
```

5.2 Model.py

```
import torch
```

```
import torch.nn as nn



class DoubleConv(nn.Module):

    def __init__(self, in_channels, out_channels):

        super().__init__()

        self.conv = nn.Sequential(

            nn.Conv2d(in_channels, out_channels, 3, padding=1),

            nn.BatchNorm2d(out_channels),

            nn.ReLU(inplace=True),

                        nn.Conv2d(out_channels,  out_channels,  3,
padding=1),

            nn.BatchNorm2d(out_channels),

            nn.ReLU(inplace=True)

        )



    def forward(self, x):

        return self.conv(x)

class UNet(nn.Module):

    def __init__(self, n_classes=2):

        super().__init__()



        self.down1 = DoubleConv(4, 64)
```

```
        self.pool1 = nn.MaxPool2d(2)
        self.down2 = DoubleConv(64, 128)
        self.pool2 = nn.MaxPool2d(2)
        self.down3 = DoubleConv(128, 256)
        self.pool3 = nn.MaxPool2d(2)

        self.bottleneck = DoubleConv(256, 512)

        self.up3 = nn.ConvTranspose2d(512, 256, 2, stride=2)
        self.conv3 = DoubleConv(512, 256)
        self.up2 = nn.ConvTranspose2d(256, 128, 2, stride=2)
        self.conv2 = DoubleConv(256, 128)
        self.up1 = nn.ConvTranspose2d(128, 64, 2, stride=2)
        self.conv1 = DoubleConv(128, 64)

        # Saída com 2 classes (0 = floresta, 1 = desmatado)
        self.out = nn.Conv2d(64, n_classes, kernel_size=1)

    def forward(self, x):
        c1 = self.down1(x)
        p1 = self.pool1(c1)
```

```
        c2 = self.down2(p1)
        p2 = self.pool2(c2)

        c3 = self.down3(p2)
        p3 = self.pool3(c3)

        bn = self.bottleneck(p3)

        u3 = self.up3(bn)
        c3 = torch.cat([u3, c3], dim=1)
        c3 = self.conv3(c3)

        u2 = self.up2(c3)
        c2 = torch.cat([u2, c2], dim=1)
        c2 = self.conv2(c2)

        u1 = self.up1(c2)
        c1 = torch.cat([u1, c1], dim=1)
        c1 = self.conv1(c1)

        out = self.out(c1)
        return out
```

5.3 Train.py

```
import torch



# Função de treino
def   train_epoch(model,   dataloader,   optimizer,   criterion,
device):
   model.train()
   total_loss = 0

   for imgs, masks in dataloader:
       imgs, masks = imgs.to(device), masks.to(device)
       optimizer.zero_grad()

       outputs = model(imgs)

       outputs = torch.nn.functional.interpolate(
               outputs, size=masks.shape[1:], mode='bilinear',
align_corners=False
       )

       loss = criterion(outputs, masks)
       loss.backward()
       optimizer.step()
       total_loss += loss.item()

   return total_loss / len(dataloader)



# Função de validação
```

```
def validate_epoch(model, dataloader, criterion, metrics,
device):
    model.eval()
    total_loss, total_iou, total_dice = 0, 0, 0

    with torch.no_grad():
        for imgs, masks in dataloader:
            imgs, masks = imgs.to(device), masks.to(device)
            outputs = model(imgs)

            outputs = torch.nn.functional.interpolate(
                outputs, size=masks.shape[1:], mode='bilinear',
align_corners=False
            )

            loss = criterion(outputs, masks)
            total_loss += loss.item()
            total_iou += metrics["iou"](outputs, masks)
            total_dice += metrics["dice"](outputs, masks)

    n = len(dataloader)
    return total_loss / n, total_iou / n, total_dice / n
```

5.4 Metrics.py

```
import torch



def iou_score(outputs, masks, eps=1e-6):

    preds = outputs.argmax(dim=1).float()

    masks = masks.float()

    inter = (preds * masks).sum(dim=(1,2))
```

```
    union = preds.sum(dim=(1,2)) + masks.sum(dim=(1,2)) - inter
    return ((inter+eps) / (union+eps)).mean().item()


def dice_score(outputs, masks, eps=1e-6):
    preds = outputs.argmax(dim=1).float()
    masks = masks.float()
    inter = (preds * masks).sum(dim=(1,2))
        return   ((2*inter+eps)   /   (preds.sum(dim=(1,2))   +
masks.sum(dim=(1,2)) + eps)).mean().item()
```

5.5 Evaluate.py

```
import torch
import rasterio
import numpy as np
from sklearn.metrics import confusion_matrix



# Função para gerar máscara predita, ou seja validação
def predict_mask(model, tif_path, device, threshold=0.5):
    model.eval()
    with rasterio.open(tif_path) as src:
        img = src.read().astype(np.float32)
```

```
    if img.max() > 1.0:
        for c in range(img.shape[0]):
            min_val = img[c].min()
            max_val = img[c].max()
             img[c] = (img[c] - min_val) / (max_val - min_val + 1e-8)

    img_tensor = torch.tensor(img).unsqueeze(0)

    with torch.no_grad():
        pred = model(img_tensor.to(device))

     # Caso binário (1 canal), garantia que masks estarão no padrão
    if pred.shape[1] == 1:
        pred_mask = torch.sigmoid(pred).squeeze().cpu().numpy()
        binary_mask = (pred_mask > threshold).astype(np.uint8)

    # Caso multiclasses (n_classes=2)
    else:
                                            pred_mask       = pred.argmax(dim=1).squeeze().cpu().numpy().astype(np.uint8)
```

```
        binary_mask = pred_mask


    return binary_mask




# Função para comparar máscara predita x real, comparação entres masks
def evaluate_prediction(pred_mask, true_mask):
    pred = pred_mask.flatten()
    true = true_mask.flatten()


    try:
            tn, fp, fn, tp = confusion_matrix(true, pred, labels=[0, 1]).ravel()
    except ValueError:
        # Utilizado para caso só exista uma classe na imagem
        if np.all(true == 0):
            tn, fp, fn, tp = len(true), 0, 0, 0
        elif np.all(true == 1):
            tn, fp, fn, tp = 0, 0, 0, len(true)
        else:
            return {"accuracy": 0, "iou": 0, "dice": 0}
```

```
    accuracy = (tp + tn) / (tp + tn + fp + fn + 1e-6)
    iou = tp / (tp + fp + fn + 1e-6)
    dice = (2 * tp) / (2 * tp + fp + fn + 1e-6)

    return {"accuracy": accuracy, "iou": iou, "dice": dice}
```

5.6 Main.py

```
import os, glob
import torch
import torch.optim as optim
import torch.nn as nn
from torch.utils.data import DataLoader
from sklearn.model_selection import train_test_split
import rasterio
import numpy as np
import matplotlib.pyplot as plt
from random import choice

from dataset import GeoTiffDataset
from model import UNet
from train import train_epoch, validate_epoch
from metrics import iou_score, dice_score
from evaluate import predict_mask, evaluate_prediction



# Função de Visualização
def visualize_prediction(model, dataset, device, idx=0):
```

```
    model.eval()
    img, mask = dataset[idx]

    with torch.no_grad():
        output = model(img.unsqueeze(0).to(device))
        pred = output.argmax(dim=1).squeeze().cpu().numpy()

    img_np = np.transpose(img.numpy(), (1, 2, 0))  # Shape (H,
W, 4)
    mask_np = mask.numpy()

    img_to_show = img_np[:, :, :3]

    img_to_show = np.clip(img_to_show, 0, 1)

    fig, axs = plt.subplots(1, 3, figsize=(12, 4))
    axs[0].imshow(img_to_show);
    axs[0].set_title("Imagem (RGB)")
    axs[1].imshow(mask_np, cmap="gray");
    axs[1].set_title("Máscara real")
    axs[2].imshow(pred, cmap="gray");
    axs[2].set_title("Predição")
    for ax in axs: ax.axis("off")
    plt.show()



# Configurações para o treinamento
image_dir = "data/images/"
mask_dir = "data/masks/"
batch_size = 4
epochs = 100
lr = 1e-4
```

```
# Definição de dataset
image_paths = sorted(glob.glob(os.path.join(image_dir,
"*.tif")))
mask_paths = sorted(glob.glob(os.path.join(mask_dir,
"*.tif")))

# Separa as imagens em treinamento, validação e teste
train_imgs, test_imgs, train_masks, test_masks =
train_test_split(image_paths, mask_paths, test_size=0.2,

random_state=42)
train_imgs, val_imgs, train_masks, val_masks =
train_test_split(train_imgs, train_masks, test_size=0.25,

random_state=42)

train_dataset = GeoTiffDataset(train_imgs, train_masks)
val_dataset = GeoTiffDataset(val_imgs, val_masks)

train_loader = DataLoader(train_dataset,
batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size)

# Especificações para o modelo
device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
model = UNet(n_classes=2).to(device)
optimizer = optim.Adam(model.parameters(), lr=lr)
criterion = nn.CrossEntropyLoss()
metrics = {"iou": iou_score, "dice": dice_score}

# Importe de pesos do modelo já treinado
```

```
model.load_state_dict(torch.load("unet_deforestation_4band.pth
", map_location=device))

# Treinamento

# Sessão a seguir está como comentada por conta que o modelo
já foi treinado, caso queira fazer um novo treinamento,
comente os pesos anteriores e descomente essa sessão
#print("Iniciando treinamento...")
#for epoch in range(epochs):
#     train_loss = train_epoch(model, train_loader, optimizer,
criterion, device)
#        val_loss, val_iou, val_dice = validate_epoch(model,
val_loader, criterion, metrics, device)
#    print(
#                f"Epoch {epoch + 1}/{epochs} | Train Loss:
{train_loss:.4f}   |   Val   Loss:   {val_loss:.4f}   |   IoU:
{val_iou:.3f} | Dice: {val_dice:.3f}")

#torch.save(model.state_dict(),
"unet_deforestation_4band.pth")
#print("Modelo salvo em unet_deforestation_4band.pth")


# Teste em imagens novas

print("\n=== Avaliação no conjunto de teste ===")
all_acc, all_iou, all_dice = [], [], []

for img_path, mask_path in zip(test_imgs, test_masks):
   pred_mask = predict_mask(model, img_path, device)
   with rasterio.open(mask_path) as src:
       true_mask = src.read(1).astype(np.uint8)
       true_mask = np.where(true_mask > 0, 1, 0)
```

```
    m = evaluate_prediction(pred_mask, true_mask)
    all_acc.append(m["accuracy"])
    all_iou.append(m["iou"])
    all_dice.append(m["dice"])

print(f"Acurácia média: {np.mean(all_acc):.3f}")
print(f"IoU médio: {np.mean(all_iou):.3f}")
print(f"Dice médio: {np.mean(all_dice):.3f}")



# Visualização de exemplo no teste
print("\n=== Visualizando exemplo de Teste ===")
test_dataset = GeoTiffDataset(test_imgs, test_masks)
idx = choice(range(len(test_dataset)))

# Chama a o dataset Teste para exibição
visualize_prediction(model, test_dataset, device, idx=idx)
```

## 6. CONCLUSÃO

A criação de um Sistema de análise de imagens de satélites, mais especificamente imagens da Amazônia, se provou uma tarefa difícil, ainda mais com a necessidade de identificar um dataset com imagens suficientes para o treinamento e que não possuísse muito ruído (nuvens ou sombras nas imagens) além da grande quantidade de épocas para treinamento usando uma placa de vídeo dedicada para isso.

Entretanto, além dessas adversidades, houve um resultado proveitoso na análise das imagens e criação de máscaras para validação do funcionamento do sistema, tendo uma taxa de acurácia média entre 85% a 95% em testes finais.

**REFERÊNCIAS**

ALFALOU, A. et al. A comparative study of CFs, LBP, HOG, SIFT, SURF, and BRIEF for security and face recognition. [S.l.: s.n.].

BARBOSA, X. de C.; BEZERRA, R. F. Breve introdução à história da inteligência artificial. [S.l.: s.n.]. Acesso em: 11 out. 2025.

BADRINARAYAN, V.; KENDALL, A.; CIPOLLA, R. SegNet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 39, n. 12, p. 2481–2495, 2017.

BRAGAGNOLO, Lucimara; DA SILVA, Roberto Valmir; GRZYBOWSKI, José Mario Vicensi. Amazon and Atlantic Forest image datasets for semantic segmentation. 2021. Disponível em: https://doi.org/10.5281/zenodo.4498086 . Acesso em: 21 out. 2025.

CHEN, L.; PAPANDREOU, G.; KOKKINOS, I.; MURPHY, K.; YUILLE, A. DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 40, n. 4, p. 834–848, 2018.

COSTA, G.; COUTO, D. Inteligência artificial aplicada ao reconhecimento de imagens. [S.l.: Escola Politécnica da Universidade de São Paulo], 2008.

DE CASTRO BARBOSA, X.; FERREIRA BEZERRA, R. Breve introdução à história da inteligência artificial. [S.l.: s.n.]. Acesso em: 11 out. 2025.

GAVA, M. Pesquisa sobre IA: 96% dos usuários de IA generativa apoiam a implementação de diretrizes para regular seu uso no trabalho. Disponível em: https://www.getapp.com.br/blog/4175/pesquisa-ia#Um-de-cada-quatro-empregam-a-IA-generativa-todos-os-dias

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

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, v. 521, n. 7553, p. 436–444, 27 maio 2015.

MITCHELL, T. M. *Machine Learning*. New York: McGraw-Hill, 1997.

O'SHEA, K.; NASH, R. An introduction to convolutional neural networks. [S.l.: s.n.].

RONNEBERGER, O.; FISCHER, P.; BROX, T. U-Net: Convolutional networks for biomedical image segmentation. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Cham: Springer, 2015.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3. ed. New Jersey: Pearson, 2010.

SESHU, V. PCA vs LDA — No more confusion! Disponível em: https://medium.com/%40seshu8hachi/pca-vs-lda-no-more-confusion-fc21fb8d06e9

SHETTY, D. et al. Diving deep into deep learning: History, evolution, types and applications. *International Journal of Innovative Technology and Exploring Engineering*, v. 9, n. 3, p. 1–13, 30 jan. 2020.

SOUZA, R. Estudo do algoritmo SIFT (Scale Invariant Feature Transform). [S.l.: Pontifícia Universidade Católica de Goiás], 2024.

SZELISKI, R. *Computer Vision: Algorithms and Applications*. 2. ed. [S.l.: Springer Nature], 2020.

TRASK, A. W. *Grokking Deep Learning*. Shelter Island, NY: Manning, 2019.

TURING, A. M. *Computing Machinery and Intelligence*. Oxford: Oxford University Press, 1950.