

Relatório Técnico Final: Sistema de Análise e Classificação de Lesões de Pele (HAM10000)

Contexto: Trabalho Final – Fundamentos de Big Data

Repositório: <https://github.com/Big-Data-Analise-de-Cancer-de-Pele/Trabalho-Final---Big-Data/tree/main>

1. Identificação da Equipe e Responsabilidades

A execução do projeto foi dividida conforme as competências técnicas dos integrantes:

- **Miguel Allievi:** Desenvolvimento dos gráficos e visualizações (Dashboard/EDA).
- **Sávio Nery:** Infraestrutura Docker, criação do `README.md` e orquestração do ambiente.
- **João Gabriel Pereira:** Seleção e limpeza prévia da base de dados e tentativa inicial de modelo.
- **Eduardo Fleming:** Tratamento avançado de dados, normalização e organização do GitHub.
- **Pedro Vargas:** Elaboração do relatório técnico, escolha final da base e definição dos parâmetros.

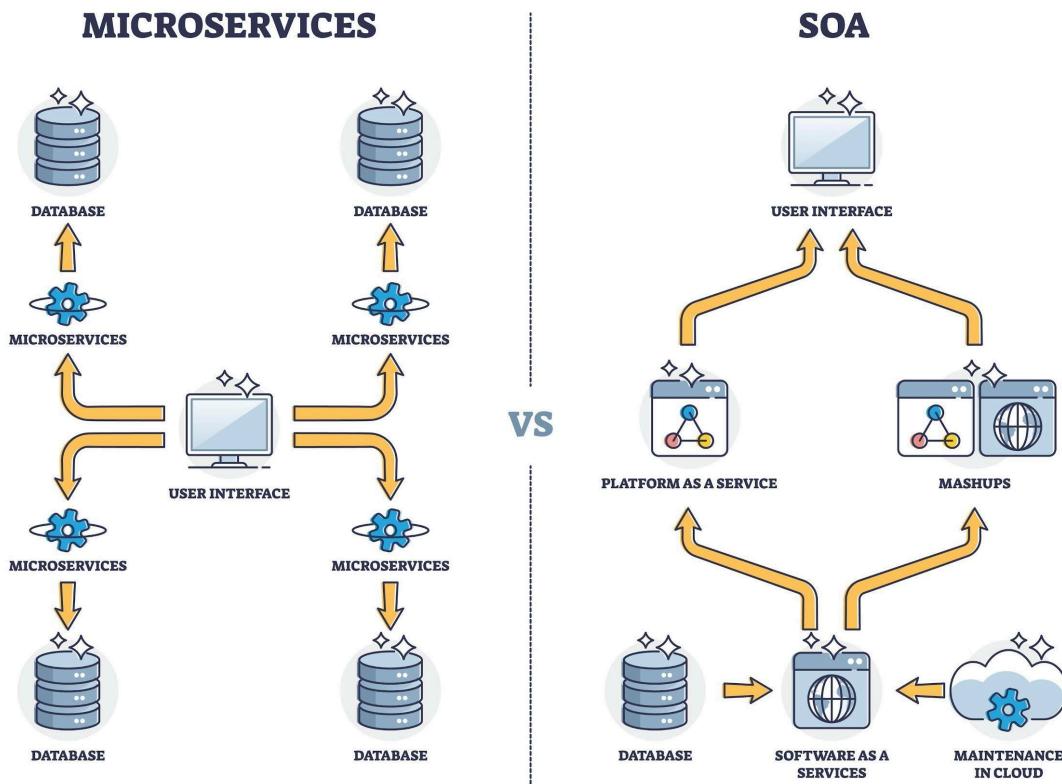
2. Propósito do Projeto

Este projeto consiste no desenvolvimento de uma arquitetura de Big Data e Machine Learning para o suporte ao diagnóstico dermatológico. Utilizando o dataset **HAM10000** (*Human Against Machine with 10000 training images*), a solução implementa um pipeline completo que vai desde a ingestão de dados em um Data Lake (Object Storage) até a disponibilização de uma interface interativa para treinamento de modelos e inferência.

O diferencial técnico reside na containerização completa via **Docker**, garantindo reproduzibilidade, e na integração com **MinIO** para simular um ambiente de nuvem (S3-compatible) para armazenamento de grandes volumes de dados.

3. Arquitetura da Solução

A solução utiliza uma arquitetura de microserviços orquestrada via Docker, garantindo portabilidade e isolamento de dependências. A estrutura divide-se nas seguintes camadas:



1. **Camada de Persistência (Data Lake):** Utiliza o **MinIO**. Atua como repositório central (Object Store), armazenando os arquivos CSV brutos e metadados. Simula um servidor S3 rodando em container.
2. **Camada de Aplicação e Apresentação:** Utiliza o **Streamlit**. Responsável pela interface visual do usuário, gestão do estado da sessão e geração de gráficos. Consome os dados diretamente do bucket MinIO usando o client oficial [minio-py](#).
3. **Camada de Processamento e Modelagem:** Executada no backend, utiliza **Pandas** para manipulação de dataframes e **TensorFlow/Keras** para a construção e treinamento da Rede Neural Convolucional (CNN).
4. **Infraestrutura: Docker & Docker Compose.** Gerenciam o ciclo de vida dos serviços, redes internas e volumes persistentes.

4. Componentes Implementados e Análise

A. Ingestão e Integração (MinIO)

O sistema conecta-se ao serviço MinIO (porta 9000 interna) para recuperar os datasets.

- **Bucket:** [datasets](#).
- **Dados:** Imagens achatadas em CSV (ex: [hmnist_28_28_L.csv](#)) e metadados clínicos.

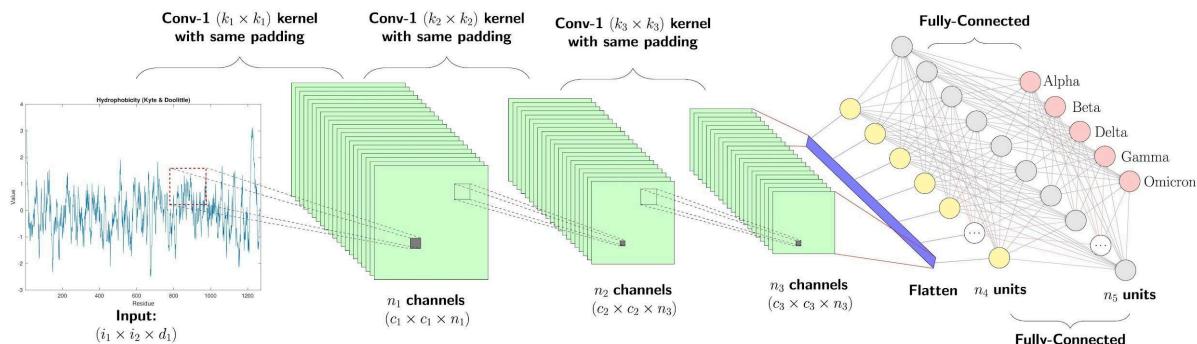
- **Resiliência:** Implementação de tratamento de erros para carregar dados em memória ([BytesIO](#)) caso o bucket esteja inacessível.

B. Análise Exploratória de Dados (EDA)

Com base na prototipagem, foram implementadas visualizações cruciais para o entendimento clínico:

- **Distribuição de Diagnósticos:** Identificação de desbalanceamento de classes, com prevalência de Nevos Melanocíticos ([nv](#)) e Queratose Benigna ([bk1](#)) sobre lesões graves como Melanoma ([me1](#)) e Carcinoma ([bcc](#)).
- **Perfil Demográfico:**
 - **Idade:** A análise por bins de 10 anos revelou uma concentração maior de casos entre 40 e 60 anos, com queda significativa após os 80 anos.
 - **Gênero:** Visualização da proporção entre casos masculinos e femininos.
- **Correlação Patologia vs. Localização:**
 - **Grupo 1 (Alto Volume):** Análise focada em Nevos e Melanomas, mapeando sua incidência predominante em áreas como tronco e dorso.
 - **Grupo 2 (Baixo Volume):** Foco em Carcinomas e lesões Vasculares, permitindo identificar padrões específicos em regiões como face e extremidades.

C. Módulo de Machine Learning (CNN)



Foi implementada uma Rede Neural Convolutacional (CNN) utilizando a API Keras/TensorFlow.

- **Pré-processamento:** Reshape dos vetores para tensores de imagem (28x28), normalização (divisão por 255.0) e One-Hot Encoding das 7 classes alvo .
- **Arquitetura da Rede:**
 - **Conv2D:** 32 filtros, kernel 3x3, ativação ReLU (Extração de características primárias).
 - **MaxPooling2D:** Redução de dimensionalidade.
 - **Conv2D:** 64 filtros, kernel 3x3, ativação ReLU (Extração de características complexas).
 - **MaxPooling2D:** Nova redução.

- **Flatten & Dense:** Camada densa de 128 neurônios com **Dropout (0.3)** para evitar o overfitting.
- **Output:** Camada densa com ativação **Softmax** para classificação multiclasse.
-
- **Performance (Baseline):**
 - O modelo protótipo atingiu uma **acurácia de ~70.6%** no conjunto de teste após 10 épocas.
 - Observou-se convergência estável entre as curvas de treino e validação, indicando boa generalização inicial, com potencial de melhoria via *Data Augmentation*.

5. Instruções de Execução e Deploy

Pré-requisitos: Docker e Docker Compose instalados; Dataset HAM10000 baixado localmente.

Passo a Passo

1. Limpeza e Build:

Para evitar conflitos de portas ou containers antigos, execute o reset completo:

Bash

```
docker container prune
```

2. Configuração do Diretório:

Baixe o repositório e substitua a pasta **streamlit** no diretório de trabalho:

```
cd /opt/ceub-bigdata/streamlit).
```

3. Subir o ambiente com Docker Compose:

```
docker-compose up -d
```

4. Ajustar portas na VirtualMachine (caso necessário):

Se houver algo usando a porta 8501 (geralmente Flask), derrube o serviço:

Garanta que a VM está expondo a porta:

```
Streamlit 8501 -> 8501
```

5. Acessar o MinIO: Abra o navegador:

```
http:// localhost:9001
```

6. Criar Bucket e enviar o dataset HAM10000

6.1. Baixar dataset HAM10000 (CSV)

Google Drive: https://drive.google.com/drive/folders/1xGpaP8dTziaH_kZ5RxjmhL_AYW_PwNfsZ?usp=sharing

6.2 Criar Bucket datasets (se não existir)

Acesse:

<http://localhost:9001/browser>

E crie o bucket:

datasets

6.3 Fazer upload dos arquivos CSV ou MinIO

Acesse:

<http://localhost:9001/browser/datasets>

Faça upload dos arquivos do dataset

7. Abrir o Streamlit

Depois de enviar os arquivos ao MinIO, abra:

<http://localhost:8501/>

A aplicação deverá carregar normalmente.

DEBUG para Devs – Caso algo dê errado

Utilize:

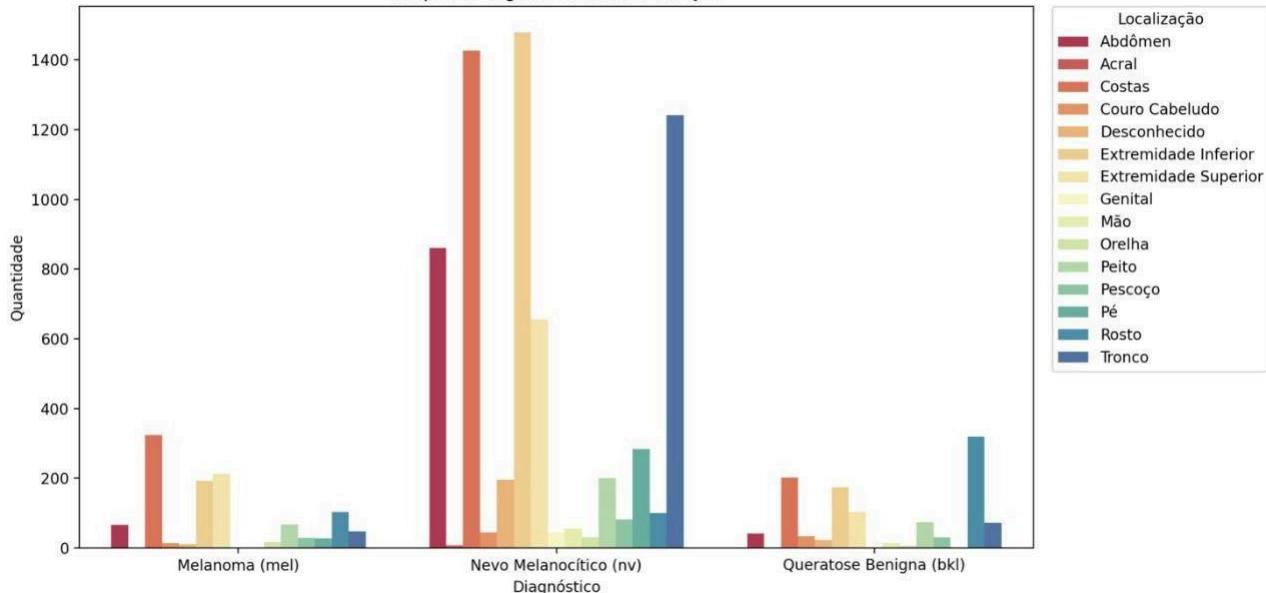
```
docker-compose down  
docker-compose build  
docker-compose up -d  
docker logs -f streamlit -app
```

☒ Dicionário de Dados (HAM10000)

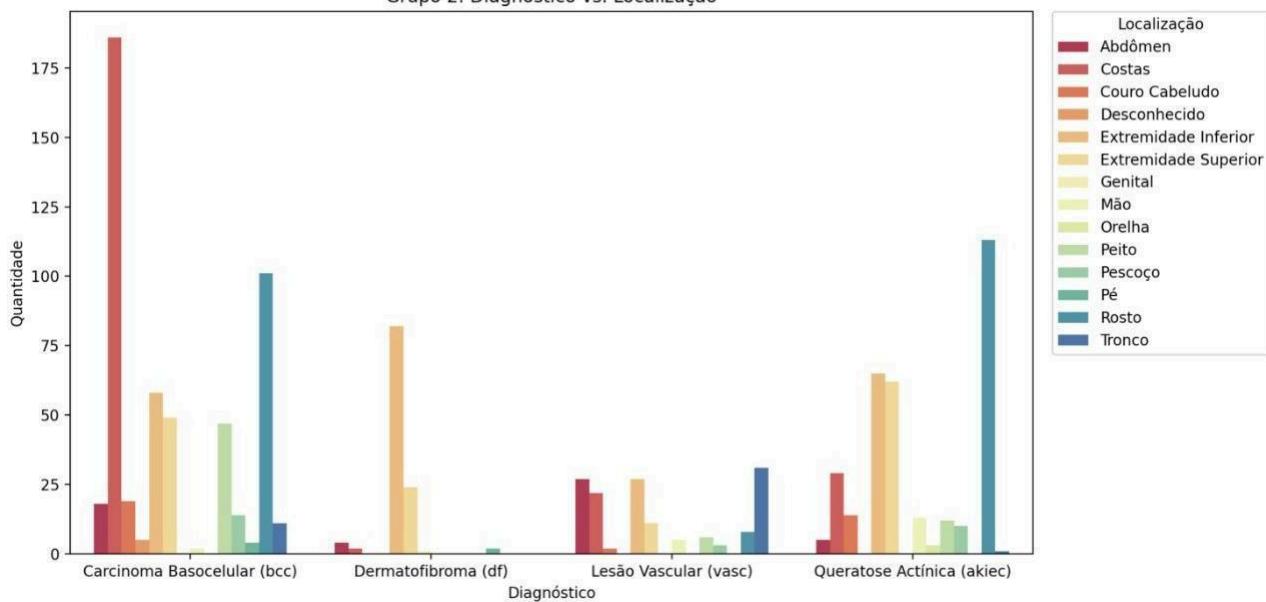
Para entender as colunas do dataset:

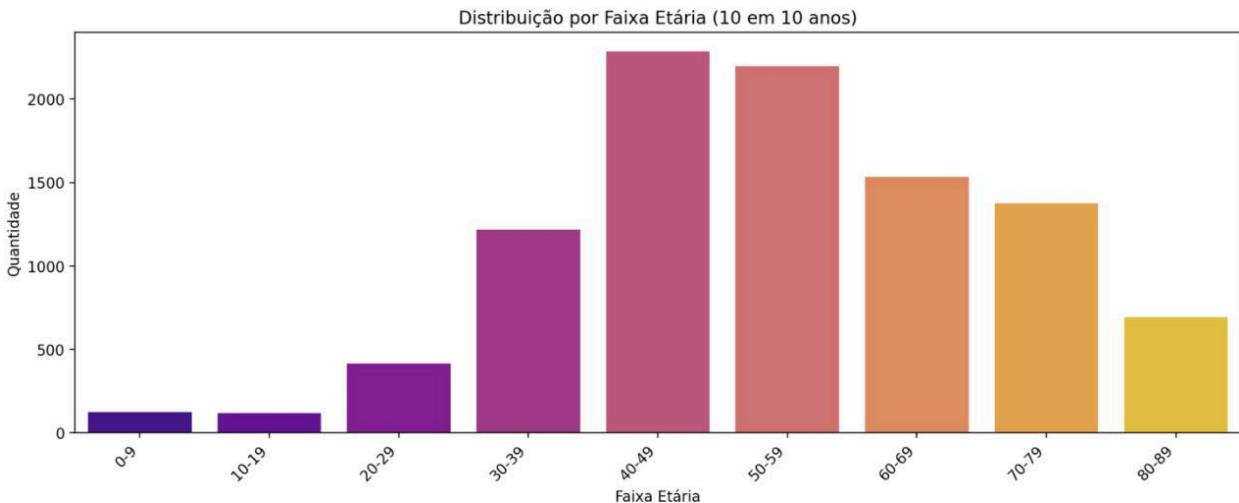
<https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/DBW86T>

Grupo 1: Diagnóstico vs. Localização



Grupo 2: Diagnóstico vs. Localização





6. Conclusão

A implementação deste ecossistema de análise dermatológica valida a eficácia de uma arquitetura moderna de dados, desacoplando a camada de persistência (MinIO) da camada de processamento e inferência. O projeto transcende a simples modelagem de algoritmos, entregando uma **solução end-to-end funcional**, containerizada e pronta para escalabilidade horizontal.

Principais Resultados Alcançados

- Validação da Arquitetura:** A integração entre **Streamlit** e **MinIO** provou ser robusta, permitindo o manuseio eficiente de datasets complexos (metadados + tensores de imagem) sem sobrecarregar a memória da aplicação, simulando um ambiente real de *Data Lake* em nuvem.
- Performance do Modelo (Baseline):** O modelo CNN customizado estabeleceu uma linha de base sólida com **70.64% de acurácia**. A ausência de *overfitting* severo nas curvas de aprendizado indica que o pipeline de pré-processamento está correto, embora a arquitetura atual (CNN simples) tenha atingido seu limite de capacidade de generalização para este dataset desbalanceado.
- Insights Clínicos:** A análise exploratória automatizada (EDA) forneceu valor imediato ao identificar o viés demográfico (concentração em faixas etárias de 40-60 anos) e a predominância de classes benignas (*Nevos*), fatores cruciais para a calibração futura de modelos em ambiente clínico.