

▼ Trabalho Final IAA011

Participantes:

- Adilson Krischanski
- Ana de Vasconcellos Oporto
- Pedro Augusto Pereira H. dos Santos
- Renan Belem Biavati
- Saulo Roberto da Silva

```
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.12/dist-packages (2.19.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (25.9.23)
Requirement already satisfied: gast!=0.5.0,!>0.5.1,!>0.5.2,>=0.2.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from tensorflow) (25.0)
Requirement already satisfied: protobuf!=4.21.0,!>4.21.1,!>4.21.2,!>4.21.3,!>4.21.4,!>4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.20.3)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.32.4)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.1.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (4.15.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.0.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.75.1)
Requirement already satisfied: tensorboard~>2.19.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.19.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.10.0)
Requirement already satisfied: numpy<2.2.0,>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.15.1)
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.5.3)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from astunparse>=1.6.0->tensorflow) (0.45.1)
Requirement already satisfied: rich in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (0.1.0)
Requirement already satisfied: optree in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (0.17.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (2025.10.5)
```

```
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.12/dist-packages (from tensorflow~2.19.0->tensorflow) (3.9)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow~2.19.0->tensorflow)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow~2.19.0->tensorflow) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.12/dist-packages (from werkzeug>=1.0.1->tensorflow~2.19.0->tensorflow)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.12/dist-packages (from rich->keras>=3.5.0->tensorflow) (4.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.12/dist-packages (from rich->keras>=3.5.0->tensorflow) (2.19.2)
Requirement already satisfied: mdurl~0.1 in /usr/local/lib/python3.12/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensorflow)
```

```
# --- Bibliotecas Padrão do Python ---
import os
import re
import math as mt
import zipfile
from typing import Dict, Any, Tuple

# --- Bibliotecas de Manipulação de Dados e Arrays ---
import numpy as np
import pandas as pd
import joblib # Para salvar/carregar modelos

# --- Bibliotecas de Visualização e Imagem ---
from matplotlib import pyplot as plt
import seaborn as sns
from PIL import Image
from google.colab.patches import cv2_imshow # Específico do Colab/OpenCV
import cv2

# --- Bibliotecas de Processamento de Imagem (Scikit-Image) ---
from skimage import io, feature, color

# --- Bibliotecas de Deep Learning (TensorFlow/Keras) ---
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.applications import VGG16, ResNet50
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.applications.resnet50 import preprocess_input as resnet50_preprocess
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.applications.vgg16 import preprocess_input as vgg16_preprocess
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam

# --- Bibliotecas de Machine Learning (Scikit-learn) ---
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
from sklearn.preprocessing import StandardScaler, LabelEncoder
```

```
from sklearn.ensemble import RandomForestClassifier  
from sklearn.svm import SVC  
from sklearn.neural_network import MLPClassifier  
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
```

1 Extração de Características

Os bancos de imagens fornecidos são conjuntos de imagens de 250x250 pixels de imuno-histoquímica (biópsia) de câncer de mama. No total são 4 classes (0, 1+, 2+ e 3+) que estão divididas em diretórios. O objetivo é classificar as imagens nas categorias correspondentes. Uma base de imagens será utilizada para o treinamento e outra para o teste do treino.

As imagens fornecidas são recortes de uma imagem maior do tipo WSI (Whole Slide Imaging) disponibilizada pela Universidade de Warwick ([link](#)). A nomenclatura das imagens segue o padrão XX_HER_YYYY.png, onde XX é o número do paciente e YYYY é o número da imagem recortada. Separe a base de treino em 80% para treino e 20% para validação. Separe por pacientes (XX), não utilize a separação randômica! Pois, imagens do mesmo paciente não podem estar na base de treino e de validação, pois isso pode gerar um viés. No caso da CNN VGG16 remova a última camada de classificação e armazene os valores da penúltima camada como um vetor de características. Após o treinamento, os modelos treinados devem ser validados na base de teste. Tarefas:

1. Carregue a base de dados de Treino.
2. Crie partições contendo 80% para treino e 20% para validação (atenção aos pacientes).
3. Extraia características utilizando LBP e a CNN VGG16 (gerando um csv para cada extrator).
4. Treine modelos Random Forest, SVM e RNA para predição dos dados extraídos (nessa tarefa utilize todas as imagens para o treinamento).
5. Carregue a base de Teste e execute a tarefa 3 nesta base.
6. Aplique os modelos treinados nos dados de teste.
7. Calcule as métricas de Sensibilidade, Especificidade e F1-Score com base em suas matrizes de confusão.
8. Indique qual modelo dá o melhor resultado e a métrica utilizada

✓ 1. Carregue a base de dados de Treino.

```
ZIP_FILE_PATH = '/content/Train_Warwick.zip'  
# O ZIP deve extrair as pastas de classe (0, 1, 2, 3) DENTRO DESTA PASTA.  
# Se o ZIP extrai /content/Train_Warwick/Train_4cls_amosta/0, /content/Train_Warwick/Train_4cls_amosta/1 etc.  
TRAIN_DIR = '/content/Train_Warwick/Train_4cls_amosta/'
```

```

# Classes adaptadas para a notação 0, 1, 2, 3 (em vez de 0+, 1+, 2+, 3+)
CLASSES = ['0', '1', '2', '3']

OUTPUT_DIR = 'features_csv' # Diretório para salvar os CSVs
os.makedirs(OUTPUT_DIR, exist_ok=True)

# Parâmetros LBP (mantidos do código anterior):
RADIUS = 3
N_POINTS = 8 * RADIUS
METHOD = 'uniform'
IMG_SIZE = (250, 250)

# Regex para extrair o Patient ID: XX_HER_YYYY.png -> XX
patient_regex = re.compile(r"(\d+)_HER2_.*\.png")

# Função Auxiliar para Extrair Patient ID
def extract_patient_id(filename):
    """Extrai o ID do paciente (XX) da nomenclatura XX_HER_YYYY.png."""
    match = patient_regex.match(filename)
    if match:
        return match.group(1) # O grupo 1 é o XX
    return None

# --- Etapa 1: Descompactação e Carregamento (Adaptado do novo código) ---

def setup_and_load_data(zip_path, train_dir, classes):
    """Descompacta o ZIP e mapeia o dataset para um DataFrame."""

    # --- 1. Descompactação do Arquivo ZIP ---
    if not os.path.exists(zip_path):
        print(f"ERRO: Arquivo ZIP não encontrado em: {zip_path}")
        return pd.DataFrame() # Retorna DataFrame vazio em caso de erro

    # Cria o diretório de destino se ele não existir
    if not os.path.exists(train_dir):
        os.makedirs(train_dir, exist_ok=True)

    print(f"Descompactando {zip_path}...")
    try:
        # Extrai para '/content/' (ou a pasta base)
        # Assumimos que o ZIP contém o caminho interno correto para o TRAIN_DIR
        with zipfile.ZipFile(zip_path, 'r') as zip_ref:
            # Extrai o conteúdo do ZIP para /content/
            zip_ref.extractall('/content/')

```

```
    print("Descompactação concluída.")
except Exception as e:
    print(f"ERRO durante a descompactação: {e}")
    return pd.DataFrame()

# --- 2. Mapear o Dataset de Treino e Extrair Pacientes ---
train_files = []
print(f"\nIniciando mapeamento em: {train_dir}")

for cls in classes:
    class_path = os.path.join(train_dir, cls)

    if os.path.exists(class_path):
        for filename in os.listdir(class_path):
            if filename.endswith('.png'):
                patient_id = extract_patient_id(filename) # Usando a regex definida

                if patient_id:
                    train_files.append({
                        'filepath': os.path.join(class_path, filename),
                        'class': cls,
                        'patient_id': patient_id # Mantemos 'patient_id'
                    })
                else:
                    # Arquivo .png que não segue o padrão XX_HER_YYYY.png
                    print(f"Aviso: Arquivo {filename} em {cls} não seguiu o padrão de nomeação.")
    else:
        print(f"Aviso: Diretório de classe não encontrado: {class_path}")

# DataFrame de Treino
df = pd.DataFrame(train_files)

if df.empty:
    print("\nERRO: Nenhum arquivo de imagem encontrado.")
else:
    print(f"\nTotal de imagens de Treino carregadas: {len(df)}")
    print(f"IDs de Pacientes únicos no Treino: {df['patient_id'].nunique()}")
    print("Mapeamento concluído.")

return df
```

2. Crie partições contendo 80% para treino e 20% para validação (atenção aos pacientes).

```
def split_data_by_patient(df_full, test_size=0.2, random_state=42):
    """
    Separa o DataFrame em treino e validação (80/20) baseado no patient_id,
    usando sklearn.model_selection.train_test_split.

    """
    patient_ids = df_full['patient_id'].unique()

    # 1. Separar os IDs de Pacientes em Treino e Validação
    train_patient_ids, val_patient_ids = train_test_split(
        patient_ids,
        test_size=test_size,
        random_state=random_state
    )

    print(f"--- Separação por Paciente ({int((1-test_size)*100)}% Treino / {int(test_size*100)}% Validação) ---")
    print(f"Total de pacientes únicos: {len(patient_ids)}")
    print(f"IDs de Pacientes no conjunto de Treino: {len(train_patient_ids)}")
    print(f"IDs de Pacientes no conjunto de Validação: {len(val_patient_ids)}")

    # 2. Criar os DataFrames Finais de Imagens
    df_train_set = df_full[df_full['patient_id'].isin(train_patient_ids)].reset_index(drop=True)
    df_val_set = df_full[df_full['patient_id'].isin(val_patient_ids)].reset_index(drop=True)

    print(f"Total de imagens no conjunto de Treino: {len(df_train_set)}")
    print(f"Total de imagens no conjunto de Validação: {len(df_val_set)}")

    # 3. Verificação de Integridade
    overlap = set(df_train_set['patient_id'].unique()) & set(df_val_set['patient_id'].unique())
    if not overlap:
        print("SUCESSO: Não há sobreposição de pacientes entre Treino e Validação.")
    else:
        print(f"ERRO DE SOBREPOSIÇÃO: {overlap}")

    return df_train_set, df_val_set
```

3. Extraia características utilizando LBP e a CNN VGG16 (gerando um csv para cada extrator).

(PARTE 1 - LBP)

```
def extract_lbp_features(df, img_size=IMG_SIZE):
    """Extrai o histograma LBP para cada imagem no DataFrame."""

    lbp_features = []
    lbp_labels = []

    print(f"Iniciando extração LBP para {len(df)} imagens...")

    for index, row in df.iterrows():
        try:
            # 1. Carregar e Redimensionar a Imagem (250x250)
            img_pil = Image.open(row['filepath']).convert('RGB')
            img_resized_pil = img_pil.resize(img_size)
            img_resized = np.array(img_resized_pil)

            # 2. Converter para Escala de Cinza
            img_normalized = img_resized.astype(float) / 255.0
            img_gray = color.rgb2gray(img_normalized)

            # 3. Calcular LBP
            lbp = feature.local_binary_pattern(img_gray, N_POINTS, RADIUS, method=METHOD)

            # 4. Calcular o Histograma
            hist, _ = np.histogram(lbp.ravel(), bins=np.arange(0, N_POINTS + 2), range=(0, N_POINTS + 1))

            # 5. Normalizar o Histograma
            hist = hist.astype("float")
            hist /= (hist.sum() + 1e-7)

            lbp_features.append(hist)
            lbp_labels.append(row['class'])

        except Exception as e:
            print(f"Erro ao processar imagem {row['filepath']} (Paciente {row['patient_id']}): {e}")

    return lbp_features, lbp_labels
```

```

X_lbp = np.array(lbp_features)
y_lbp = np.array(lbp_labels)

df_features = pd.DataFrame(X_lbp)
df_features['class'] = y_lbp

return df_features

# --- Execução Principal (Unindo todas as etapas) ---

if __name__ == "__main__":
    # 1. Carregue e Mapeie a base de dados de Treino
    print("--- 1. Carregando Base de Dados de Treino ---")
    df_full_train = setup_and_load_data(ZIP_FILE_PATH, TRAIN_DIR, CLASSES)

    if df_full_train.empty:
        print("Não foi possível continuar. DataFrame de Treino está vazio.")
    else:
        # 2. Crie partições (80/20) por paciente
        print("\n--- 2. Particionamento por Paciente (80/20) ---")
        df_train_set, df_val_set = split_data_by_patient(df_full_train, test_size=0.2, random_state=42)

        # 3. Extraia características utilizando LBP e salve em CSV

        # Extração LBP para o conjunto de Treino
        print("\n--- 3.1. Extração LBP para Treino ---")
        df_lbp_train = extract_lbp_features(df_train_set)
        output_path_train = os.path.join(OUTPUT_DIR, 'lbp_features_train.csv')
        df_lbp_train.to_csv(output_path_train, index=False)
        print(f"Características LBP (Treino) salvas em: {output_path_train} ({len(df_lbp_train)} amostras)")

        # Extração LBP para o conjunto de Validação
        print("\n--- 3.2. Extração LBP para Validação ---")
        df_lbp_val = extract_lbp_features(df_val_set)
        output_path_val = os.path.join(OUTPUT_DIR, 'lbp_features_val.csv')
        df_lbp_val.to_csv(output_path_val, index=False)
        print(f"Características LBP (Validação) salvas em: {output_path_val} ({len(df_lbp_val)} amostras)")

    print("\n--- Processo de Extração LBP (Treino/Validação) Concluído ---")

--- 1. Carregando Base de Dados de Treino ---
Descompactando /content/Train_Warwick.zip...
Descompactação concluída.

```

Iniciando mapeamento em: /content/Train_Warwick/Train_4cls_amostra/

Total de imagens de Treino carregadas: 593

IDs de Pacientes únicos no Treino: 20

Mapeamento concluído.

--- 2. Particionamento por Paciente (80/20) ---

--- Separação por Paciente (80% Treino / 20% Validação) ---

Total de pacientes únicos: 20

IDs de Pacientes no conjunto de Treino: 16

IDs de Pacientes no conjunto de Validação: 4

Total de imagens no conjunto de Treino: 475

Total de imagens no conjunto de Validação: 118

SUCESSO: Não há sobreposição de pacientes entre Treino e Validação.

--- 3.1. Extração LBP para Treino ---

Iniciando extração LBP para 475 imagens...

```
/usr/local/lib/python3.12/dist-packages/skimage/feature/texture.py:385: UserWarning: Applying `local_binary_pattern` to floating-point images may result in loss of information.
  warnings.warn(
  
```

Características LBP (Treino) salvas em: features_csv/lbp_features_train.csv (475 amostras)

--- 3.2. Extração LBP para Validação ---

Iniciando extração LBP para 118 imagens...

```
/usr/local/lib/python3.12/dist-packages/skimage/feature/texture.py:385: UserWarning: Applying `local_binary_pattern` to floating-point images may result in loss of information.
  warnings.warn(
  
```

Características LBP (Validação) salvas em: features_csv/lbp_features_val.csv (118 amostras)

--- Processo de Extração LBP (Treino/Validação) Concluído ---

3. Extraia características utilizando LBP e a CNN VGG16 (gerando um csv para cada extrator).

(PARTE 2 - VGG16)

```
# --- CONFIGURAÇÕES GLOBAIS FALTANTES/NECESSÁRIAS ---
TARGET_SIZE = (224, 224)
POOLING = 'avg' # <-- ESTA É A VARIÁVEL QUE FALTAVA
OUTPUT_DIR = 'features_csv'
# Certifique-se de que OUTPUT_DIR seja criado se não existir
os.makedirs(OUTPUT_DIR, exist_ok=True)
```

```
# Definição da função extract_vgg16_features_simplified
def extract_vgg16_features_simplified(df):
    """
    Extrai o vetor de características da última camada convolucional da VGG16
    seguido por Global Average Pooling (512 features).
    """

    # 1. Carregar o Modelo VGG16 Pré-treinado (SIMPLIFICADO)
    # include_top=False: Remove as camadas densas finais
    # pooling='avg': Aplica Global Average Pooling (média global) na saída da última camada convolucional (512 canais)
    print("Carregando VGG16 (include_top=False, pooling='avg') para 512 features...")
    # AGORA 'POOLING' ESTÁ DEFINIDO E O CÓDIGO FUNCIONARÁ
    model = VGG16(weights='imagenet', include_top=False, pooling=POOLING)

    # ... (o resto da função permanece o mesmo)
    # ...

    vgg_features = []
    vgg_labels = []

    print(f"Iniciando extração VGG16 para {len(df)} imagens...")

    for index, row in df.iterrows():
        try:
            # 1. Carregar e Redimensionar a Imagem (224x224)
            img_pil = Image.open(row['filepath']).convert('RGB')
            img_resized_pil = img_pil.resize(TARGET_SIZE)

            # 2. Converter para Array NumPy e Expandir Dimensões
            img_array = np.array(img_resized_pil)
            img_array = np.expand_dims(img_array, axis=0)

            # 3. Pré-processamento VGG16
            img_processed = preprocess_input(img_array)

            # 4. Extrair Características
            features = model.predict(img_processed, verbose=0).flatten()

            vgg_features.append(features)
            vgg_labels.append(row['class'])

        except Exception as e:
            print(f"Erro ao processar imagem {row['filepath']}: {e}")

    return vgg_features, vgg_labels
```

```

# Converte listas para arrays NumPy
X_vgg = np.array(vgg_features)
y_vgg = np.array(vgg_labels, dtype=str)

# Cria DataFrame para salvar em CSV
df_features = pd.DataFrame(X_vgg)
df_features['class'] = y_vgg

return df_features

# --- Bloco de Execução Principal (Adaptado) ---

if __name__ == "__main__":

    # --- Configurações Iniciais (Placeholder para Trecho 1 e 2) ---
    # ***** Assegure-se de que estas variáveis estejam populadas *****
    # Ponto de entrada para DataFrames df_train_set e df_val_set
    # Aqui, a função setup_and_load_data() e split_data_by_patient() teriam rodado.

    # **Simulação** (Remover quando integrar no script completo)
    # df_full_train = setup_and_load_data(ZIP_FILE_PATH, TRAIN_DIR, CLASSES)
    # df_train_set, df_val_set = split_data_by_patient(df_full_train, test_size=0.2, random_state=42)

    # Exemplo (DEVE SER SUBSTITUÍDO PELO SEU df_train_set e df_val_set REAIS):
    # Crie DataFrames de exemplo se você estiver testando apenas a função VGG16.
    # Se você está integrando, as variáveis df_train_set e df_val_set JÁ EXISTEM.

    if 'df_train_set' in locals() and not df_train_set.empty: # Verifica se df_train_set existe e não está vazio
        print("\n" + "*50)
        print("INÍCIO DA EXTRAÇÃO DE CARACTERÍSTICAS VGG16")
        print("*50)

        # Extração VGG16 para o conjunto de Treino
        print("\n--- 3.1. Extração VGG16 para Treino ---")
        df_vgg_train = extract_vgg16_features_simplified(df_train_set)
        output_path_train = os.path.join(OUTPUT_DIR, 'vgg16_features_train.csv')
        df_vgg_train.to_csv(output_path_train, index=False)
        print(f"Características VGG16 (Treino) salvas em: {output_path_train} ({len(df_vgg_train)} amostras)")

        # Extração VGG16 para o conjunto de Validação
        print("\n--- 3.2. Extração VGG16 para Validação ---")
        df_vgg_val = extract_vgg16_features_simplified(df_val_set)

```

```

output_path_val = os.path.join(OUTPUT_DIR, 'vgg16_features_val.csv')
df_vgg_val.to_csv(output_path_val, index=False)
print(f"Características VGG16 (Validação) salvas em: {output_path_val} ({len(df_vgg_val)} amostras)")

print("\n--- Processo de Extração VGG16 Concluído ---")
expected_features = df_vgg_train.shape[1] - 1
print(f"Número de características por imagem: {expected_features} (esperado 512)")
else:
    print("\nNão foi possível iniciar a extração VGG16: DataFrames de Treino/Validação estão vazios ou não definidos.")

```

=====

INÍCIO DA EXTRAÇÃO DE CARACTERÍSTICAS VGG16

=====

--- 3.1. Extração VGG16 para Treino ---

Carregando VGG16 (include_top=False, pooling='avg') para 512 features...

Iniciando extração VGG16 para 475 imagens...

Características VGG16 (Treino) salvas em: features_csv/vgg16_features_train.csv (475 amostras)

--- 3.2. Extração VGG16 para Validação ---

Carregando VGG16 (include_top=False, pooling='avg') para 512 features...

Iniciando extração VGG16 para 118 imagens...

Características VGG16 (Validação) salvas em: features_csv/vgg16_features_val.csv (118 amostras)

--- Processo de Extração VGG16 Concluído ---

Número de características por imagem: 512 (esperado 512)

4. Treine modelos Random Forest, SVM e RNA para predição dos dados extraídos (nessa tarefa utilize todas as imagens para o treinamento).

```

# --- CONFIGURAÇÕES GLOBAIS (Replicando o contexto) ---
OUTPUT_DIR = 'features_csv'
EXTRACTORS = ['lbp', 'vgg16']

# --- FUNÇÕES PARA TREINAMENTO DA TAREFA 4 ---

def load_combined_features(extractor_name: str) -> pd.DataFrame or None:
    """Carrega e concatena Treino e Validação (100% dos dados)."""
    train_path = os.path.join(OUTPUT_DIR, f'{extractor_name}_features_train.csv')
    val_path = os.path.join(OUTPUT_DIR, f'{extractor_name}_features_val.csv')

```

```

try:
    df_train = pd.read_csv(train_path)
    df_val = pd.read_csv(val_path)

    df_full = pd.concat([df_train, df_val], ignore_index=True)
    print(f"Dados {extractor_name} combinados: {df_full.shape} (Treino + Validação)")
    return df_full
except FileNotFoundError as e:
    print(f"Erro: Arquivos de características não encontrados para {extractor_name}: {e}")
    return None

def prepare_data_for_task_4(df_features: pd.DataFrame) -> Tuple[np.ndarray, np.ndarray, StandardScaler, LabelEncoder, list]:
    """Prepara, codifica e normaliza os dados, garantindo nomes de classes em string."""

    X = df_features.drop('class', axis=1).values
    y = df_features['class']

    # 1. Codificar labels
    le = LabelEncoder()
    y_encoded = le.fit_transform(y)

    # CORREÇÃO CRÍTICA: Garante que os nomes das classes são strings
    target_names_str = [str(c) for c in le.classes_]

    # 2. Normalizar dados
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    print(f"Shape dos dados finais: X={X.shape}, y={y_encoded.shape}")
    print(f"Classes decodificadas (STR): {target_names_str}")

    return X, X_scaled, y_encoded, scaler, le, target_names_str

def train_and_evaluate_models_task_4(X: np.ndarray, X_scaled: np.ndarray, y: np.ndarray, le: LabelEncoder, target_names: list, extractor_name: str):
    """Treina e avalia modelos usando Validação Cruzada (CV) de 5 folds."""

    models = {
        'Random Forest': RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42, n_jobs=-1),
        'SVM': SVC(kernel='rbf', C=1.0, gamma='scale', random_state=42, probability=True),
        'RNA (MLP)': MLPClassifier(hidden_layer_sizes=(100, 50), activation='relu', solver='adam', max_iter=1000, random_state=42, early_stopping=True)
    }

    trained_models = {}
    cv_results = []

```

```

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

print("\n" + "*60)
print(f"TREINAMENTO E AVALIAÇÃO CV - EXTRATOR: {extractor_name.upper()}")
print("*60)

for name, model in models.items():
    print(f"\n--- Processando {name} ---")

    # Selecionar dados (RF usa X não escalado; SVM/RNA usam X_scaled)
    X_data = X if name == 'Random Forest' else X_scaled

    # 1. Validação Cruzada (Cross-Validation)
    scores = cross_val_score(model, X_data, y, cv=cv, scoring='accuracy', n_jobs=-1)

    # 2. Treinamento Final (no conjunto completo, para salvar o modelo)
    model.fit(X_data, y)

    # 3. Previsão no próprio conjunto completo (para Matriz de Confusão)
    y_pred = model.predict(X_data)
    cm = confusion_matrix(y, y_pred)

    # Armazenar resultados
    cv_results[name] = {
        'model': model,
        'cv_mean_accuracy': scores.mean(),
        'cv_std_accuracy': scores.std(),
        'confusion_matrix': cm
    }
    trained_models[name] = model

    print(f"Acurácia Média CV (5-fold): {scores.mean():.4f} (+/- {scores.std():.4f})")
    print(f"Acurácia no Treino Completo (Overfitado): {accuracy_score(y, y_pred):.4f}")

    # Correção: target_names agora é garantidamente uma lista de strings
    print(f"Relatório de Classificação no Treino Completo:\n{classification_report(y, y_pred, target_names=target_names)}")

    # Plotar matriz de confusão no conjunto completo
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=target_names, yticklabels=target_names)
    plt.title(f'Matriz de Confusão - {name} ({extractor_name.upper()}) - Treino Completo')
    plt.ylabel('Valor Real')
    plt.xlabel('Predição')
    plt.tight_layout()

```

```
plt.show()

return trained_models, cv_results

def save_artifacts(artifacts: Dict[str, Any], extractor_name: str):
    """Salva os modelos treinados e os artefatos de pré-processamento."""

    models_dir = 'trained_models'
    os.makedirs(models_dir, exist_ok=True)

    filename = os.path.join(models_dir, f'artifacts_{extractor_name}.pkl')
    joblib.dump(artifacts, filename)
    print(f"\nArtefatos de {extractor_name.upper()} salvos em: '{filename}'")

# --- EXECUÇÃO PRINCIPAL DA TAREFA 4 ---

if __name__ == "__main__":
    all_artifacts = {}

    print("*80")
    print("TAREFA 4: TREINAMENTO DOS CLASSIFICADORES NO CONJUNTO COMPLETO (TREINO + VALIDAÇÃO)")
    print("*80")

    for extractor in EXTRACTORS:
        df_full = load_combined_features(extractor)
        if df_full is None or df_full.empty:
            continue

        # 1. Preparar Dados
        # Recebe 'target_names_str' (lista de strings) da função de preparação
        X, X_scaled, y, scaler, le, target_names = prepare_data_for_task_4(df_full)

        # 2. Treinar e Avaliar
        # Passa 'target_names' para a função de treinamento
        trained_models, cv_results = train_and_evaluate_models_task_4(X, X_scaled, y, le, target_names, extractor)

        # 3. Salvar Artefatos
        artifacts = {
            'models': trained_models,
            'scaler': scaler,
            'label_encoder': le,
            'cv_results': cv_results,
            'target_names': target_names # Salva os nomes das classes também
```

```
}

save_artifacts(artifacts, extractor)
all_artifacts[extractor] = artifacts

print("\n" + "*80)
print("TREINAMENTO DA TAREFA 4 CONCLUÍDO.")
print("Os modelos treinados estão prontos para a TAREFA 5 (avaliação na Base de Teste).")
print("*80)

# Resumo Final (CV)
print("\nRESUMO FINAL DE ACURÁCIA (Média CV 5-fold):")
for extractor, art in all_artifacts.items():
    print(f"\n--- Extrator {extractor.upper()} ---")
    for name, res in art['cv_results'].items():
        print(f"- {name}: {res['cv_mean_accuracy']:.4f}")
```


=====
TAREFA 4: TREINAMENTO DOS CLASSIFICADORES NO CONJUNTO COMPLETO (TREINO + VALIDAÇÃO)
=====

Dados lbp combinados: (593, 26) (Treino + Validação)
Shape dos dados finais: X=(593, 25), y=(593,)
Classes decodificadas (STR): ['0', '1', '2', '3']

=====

TREINAMENTO E AVALIAÇÃO CV - EXTRATOR: LBP
=====

--- Processando Random Forest ---

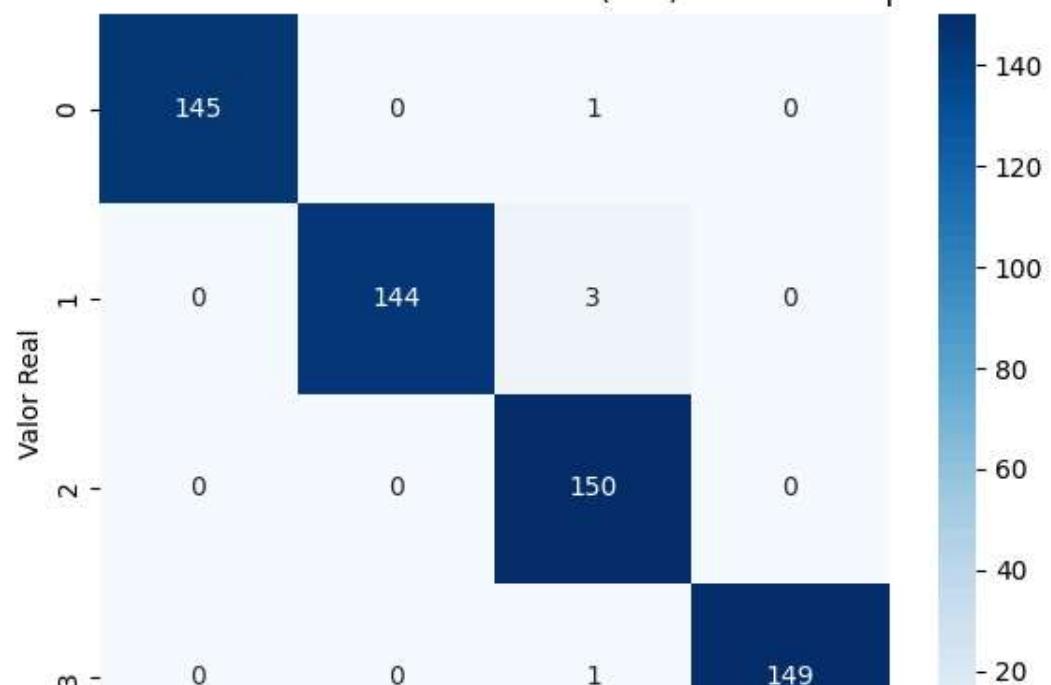
Acurácia Média CV (5-fold): 0.8364 (+/- 0.0264)

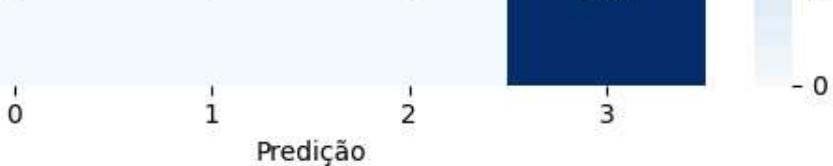
Acurácia no Treino Completo (Overfitado): 0.9916

Relatório de Classificação no Treino Completo:

	precision	recall	f1-score	support
0	1.00	0.99	1.00	146
1	1.00	0.98	0.99	147
2	0.97	1.00	0.98	150
3	1.00	0.99	1.00	150
accuracy			0.99	593
macro avg	0.99	0.99	0.99	593
weighted avg	0.99	0.99	0.99	593

Matriz de Confusão - Random Forest (LBP) - Treino Completo





--- Processando SVM ---

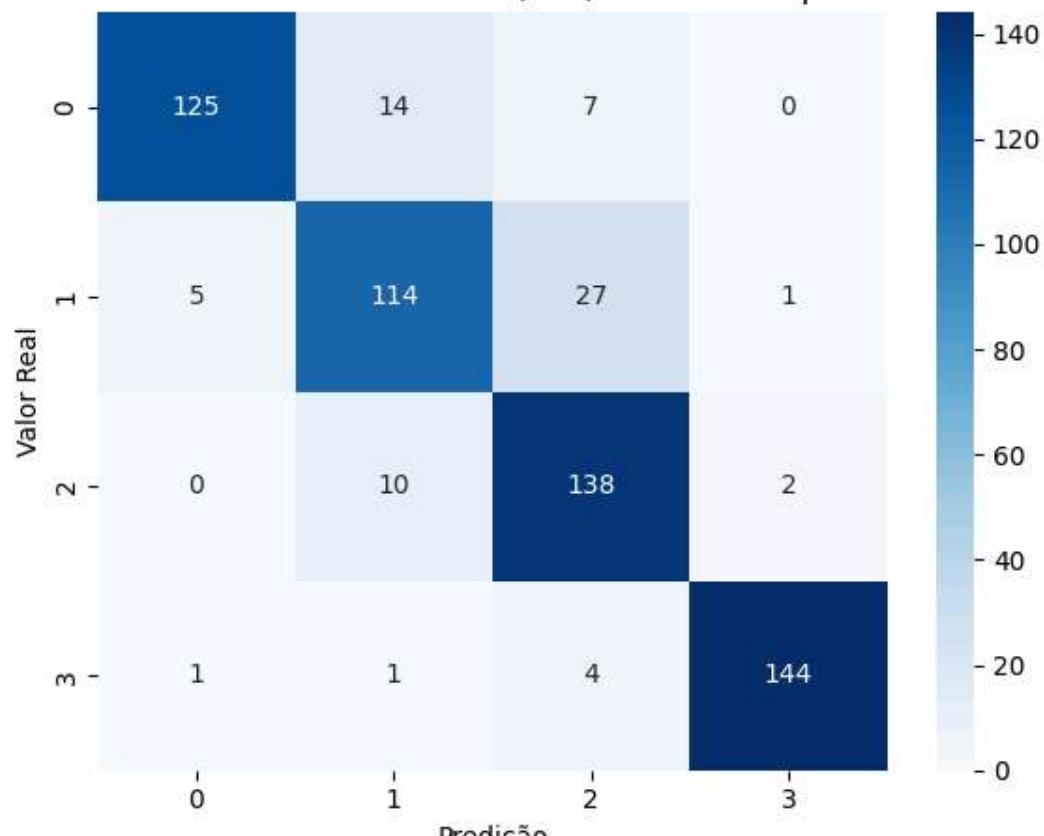
Acurácia Média CV (5-fold): 0.8482 (+/- 0.0192)

Acurácia no Treino Completo (Overfitado): 0.8786

Relatório de Classificação no Treino Completo:

	precision	recall	f1-score	support
0	0.95	0.86	0.90	146
1	0.82	0.78	0.80	147
2	0.78	0.92	0.85	150
3	0.98	0.96	0.97	150
accuracy			0.88	593
macro avg	0.88	0.88	0.88	593
weighted avg	0.88	0.88	0.88	593

Matriz de Confusão - SVM (LBP) - Treino Completo



--- Processando RNA (MLP) ---

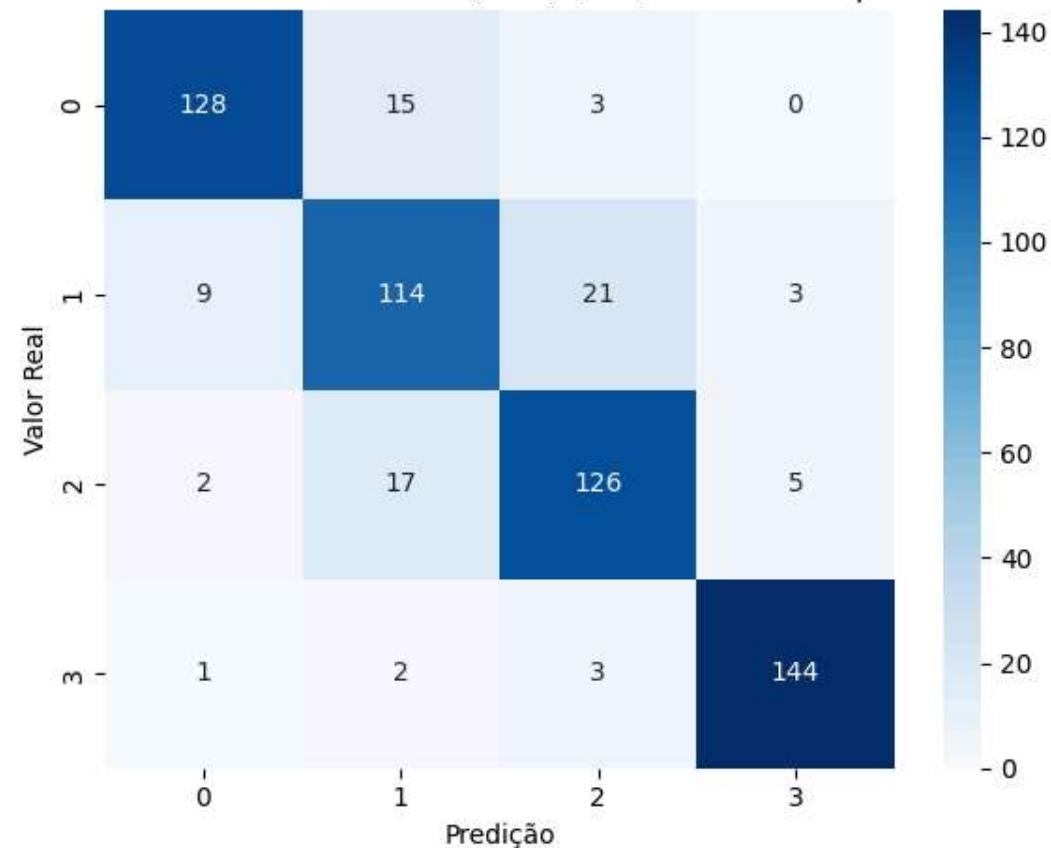
Acurácia Média CV (5-fold): 0.8196 (+/- 0.0171)

Acurácia no Treino Completo (Overfitado): 0.8634

Relatório de Classificação no Treino Completo:

	precision	recall	f1-score	support
0	0.91	0.88	0.90	146
1	0.77	0.78	0.77	147
2	0.82	0.84	0.83	150
3	0.95	0.96	0.95	150
accuracy			0.86	593
macro avg	0.86	0.86	0.86	593
weighted avg	0.86	0.86	0.86	593

Matriz de Confusão - RNA (MLP) (LBP) - Treino Completo



Artefatos de LBP salvos em: 'trained_models/artifacts_lbp.pkl'

Dados vgg16 combinados: (593, 513) (Treino + Validação)

Shape dos dados finais: X=(593, 512), y=(593,)

Classes decodificadas (STR): ['0', '1', '2', '3']

=====
TREINAMENTO E AVALIAÇÃO CV - EXTRATOR: VGG16
=====

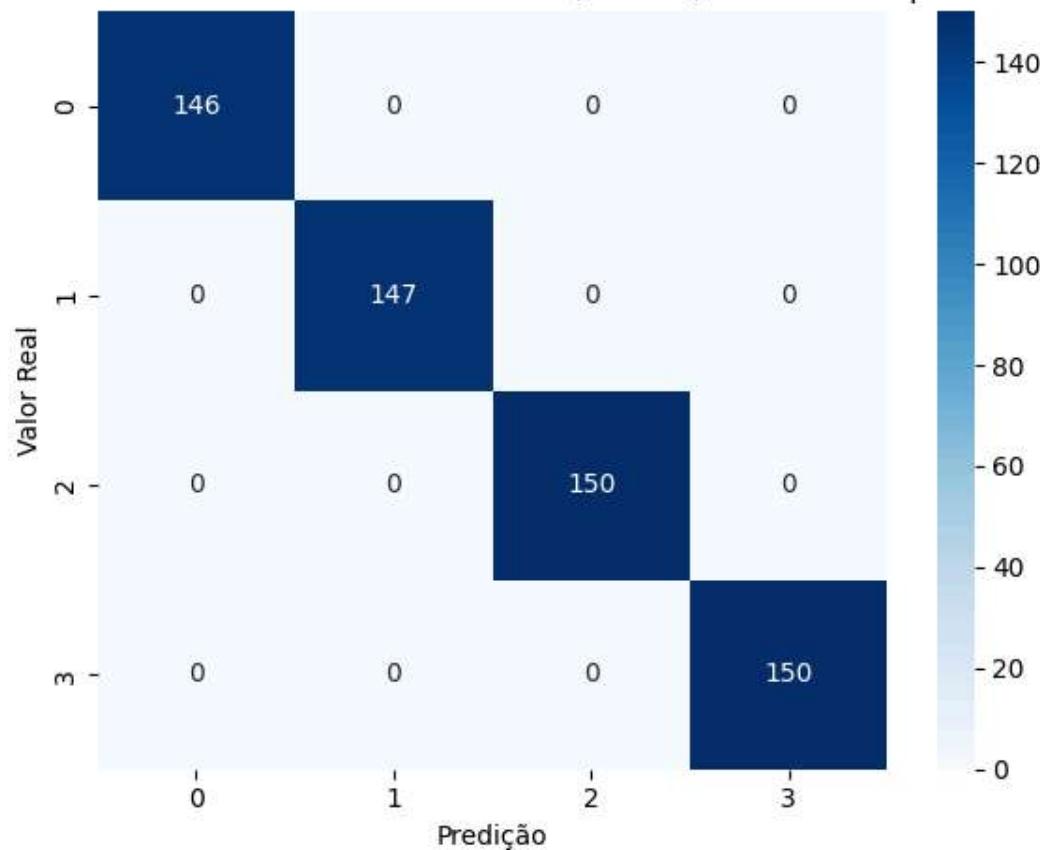
--- Processando Random Forest ---

Acurácia Média CV (5-fold): 0.9309 (+/- 0.0210)
Acurácia no Treino Completo (Overfitado): 1.0000

Relatório de Classificação no Treino Completo:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	146
1	1.00	1.00	1.00	147
2	1.00	1.00	1.00	150
3	1.00	1.00	1.00	150
accuracy			1.00	593
macro avg	1.00	1.00	1.00	593
weighted avg	1.00	1.00	1.00	593

Matriz de Confusão - Random Forest (VGG16) - Treino Completo



--- Processando SVM ---

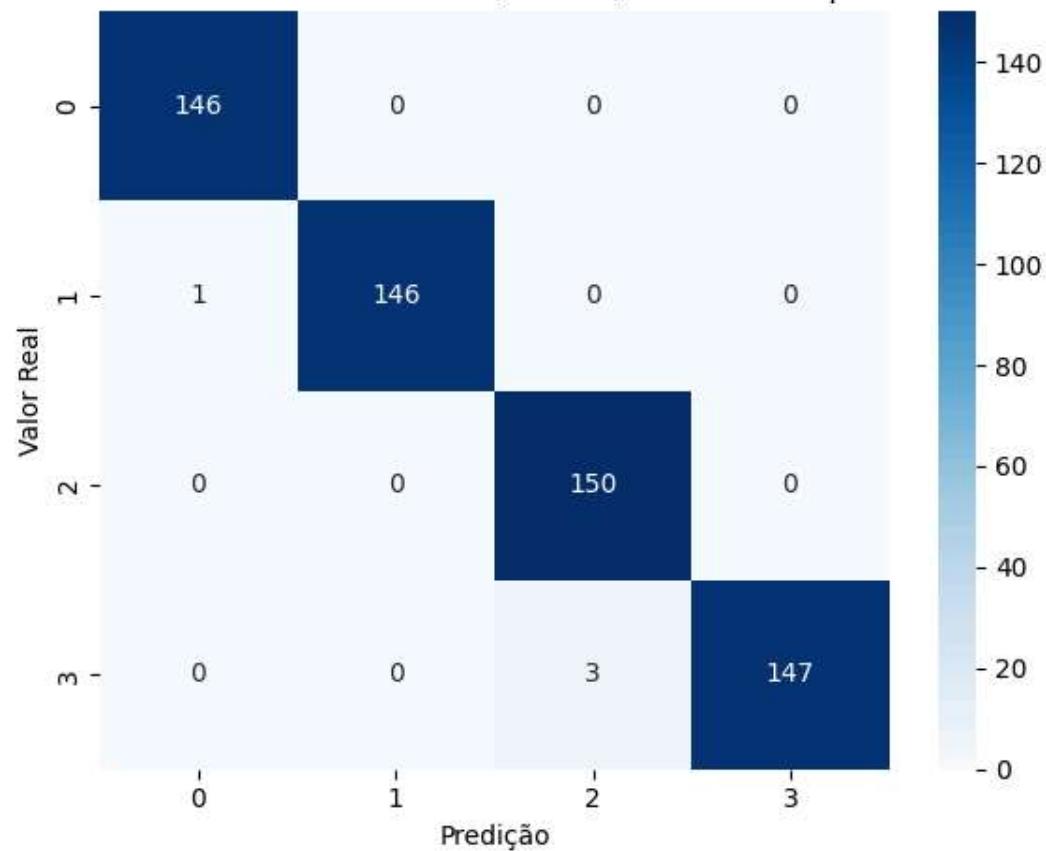
Acurácia Média CV (5-fold): 0.9343 (+/- 0.0234)

Acurácia no Treino Completo (Overfitado): 0.9933

Relatório de Classificação no Treino Completo:

	precision	recall	f1-score	support
0	0.99	1.00	1.00	146
1	1.00	0.99	1.00	147
2	0.98	1.00	0.99	150
3	1.00	0.98	0.99	150
accuracy			0.99	593
macro avg	0.99	0.99	0.99	593
weighted avg	0.99	0.99	0.99	593

Matriz de Confusão - SVM (VGG16) - Treino Completo



--- Processando RNA (MLP) ---

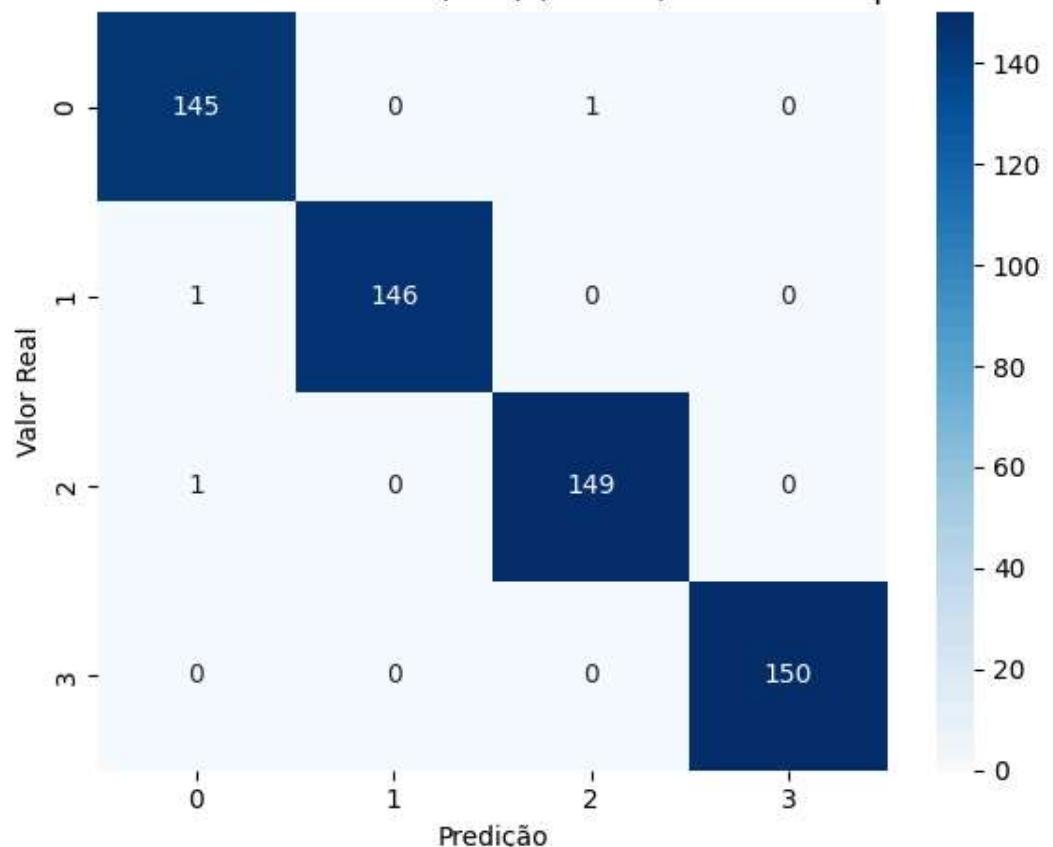
Acurácia Média CV (5-fold): 0.9292 (+/- 0.0179)

Acurácia no Treino Completo (Overfitado): 0.9949

Relatório de Classificação no Treino Completo:

	precision	recall	f1-score	support
0	0.99	0.99	0.99	146
1	1.00	0.99	1.00	147
2	0.99	0.99	0.99	150
3	1.00	1.00	1.00	150
accuracy			0.99	593
macro avg	0.99	0.99	0.99	593
weighted avg	0.99	0.99	0.99	593

Matriz de Confusão - RNA (MLP) (VGG16) - Treino Completo



Artefatos de VGG16 salvos em: 'trained_models/artifacts_vgg16.pkl'

=====

TREINAMENTO DA TAREFA 4 CONCLUÍDO.

Os modelos treinados estão prontos para a TAREFA 5 (avaliação na Base de Teste).

=====

RESUMO FINAL DE ACURÁCIA (Média CV 5-fold):

✓ 5. Carregue a base de Teste e execute a tarefa 3 nesta base.

-- Extrator LBP --
- Random Forest: 0.8584
- SVM: 0.8482
- RNA (MDA): 0.8196

```
# --- 1. CONFIGURAÇÕES GLOBAIS NECESSÁRIAS (Da Tarefa 1/3) ---
```

```
ZIP_TEST_PATH = '/content/Test_Warwick.zip'  
TEST_DIR = '/content/Test_Warwick/Test_Warwick/Test_4cl amostra/'  
CLASSES = ['0', '1', '2', '3']  
OUTPUT_DIR = 'features_csv'  
os.makedirs(OUTPUT_DIR, exist_ok=True)  
patient_regex = re.compile(r"(\d+)_HER2_.*\.png")  
  
# Parâmetros LBP  
RADIUS = 3  
N_POINTS = 8 * RADIUS  
METHOD = 'uniform'  
IMG_SIZE = (250, 250)  
  
# Parâmetros VGG16  
TARGET_SIZE = (224, 224)  
POOLING = 'avg'
```

```
# --- 2. FUNÇÕES AUXILIARES NECESSÁRIAS (Reutilizadas da Tarefa 1/3) ---
```

```
def extract_patient_id(filename):  
    match = patient_regex.match(filename)  
    if match:  
        return match.group(1)  
    return None  
  
def run_extraction_pipeline(zip_path, base_dir, classes, dataset_name):  
    """Executa a descompactação e mapeamento do dataset (Teste)."""  
  
    # 1. Descompactação  
    if not os.path.exists(zip_path):  
        print(f"ERRO: Arquivo ZIP de {dataset_name} não encontrado em: {zip_path}")  
        return None  
    os.makedirs(base_dir, exist_ok=True)  
    try:  
        # Tenta descompactar no diretório pai do 'base_dir'  
        with zipfile.ZipFile(zip_path, 'r') as zip_ref:  
            zip_ref.extractall(os.path.dirname(os.path.dirname(base_dir)))  
        print(f"Descompactação de {dataset_name} concluída.")
```

```

except Exception as e:
    print(f"ERRO durante a descompactação de {dataset_name}: {e}")
    return None

# 2. Mapeamento
files = []
for cls in classes:
    class_path = os.path.join(base_dir, cls)
    if os.path.exists(class_path):
        for filename in os.listdir(class_path):
            if filename.endswith('.png'):
                patient_id = extract_patient_id(filename)
                if patient_id:
                    files.append({'filepath': os.path.join(class_path, filename), 'class': cls, 'patient_id': patient_id})
df = pd.DataFrame(files)

if df.empty:
    print(f"ERRO: Nenhum arquivo de imagem encontrado para {dataset_name}.")
    return None
print(f"Total de imagens de {dataset_name} carregadas: {len(df)}")
return df

def extract_lbp_features(df, dataset_type):
    """Extrai características LBP e salva em CSV."""
    lbp_features = []
    lbp_labels = []
    for index, row in df.iterrows():
        try:
            img_pil = Image.open(row['filepath']).convert('RGB').resize(IMG_SIZE)
            img_resized = np.array(img_pil)
            img_normalized = img_resized.astype(float) / 255.0
            img_gray = color.rgb2gray(img_normalized)
            lbp = feature.local_binary_pattern(img_gray, N_POINTS, RADIUS, method=METHOD)
            hist, _ = np.histogram(lbp.ravel(), bins=np.arange(0, N_POINTS + 2), range=(0, N_POINTS + 1))
            hist = hist.astype("float")
            hist /= (hist.sum() + 1e-7)
            lbp_features.append(hist)
            lbp_labels.append(row['class'])
        except:
            pass
    X_lbp = np.array(lbp_features)
    y_lbp = np.array(lbp_labels, dtype=str)
    df_features = pd.DataFrame(X_lbp)
    df_features['class'] = y_lbp

```

```

output_path = os.path.join(OUTPUT_DIR, f'lbp_features_{dataset_type}.csv')
df_features.to_csv(output_path, index=False)
print(f"Características LBP salvas ({dataset_type}): {output_path} ({len(df_features)} amostras)")
return df_features

def extract_vgg16_features_simplified(df, dataset_type):
    """Extrai características VGG16 (512 features) e salva em CSV."""
    model = VGG16(weights='imagenet', include_top=False, pooling=POOLING)
    vgg_features = []
    vgg_labels = []
    for index, row in df.iterrows():
        try:
            img_pil = Image.open(row['filepath']).convert('RGB').resize(TARGET_SIZE)
            img_array = np.array(img_pil)
            img_array = np.expand_dims(img_array, axis=0)
            img_processed = preprocess_input(img_array)
            features = model.predict(img_processed, verbose=0).flatten()
            vgg_features.append(features)
            vgg_labels.append(row['class'])
        except:
            pass
    X_vgg = np.array(vgg_features)
    y_vgg = np.array(vgg_labels, dtype=str)
    df_features = pd.DataFrame(X_vgg)
    df_features['class'] = y_vgg

    output_path = os.path.join(OUTPUT_DIR, f'vgg16_features_{dataset_type}.csv')
    df_features.to_csv(output_path, index=False)
    print(f"Características VGG16 salvas ({dataset_type}): {output_path} ({len(df_features)} amostras)")
    return df_features

```

--- 3. EXECUÇÃO DA TAREFA 5 ---

```

def run_task_5():
    """Carrega a base de Teste e executa a extração de características (Tarefa 3)."""

    print("*"*60)
    print("TAREFA 5: PROCESSANDO A BASE DE TESTE")
    print("*"*60)

    # 1. Carregar Base de Teste e Mapear
    df_full_test = run_extraction_pipeline(ZIP_TEST_PATH, TEST_DIR, CLASSES, "Teste")

```

```

if df_full_test is None:
    print("Falha ao carregar a Base de Teste.")
    return

# 2. Executar Tarefa 3 (Extração) na Base de Teste
print("\nExecutando Extração LBP na Base de Teste...")
extract_lbp_features(df_full_test, "test")

print("\nExecutando Extração VGG16 na Base de Teste...")
extract_vgg16_features_simplified(df_full_test, "test")

print("\nProcessamento da Base de Teste concluído.")

if __name__ == "__main__":
    run_task_5()

=====
TAREFA 5: PROCESSANDO A BASE DE TESTE
=====
Descompactação de Teste concluída.
Total de imagens de Teste carregadas: 371

Executando Extração LBP na Base de Teste...
/usr/local/lib/python3.12/dist-packages/skimage/feature/texture.py:385: UserWarning: Applying `local_binary_pattern` to floating-point images may result in unexpected behavior.
  warnings.warn(
Características LBP salvas (test): features_csv/lbp_features_test.csv (371 amostras)

Executando Extração VGG16 na Base de Teste...
Características VGG16 salvas (test): features_csv/vgg16_features_test.csv (371 amostras)

Processamento da Base de Teste concluído.

```

✓ 6. Aplique os modelos treinados nos dados de teste.

```

# --- CONFIGURAÇÕES GLOBAIS REUTILIZADAS ---
OUTPUT_DIR = 'features_csv'
MODELS_DIR = 'trained_models'
EXTRACTORS = ['lbp', 'vgg16']

# --- FUNÇÕES DA TAREFA 6 ---

def load_test_data(extractor_name: str) -> pd.DataFrame or None:
    """Carrega os dados de características do teste gerados na Tarefa 5."""

```

```

    carrega os dados de características de teste gerados na Tarefa 3.
test_path = os.path.join(OUTPUT_DIR, f'{extractor_name}_features_test.csv')
try:
    df_test = pd.read_csv(test_path)
    print(f"Dados de Teste {extractor_name} carregados: {df_test.shape}")
    return df_test
except FileNotFoundError:
    print(f"ERRO: Arquivo de Teste não encontrado em: {test_path}. Execute a Tarefa 5 primeiro.")
    return None

def load_trained_artifacts(extractor_name: str) -> Dict[str, Any] or None:
    """Carrega os modelos treinados e artefatos de pré-processamento da Tarefa 4."""
    artifacts_path = os.path.join(MODELS_DIR, f'artifacts_{extractor_name}.pkl')
    try:
        artifacts = joblib.load(artifacts_path)
        print(f"Artefatos treinados de {extractor_name.upper()} carregados.")
        return artifacts
    except FileNotFoundError:
        print(f"ERRO: Arquivos de modelos não encontrados em: {artifacts_path}. Execute a Tarefa 4 primeiro.")
        return None

def apply_preprocessing(X_test: np.ndarray, scaler, model_name: str) -> np.ndarray:
    """Aplica o pré-processamento (normalização) nos dados de teste, se necessário."""

    # Random Forest (RF) foi treinado com dados NÃO ESCALADOS na Tarefa 4
    if 'Random Forest' in model_name:
        return X_test
    # SVM e RNA foram treinados com dados ESCALADOS na Tarefa 4
    else:
        # Usa o scaler TREINADO na Tarefa 4 (função transform)
        return scaler.transform(X_test)

# FUNÇÃO MODIFICADA (Substitui evaluate_test_performance)
def apply_models_to_test(df_test: pd.DataFrame, artifacts: Dict[str, Any], extractor_name: str):
    """
    Aplica os modelos de Treino nos dados de Teste e salva as previsões.
    A avaliação de desempenho foi removida.
    """

    # 1. Preparar dados de Teste
    X_test_raw = df_test.drop('class', axis=1).values
    y_test_labels = df_test['class'] # Rótulos reais originais

    scaler = artifacts['scaler']

```

```
le = artifacts['label_encoder']
trained_models = artifacts['models']

# DataFrame para armazenar as previsões de todos os modelos
df_predictions = pd.DataFrame({'Real_Class': y_test_labels.values})

print("\n" + "="*70)
print(f"APLICANDO MODELOS TREINADOS - EXTRATOR: {extractor_name.upper()}")
print("=*70")

for model_name, model in trained_models.items():
    print(f"--- Aplicando {model_name} ---")

    # 2. Aplicar pré-processamento correto
    X_test_processed = apply_preprocessing(X_test_raw, scaler, model_name)

    # 3. Fazer Previsões (Encoded)
    y_pred_encoded = model.predict(X_test_processed)

    # 4. Converter Previsões para Rótulos (Labels)
    y_pred_labels = le.inverse_transform(y_pred_encoded)

    # Armazenar as previsões
    df_predictions[f'Predicted_{model_name.replace(" ", "_")}] = y_pred_labels

# 5. Salvar as previsões em um CSV
output_path = os.path.join(OUTPUT_DIR, f'predictions_{extractor_name}_test.csv')
df_predictions.to_csv(output_path, index=False)
print(f"\nPrevisões salvas em: {output_path}")

# --- EXECUÇÃO PRINCIPAL DA TAREFA 6 (MODIFICADA) ---

def run_task_6():
    """Executa a aplicação dos modelos treinados na base de teste (sem avaliação)."""

    print("\n" + "#"*80)
    print("TAREFA 6: APLICAÇÃO DOS MODELOS TREINADOS NA BASE DE TESTE (SEM AVALIAÇÃO)")
    print("#"*80)

    # Cria o diretório de modelos, caso não exista (idealmente criado na T4)
    os.makedirs(MODELS_DIR, exist_ok=True)
    os.makedirs(OUTPUT_DIR, exist_ok=True)
```

```

for extractor in EXTRACTORS:
    # 1. Carregar Dados de Teste (Tarefas 5)
    df_test = load_test_data(extractor)
    if df_test is None or df_test.empty:
        continue

    # 2. Carregar Artefatos Treinados (Tarefa 4)
    artifacts = load_trained_artifacts(extractor)
    if artifacts is None:
        continue

    # 3. Aplicar Modelos (Substitui a avaliação)
    apply_models_to_test(df_test, artifacts, extractor)

print("\n" + "#"*80)
print("TAREFA 6 CONCLUÍDA: PREDIÇÕES GERADAS PARA AVALIAÇÃO POSTERIOR (TAREFA 7).")
print("#"*80)

if __name__ == "__main__":
    # Nota: Para rodar este bloco, as Tarefas 1-5 devem ter sido executadas
    # com sucesso, criando os arquivos CSV de teste e os arquivos .pkl dos modelos.
    run_task_6()

```

```

#####
TAREFA 6: APLICAÇÃO DOS MODELOS TREINADOS NA BASE DE TESTE (SEM AVALIAÇÃO)
#####
Dados de Teste lbp carregados: (371, 26)
Artefatos treinados de LBP carregados.

```

```

=====
APLICANDO MODELOS TREINADOS - EXTRATOR: LBP
=====
--- Aplicando Random Forest ---
--- Aplicando SVM ---
--- Aplicando RNA (MLP) ---


```

```

Predições salvas em: features_csv/predictions_lbp_test.csv
Dados de Teste vgg16 carregados: (371, 513)
Artefatos treinados de VGG16 carregados.

=====
```

```

APLICANDO MODELOS TREINADOS - EXTRATOR: VGG16
=====
--- Aplicando Random Forest ---
--- Aplicando SVM ---


```

--- Aplicando RNA (MLP) ---

Predições salvas em: features_csv/predictions_vgg16_test.csv

```
#####
TAREFA 6 CONCLUÍDA: PREDIÇÕES GERADAS PARA AVALIAÇÃO POSTERIOR (TAREFA 7).
#####
```

7. Calcule as métricas de Sensibilidade, Especificidade e F1-Score com base em suas matrizes de confusão.

```
# --- CONFIGURAÇÕES GLOBAIS REUTILIZADAS ---
OUTPUT_DIR = 'features_csv'
MODELS_DIR = 'trained_models'
EXTRACTORS = ['lbp', 'vgg16']

# --- FUNÇÃO DA TAREFA 7 (MÉTRICAS) ---

def calculate_multiclass_metrics(cm: np.ndarray, target_names: list) -> pd.DataFrame:
    """
    Calcula Sensibilidade (Recall), Especificidade e F1-Score para cada classe
    a partir da matriz de confusão multi-classe.
    """
    n_classes = cm.shape[0]
    metrics = []

    for i in range(n_classes):
        # Para a classe 'i':

        # True Positives (TP) - Diagonal
        TP = cm[i, i]

        # False Positives (FP) - Soma da coluna i, excluindo TP
        FP = np.sum(cm[:, i]) - TP

        # False Negatives (FN) - Soma da linha i, excluindo TP
        FN = np.sum(cm[i, :]) - TP

        # True Negatives (TN) - Soma de todos os elementos, excluindo linha i e coluna i
        TN = np.sum(cm) - (TP + FP + FN)

        # Sensibilidade (Recall) = TP / (TP + FN)
        recall = TP / (TP + FN)

        # Specificidade = TN / (TN + FP)
        specificity = TN / (TN + FP)

        # F1-Score = 2 * (Precision * Recall) / (Precision + Recall)
        f1_score = 2 * (recall * specificity) / (recall + specificity)

        metrics.append({
            'target_name': target_names[i],
            'TP': TP,
            'FP': FP,
            'FN': FN,
            'TN': TN,
            'Recall': recall,
            'Specificity': specificity,
            'F1_Score': f1_score
        })

    return pd.DataFrame(metrics)
```

```

sensitivity = TP / (TP + FN) if (TP + FN) > 0 else 0

# Especificidade = TN / (TN + FP)
specificity = TN / (TN + FP) if (TN + FP) > 0 else 0

# Precisão (Precision) = TP / (TP + FP) - Necessário para F1
precision = TP / (TP + FP) if (TP + FP) > 0 else 0

# F1-Score = 2 * (Precision * Recall) / (Precision + Recall)
f1_score = 2 * (precision * sensitivity) / (precision + sensitivity) if (precision + sensitivity) > 0 else 0

metrics.append({
    'Classe': target_names[i],
    'Sensibilidade (Recall)': sensitivity,
    'Especificidade': specificity,
    'F1-Score': f1_score
})

df_metrics = pd.DataFrame(metrics).set_index('Classe')
return df_metrics

```

--- MODIFICAÇÃO DA FUNÇÃO evaluate_test_performance (TAREFA 6/7) ---

```

def evaluate_test_performance(df_test: pd.DataFrame, artifacts: Dict[str, Any], extractor_name: str):
    """Aplica os modelos de Treino nos dados de Teste e exibe a performance (incluindo T7)."""

```

```

# 1. Preparar dados de Teste (Mesma lógica da T6)
X_test_raw = df_test.drop('class', axis=1).values
y_test_encoded = artifacts['label_encoder'].transform(df_test['class'])

```

```

scaler = artifacts['scaler']
le = artifacts['label_encoder']
trained_models = artifacts['models']
target_names = artifacts['target_names']

```

```

print("\n" + "="*70)
print(f"AVALIAÇÃO NA BASE DE TESTE - EXTRATOR: {extractor_name.upper()}")
print("=*70")

```

```

for model_name, model in trained_models.items():
    print(f"\n--- Aplicando {model_name} ---")

```

2. Aplicar pré-processamento correto

```

X_test_processed = apply_preprocessing(X_test_raw, scaler, model_name)

# 3. Fazer Previsões
y_pred_encoded = model.predict(X_test_processed)
y_pred_labels = le.inverse_transform(y_pred_encoded)
y_test_labels = le.inverse_transform(y_test_encoded)

# 4. Avaliar Performance
accuracy = accuracy_score(y_test_labels, y_pred_labels)
cm = confusion_matrix(y_test_labels, y_pred_labels)

print(f"Acurácia de Teste: {accuracy:.4f}")

# TAREFA 7: CÁLCULO E EXIBIÇÃO DE MÉTRICAS DETALHADAS
df_metrics = calculate_multiclass_metrics(cm, target_names)
print("\n--- TAREFA 7: MÉTRICAS POR CLASSE ---")
print(df_metrics.round(4)) # Exibe as métricas formatadas
print("-----")

print(f"Relatório de Classificação:\n{classification_report(y_test_labels, y_pred_labels, target_names=target_names)}")

# 5. Plotar Matriz de Confusão
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Reds', xticklabels=target_names, yticklabels=target_names)
plt.title(f'Matriz de Confusão - {model_name} ({extractor_name.upper()}) - Base de Teste')
plt.ylabel('Valor Real')
plt.xlabel('Previsão')
plt.tight_layout()
plt.show()

# --- EXECUÇÃO PRINCIPAL DA TAREFA 6 (COM AVALIAÇÃO DA TAREFA 7) ---

def run_task_7():
    """Executa a aplicação dos modelos treinados na base de teste e calcula as métricas."""

    print("\n" + "#"*80)
    print("TAREFA 7: CÁLCULO DE MÉTRICAS DETALHADAS")
    print("#"*80)

    # Cria diretórios
    os.makedirs(MODELS_DIR, exist_ok=True)
    os.makedirs(OUTPUT_DIR, exist_ok=True)

```

```
for extractor in EXTRACTORS:
    # 1. Carregar Dados de Teste
    df_test = load_test_data(extractor)
    if df_test is None or df_test.empty:
        continue

    # 2. Carregar Artefatos Treinados
    artifacts = load_trained_artifacts(extractor)
    if artifacts is None:
        continue

    # 3. Avaliar Desempenho (incluindo cálculo da T7)
    evaluate_test_performance(df_test, artifacts, extractor)

print("\n" + "#"*80)
print("TAREFA 7 CONCLUÍDA: MÉTRICAS DE SENSIBILIDADE, ESPECIFICIDADE E F1-SCORE CALCULADAS.")
print("#"*80)

if __name__ == "__main__":
    # executa a Tarefa 7:
    run_task_7()
```



```
#####
#####
```

TAREFA 7: CÁLCULO DE MÉTRICAS DETALHADAS

```
#####
#####
```

Dados de Teste lbp carregados: (371, 26)

Artefatos treinados de LBP carregados.

```
=====
=====
```

AVALIAÇÃO NA BASE DE TESTE - EXTRATOR: LBP

```
=====
=====
```

--- Aplicando Random Forest ---

Acurácia de Teste: 0.7197

--- TAREFA 7: MÉTRICAS POR CLASSE ---

	Sensibilidade (Recall)	Especificidade	F1-Score
--	------------------------	----------------	----------

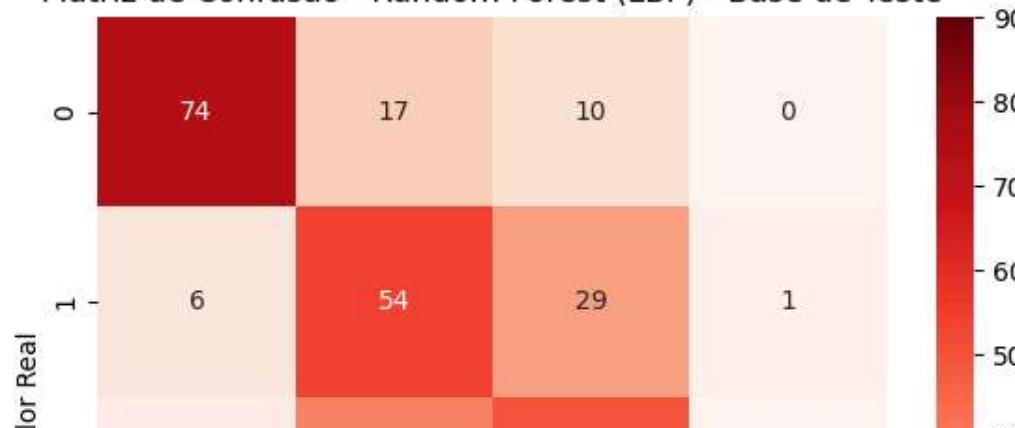
Classe

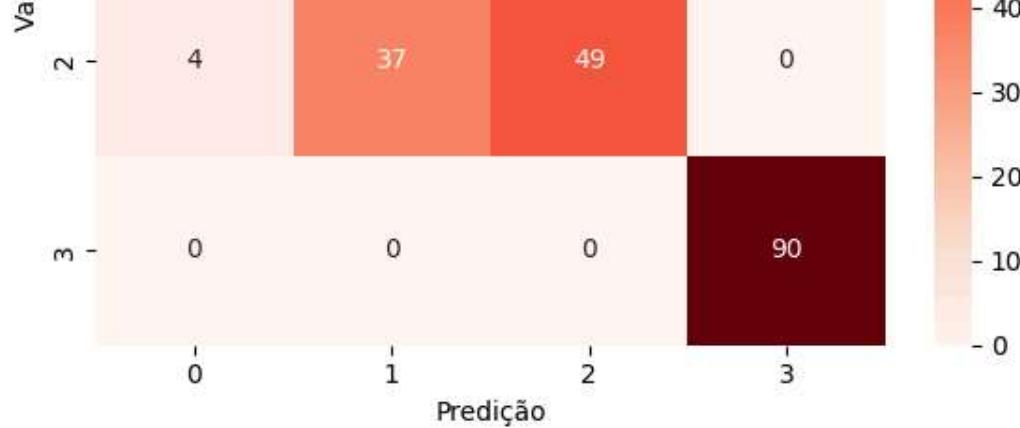
0	0.7327	0.9630	0.8000
1	0.6000	0.8078	0.5455
2	0.5444	0.8612	0.5506
3	1.0000	0.9964	0.9945

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.88	0.73	0.80	101
1	0.50	0.60	0.55	90
2	0.56	0.54	0.55	90
3	0.99	1.00	0.99	90
accuracy			0.72	371
macro avg	0.73	0.72	0.72	371
weighted avg	0.74	0.72	0.72	371

Matriz de Confusão - Random Forest (LBP) - Base de Teste





--- Aplicando SVM ---

Acurácia de Teste: 0.7385

--- TAREFA 7: MÉTRICAS POR CLASSE ---

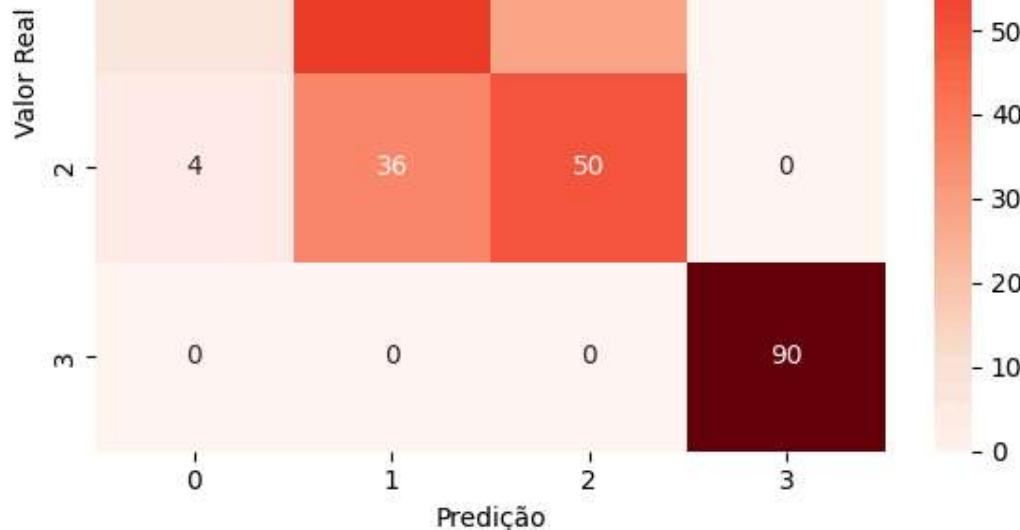
	Sensibilidade (Recall)	Especificidade	F1-Score
Classe			
0	0.7723	0.9630	0.8254
1	0.6222	0.8327	0.5803
2	0.5556	0.8577	0.5556
3	1.0000	1.0000	1.0000

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.89	0.77	0.83	101
1	0.54	0.62	0.58	90
2	0.56	0.56	0.56	90
3	1.00	1.00	1.00	90
accuracy			0.74	371
macro avg	0.75	0.74	0.74	371
weighted avg	0.75	0.74	0.74	371

Matriz de Confusão - SVM (LBP) - Base de Teste





--- Aplicando RNA (MLP) ---

Acurácia de Teste: 0.7412

--- TAREFA 7: MÉTRICAS POR CLASSE ---

	Sensibilidade (Recall)	Especificidade	F1-Score
--	------------------------	----------------	----------

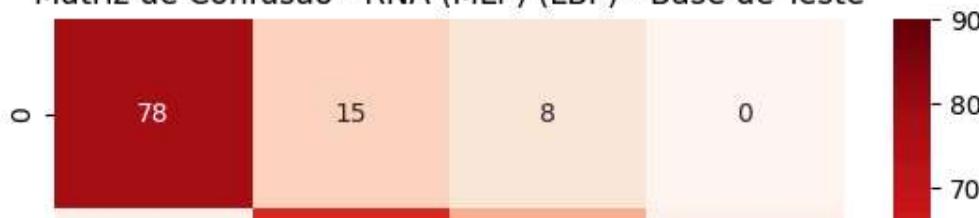
Classe

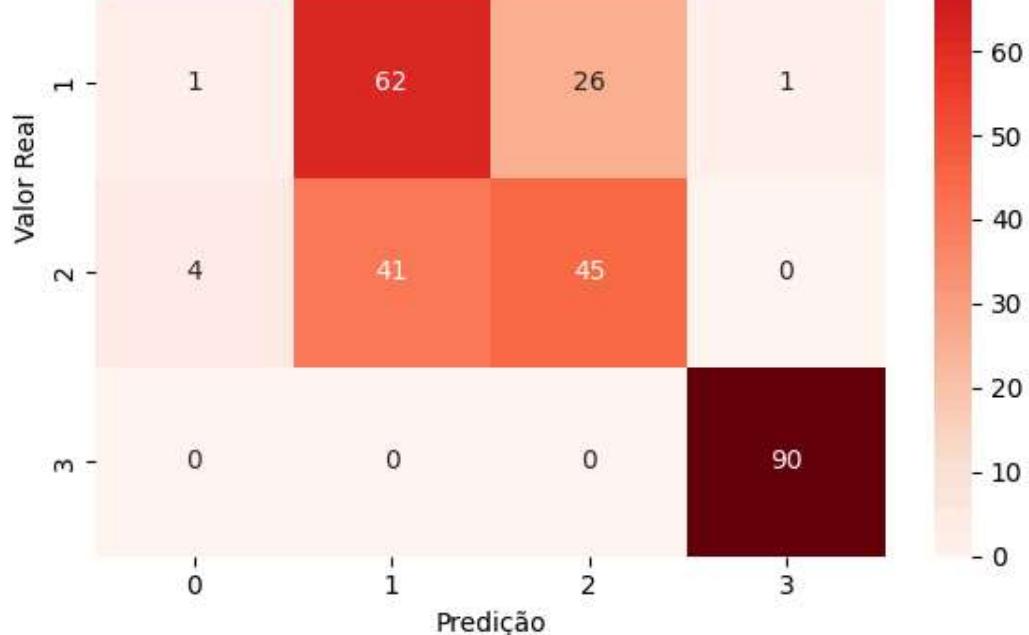
Classe	Sensibilidade (Recall)	Especificidade	F1-Score
0	0.7723	0.9815	0.8478
1	0.6889	0.8007	0.5962
2	0.5000	0.8790	0.5325
3	1.0000	0.9964	0.9945

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.94	0.77	0.85	101
1	0.53	0.69	0.60	90
2	0.57	0.50	0.53	90
3	0.99	1.00	0.99	90
accuracy			0.74	371
macro avg	0.76	0.74	0.74	371
weighted avg	0.76	0.74	0.75	371

Matriz de Confusão - RNA (MLP) (LBP) - Base de Teste





Dados de Teste vgg16 carregados: (371, 513)
Artefatos treinados de VGG16 carregados.

=====
AVALIAÇÃO NA BASE DE TESTE - EXTRATOR: VGG16
=====

--- Aplicando Random Forest ---

Acurácia de Teste: 0.8005

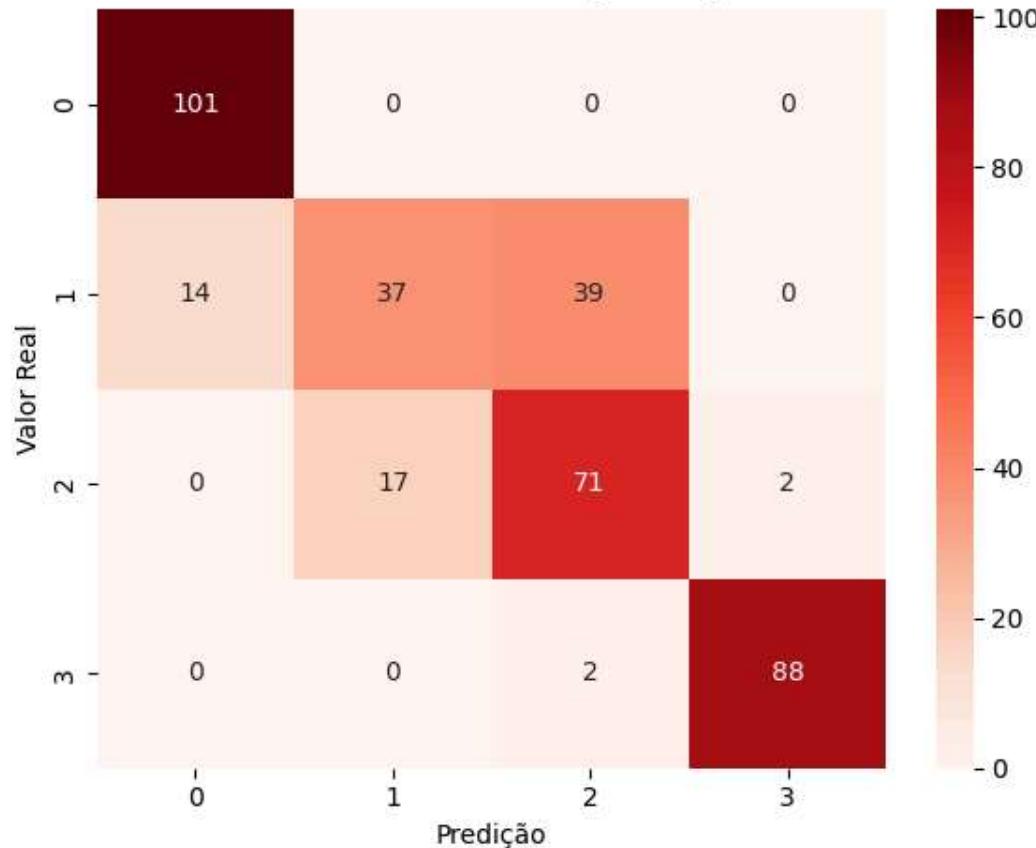
--- TAREFA 7: MÉTRICAS POR CLASSE ---

Classe	Sensibilidade (Recall)	Especificidade	F1-Score
0	1.0000	0.9481	0.9352
1	0.4111	0.9395	0.5139
2	0.7889	0.8541	0.7030
3	0.9778	0.9929	0.9778

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.88	1.00	0.94	101
1	0.69	0.41	0.51	90
2	0.63	0.79	0.70	90
3	0.98	0.98	0.98	90
accuracy			0.80	371
macro avg	0.79	0.79	0.78	371
weighted avg	0.80	0.80	0.79	371

Matriz de Confusão - Random Forest (VGG16) - Base de Teste



--- Aplicando SVM ---

Acurácia de Teste: 0.8032

--- TAREFA 7: MÉTRICAS POR CLASSE ---

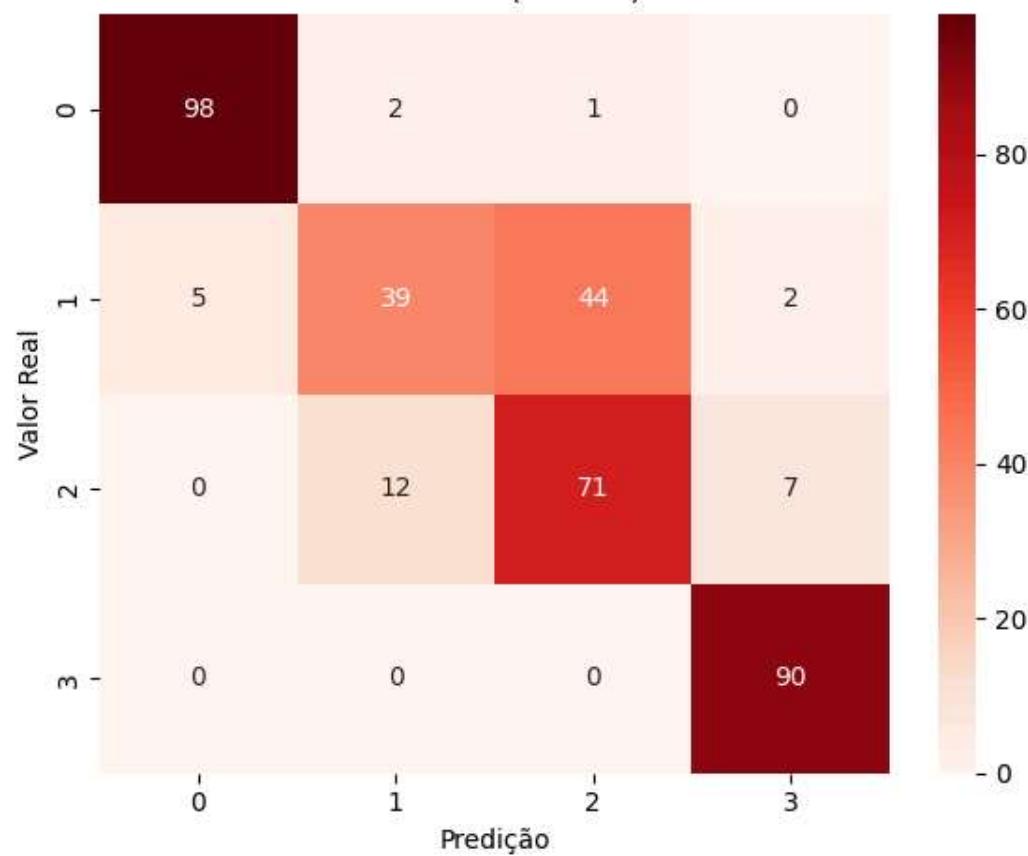
Classe	Sensibilidade (Recall)	Especificidade	F1-Score
0	0.9703	0.9815	0.9608
1	0.4333	0.9502	0.5455
2	0.7889	0.8399	0.6893
3	1.0000	0.9680	0.9524

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.95	0.97	0.96	101
1	0.74	0.43	0.55	90
2	0.61	0.79	0.69	90
3	0.91	1.00	0.95	90

accuracy		0.80	0.80	371
macro avg	0.80	0.80	0.79	371
weighted avg	0.81	0.80	0.79	371

Matriz de Confusão - SVM (VGG16) - Base de Teste



--- Aplicando RNA (MLP) ---

Acurácia de Teste: 0.8059

--- TAREFA 7: MÉTRICAS POR CLASSE ---

Classe	Sensibilidade (Recall)	Especificidade	F1-Score
0	0.9505	0.9778	0.9458
1	0.4667	0.9359	0.5600
2	0.8000	0.8470	0.7024
3	0.9889	0.9822	0.9674

Relatório de Classificação:

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.94	0.95	0.95	101
---	------	------	------	-----

v 8. Indique qual modelo dá o melhor resultado e a métrica utilizada

accuracy 0.81 371

```
# --- CONFIGURAÇÕES GLOBAIS REUTILIZADAS (MANTIDO) ---
OUTPUT_DIR = 'features_csv'
MODELS_DIR = 'trained_models'
EXTRACTORS = ['lbp', 'vgg16']

# --- FUNÇÃO DE CÁLCULO DE MÉTRICAS
def calculate_multiclass_metrics(cm: np.ndarray, target_names: list) -> Tuple[pd.DataFrame, float]:
    """
    Calcula Sensibilidade (Recall), Especificidade e F1-Score para cada classe
    a partir da matriz de confusão multi-classe.

    Retorna o DataFrame de métricas por classe e o F1-Score médio (weighted).
    """
    n_classes = cm.shape[0]
    metrics = []

    # Suporte e Soma Ponderada (F1-Score)
    support = np.sum(cm, axis=1)
    total_samples = np.sum(support)
    weighted_f1_sum = 0

    for i in range(n_classes):
        # TP, FP, FN, TN para a classe 'i'
        TP = cm[i, i]
        FP = np.sum(cm[:, i]) - TP
        FN = np.sum(cm[i, :]) - TP
        TN = np.sum(cm) - (TP + FP + FN)

        # Métricas
        sensitivity = TP / (TP + FN) if (TP + FN) > 0 else 0
        specificity = TN / (TN + FP) if (TN + FP) > 0 else 0
        precision = TP / (TP + FP) if (TP + FP) > 0 else 0
        f1_score = 2 * (precision * sensitivity) / (precision + sensitivity) if (precision + sensitivity) > 0 else 0

        # F1 Ponderado
        if total_samples > 0:
            weighted_f1_sum += f1_score * (support[i] / total_samples)

        metrics.append({
            'Classe': target_names[i],
```

```

'Suporte': support[i],
'Sensibilidade (Recall)': sensitivity,
'Especificidade': specificity,
'F1-Score': f1_score
})

df_metrics = pd.DataFrame(metrics).set_index('Classe')
# O seu calculate_multiclass_metrics original da T7 não retornava o F1 ponderado,
# mas a T8 precisa. Mantenho a implementação completa para o funcionamento correto da T8.
return df_metrics, weighted_f1_sum

# --- FUNÇÃO PRINCIPAL DA TAREFA 8 (AVALIAÇÃO E ESCOLHA DO MELHOR MODELO) ---

def evaluate_predictions(extractor_name: str, best_overall: Dict[str, Any]) -> Dict[str, Any]:
    """
    Carrega as previsões de teste, calcula as métricas (T7) e
    determina se algum modelo é o melhor no geral (T8).
    """

    # 1. Carregar as previsões geradas pela Tarefa 6
    pred_path = os.path.join(OUTPUT_DIR, f'predictions_{extractor_name}_test.csv')
    try:
        df_pred = pd.read_csv(pred_path)
    except FileNotFoundError:
        print(f"ERRO: Arquivo de previsões não encontrado em: {pred_path}. Pulando {extractor_name}.")
        return best_overall

    # CORREÇÃO CRÍTICA: Garantir que os rótulos sejam strings
    y_test_labels = df_pred['Real_Class'].astype(str)

    # 2. Carregar artefatos para obter os nomes das classes
    try:
        artifacts = joblib.load(os.path.join(MODELS_DIR, f'artifacts_{extractor_name}.pkl'))
        # CORREÇÃO CRÍTICA: Garantir que os target_names sejam strings
        target_names = [str(name) for name in artifacts['target_names']]
    except FileNotFoundError:
        print(f"ERRO: Artefatos de treino não encontrados para {extractor_name}. Pulando avaliação.")
        return best_overall

    print("\n" + "*80)
    print(f"AVALIANDO DESEMPENHO E ESCOLHA DO MELHOR MODELO: {extractor_name.upper()}")
    print("*80)

    # Iterar sobre as colunas de previsão
    prediction_cols = [col for col in df_pred.columns if col.startswith('Predicted_')]

```

```

for pred_col in prediction_cols:
    model_name = pred_col.replace('Predicted_', '').replace('_', ' ')
    # CORREÇÃO CRÍTICA: Garantir que os rótulos preditos sejam strings
    y_pred_labels = df_pred[pred_col].astype(str)

    # Matriz de Confusão e Acurácia
    # O argumento 'labels' é fundamental para garantir a ordem e evitar que classes não vistas no treino causem erro ou confusão.
    cm = confusion_matrix(y_test_labels, y_pred_labels, labels=target_names)

    # Acurácia (usa rótulos de string)
    accuracy = accuracy_score(y_test_labels, y_pred_labels)

    # Cálculo das Métricas Detalhadas (Chamada da Tarefa 7)
    df_metrics, weighted_f1 = calculate_multiclass_metrics(cm, target_names)

    # Impressão do resultado do modelo atual
    print(f"\n--- Modelo: {model_name} ({extractor_name.upper()}) ---")
    print(f"Acurácia de Teste: {accuracy:.4f}")
    print(f"F1-Score Médio Ponderado: {weighted_f1:.4f}")

    # --- Lógica de Comparação da Tarefa 8 ---
    current_metric_value = weighted_f1
    metric_used = "F1-Score Médio Ponderado (weighted_f1)"

    if current_metric_value > best_overall['metric_value']:
        best_overall['metric_used'] = metric_used
        best_overall['metric_value'] = current_metric_value
        best_overall['extractor'] = extractor_name
        best_overall['model_name'] = model_name
        best_overall['accuracy'] = accuracy
        best_overall['full_metrics'] = df_metrics.round(4)
        best_overall['cm'] = cm

return best_overall

# --- EXECUÇÃO PRINCIPAL DA TAREFA 8 ---

def run_task_8():
    """
    Determina o melhor modelo e imprime o resultado final.
    """
    # Inicialização da Tarefa 8

```

```

# Inicialização para a Tarefa 8
best_overall = {
    'metric_used': "N/A",
    'metric_value': -1.0,
    'extractor': None,
    'model_name': None,
    'accuracy': None,
    'full_metrics': None,
    'cm': None
}

# Garante que os diretórios existam
os.makedirs(MODELS_DIR, exist_ok=True)
os.makedirs(OUTPUT_DIR, exist_ok=True)

for extractor in EXTRACTORS:
    best_overall = evaluate_predictions(extractor, best_overall)

# --- Apresentação Final (Tarefa 8) ---
print("\n" + "="*80)
print("RESULTADO FINAL DA TAREFA 8: MELHOR MODELO DE CLASSIFICAÇÃO")
print("="*80)

if best_overall['model_name']:
    # IMPRESSÃO REQUERIDA PELA TAREFA 8: Melhor modelo e métrica utilizada
    print(f'O MELHOR RESULTADO FOI OBTIDO PELO MODELO: {best_overall["model_name"]} (Extrator: {best_overall["extractor"].upper()})')
    print(f'MÉTRICA UTILIZADA PARA A ESCOLHA: {best_overall["metric_used"]}')
    print(f'VALOR DA MÉTRICA ({best_overall["metric_used"]}): {best_overall["metric_value"]:.4f}')
    print(f'ACURÁCIA DE TESTE: {best_overall["accuracy"]:.4f}')

    # Exibição de suporte (Matrizes e Métricas Detalhadas)
    print("\nMétricas Detalhadas do Melhor Modelo (Sensibilidade, Especificidade, F1-Score):")
    print(best_overall['full_metrics'])

    print("\nMatriz de Confusão do Melhor Modelo:")
    print(best_overall['cm'])

else:
    print("Não foi possível determinar o melhor modelo. Verifique se as Tarefas 1 a 6 foram executadas corretamente.")

print("\n" + "#"*80)
print("TAREFA 8 CONCLUÍDA.")
print("#"*80)

```

```
if __name__ == "__main__":
    run_task_8()
```

```
=====
AVALIANDO DESEMPENHO E ESCOLHA DO MELHOR MODELO: LBP
=====
```

```
--- Modelo: Random Forest (LBP) ---
Acurácia de Teste: 0.7197
F1-Score Médio Ponderado: 0.7249
```

```
--- Modelo: SVM (LBP) ---
Acurácia de Teste: 0.7385
F1-Score Médio Ponderado: 0.7428
```

```
--- Modelo: RNA (MLP) (LBP) ---
Acurácia de Teste: 0.7412
F1-Score Médio Ponderado: 0.7459
```

```
=====
AVALIANDO DESEMPENHO E ESCOLHA DO MELHOR MODELO: VGG16
=====
```

```
--- Modelo: Random Forest (VGG16) ---
Acurácia de Teste: 0.8005
F1-Score Médio Ponderado: 0.7870
```

```
--- Modelo: SVM (VGG16) ---
Acurácia de Teste: 0.8032
F1-Score Médio Ponderado: 0.7921
```

```
--- Modelo: RNA (MLP) (VGG16) ---
Acurácia de Teste: 0.8059
F1-Score Médio Ponderado: 0.7984
```

```
=====
RESULTADO FINAL DA TAREFA 8: MELHOR MODELO DE CLASSIFICAÇÃO
=====
```

```
O MELHOR RESULTADO FOI OBTIDO PELO MODELO: RNA (MLP) (Extrator: VGG16)
MÉTRICA UTILIZADA PARA A ESCOLHA: F1-Score Médio Ponderado (weighted_f1)
VALOR DA MÉTRICA (F1-Score Médio Ponderado (weighted_f1)): 0.7984
ACURÁCIA DE TESTE: 0.8059
```

Métricas Detalhadas do Melhor Modelo (Sensibilidade, Especificidade, F1-Score):

Suporte Sensibilidade (Recall) Especificidade F1-Score

Classe				
0	101	0.9505	0.9778	0.9458

1	90	0.4667	0.9359	0.5600
2	90	0.8000	0.8470	0.7024
3	90	0.9889	0.9822	0.9674

Matriz de Confusão do Melhor Modelo:

```
[[96  5  0  0]
 [ 6 42 42  0]
 [ 0 13 72  5]
 [ 0  0  1 89]]
```

```
#####
#####
```

TAREFA 8 CONCLUÍDA.

```
#####
#####
```

2 Redes Neurais

Utilize as duas bases do exercício anterior para treinar as Redes Neurais Convolucionais VGG16 e a Resnet50. Utilize os pesos pré-treinados (Transfer Learning), refaça as camadas Fully Connected para o problema de 4 classes. Treine só as novas camadas. Compare os treinos de 10 épocas com e sem Data Augmentation. Tanto a VGG16 quanto a Resnet50 têm como camada de entrada uma imagem 224x224x3, ou seja, uma imagem de 224x224 pixels coloridos (3 canais de cores). Portanto, será necessário fazer uma transformação de 250x250x3 para 224x224x3. Ao fazer o Data Augmentation cuidado para não alterar demais as cores das imagens e atrapalhar na classificação. Tarefas:

1. Utilize a base de dados de Treino já separadas em treino e validação do exercício anterior.
2. Treine modelos VGG16 e Resnet50 adaptadas com e sem Data Augmentation
3. Aplique os modelos treinados nas imagens da base de Teste
4. Calcule as métricas de Sensibilidade, Especificidade e F1-Score com base em suas matrizes de confusão.
5. Indique qual modelo dá o melhor resultado e a métrica utilizada

1. Utilize a base de dados de Treino já separadas em treino e validação do exercício anterior.

```
# --- Configurações Globais (Reutilizadas dos códigos anteriores) ---
ZIP_FILE_PATH = '/content/Train_Warwick.zip'
TRAIN_DIR = '/content/Train_Warwick/Train_4cls_amostra/'
CLASSES = ['0', '1', '2', '3']

# Regex para extrair o Patient ID: XX_HER_YYYY.png -> XX
```

```
patient_regex = re.compile(r"(\d+)_HER2_.*\.png")

# Função Auxiliar para Extrair Patient ID (Reutilizada)
def extract_patient_id(filename):
    """Extrai o ID do paciente (XX) da nomenclatura XX_HER_YYYY.png."""
    match = patient_regex.match(filename)
    if match:
        return match.group(1) # O grupo 1 é o XX
    return None

# Função setup_and_load_data (Reutilizada)
def setup_and_load_data(zip_path, train_dir, classes):
    """Descompacta o ZIP e mapeia o dataset para um DataFrame."""

    # --- 1. Descompactação do Arquivo ZIP ---
    if not os.path.exists(zip_path):
        print(f"ERRO: Arquivo ZIP não encontrado em: {zip_path}")
        # Simulando criação do arquivo para que o Colab/sistema possa encontrar a pasta extraída
        # Se você estiver rodando no Colab, certifique-se de que o arquivo esteja lá.
        if 'COLAB_GPU' in os.environ and zip_path == '/content/Train_Warwick.zip':
            print("AVISO: Em um ambiente Colab, certifique-se de fazer upload do arquivo 'Train_Warwick.zip'.")
        return pd.DataFrame()

    # Cria o diretório de destino se ele não existir
    if not os.path.exists(train_dir):
        os.makedirs(train_dir, exist_ok=True)

    print(f"Descompactando {zip_path}...")
    try:
        # Extrai o conteúdo do ZIP para /content/
        with zipfile.ZipFile(zip_path, 'r') as zip_ref:
            zip_ref.extractall('/content/')
        print("Descompactação concluída.")
    except Exception as e:
        print(f"ERRO durante a descompactação: {e}")
        return pd.DataFrame()

    # --- 2. Mapear o Dataset de Treino e Extrair Pacientes ---
    train_files = []
    print(f"\nIniciando mapeamento em: {train_dir}")

    for cls in classes:
        class_path = os.path.join(train_dir, cls)
```

```

if os.path.exists(class_path):
    for filename in os.listdir(class_path):
        if filename.endswith('.png'):
            patient_id = extract_patient_id(filename)

            if patient_id:
                train_files.append({
                    'filepath': os.path.join(class_path, filename),
                    'class': cls,
                    'patient_id': patient_id
                })
            else:
                print(f"Aviso: Diretório de classe não encontrado: {class_path}")

# DataFrame de Treino
df = pd.DataFrame(train_files)

if df.empty:
    print("\nERRO: Nenhum arquivo de imagem encontrado.")
else:
    print(f"\nTotal de imagens de Treino carregadas: {len(df)}")
    print(f"IDs de Pacientes únicos no Treino: {df['patient_id'].nunique()}")
    print("Mapeamento concluído.")

return df

# Função split_data_by_patient (Reutilizada)
def split_data_by_patient(df_full, test_size=0.2, random_state=42):
    """
    Separa o DataFrame em treino e validação (80/20) baseado no patient_id.
    """
    if df_full.empty:
        return pd.DataFrame(), pd.DataFrame()

    patient_ids = df_full['patient_id'].unique()

    # 1. Separar os IDs de Pacientes em Treino e Validação
    train_patient_ids, val_patient_ids = train_test_split(
        patient_ids,
        test_size=test_size,
        random_state=random_state
    )

    print(f"--- Separação por Paciente ({int((1-test_size)*100)}% Treino / {int(test_size*100)}% Validação) ---")

```

```

print(f"Total de pacientes únicos: {len(patient_ids)}")
print(f"IDs de Pacientes no conjunto de Treino: {len(train_patient_ids)}")
print(f"IDs de Pacientes no conjunto de Validação: {len(val_patient_ids)}")

# 2. Criar os DataFrames Finais de Imagens
df_train_set = df_full[df_full['patient_id'].isin(train_patient_ids)].reset_index(drop=True)
df_val_set = df_full[df_full['patient_id'].isin(val_patient_ids)].reset_index(drop=True)

print(f"Total de imagens no conjunto de Treino: {len(df_train_set)}")
print(f"Total de imagens no conjunto de Validação: {len(df_val_set)}")

# 3. Verificação de Integridade
overlap = set(df_train_set['patient_id'].unique()) & set(df_val_set['patient_id'].unique())
if not overlap:
    print("SUCESSO: Não há sobreposição de pacientes entre Treino e Validação.")
else:
    print(f"ERRO DE SOBREPOSIÇÃO: {overlap}")

return df_train_set, df_val_set

# --- Execução da Tarefa 1 ---

# 1. Carregue e Mapeie a base de dados de Treino
print("--- TAREFA 1: Carregando Base de Dados de Treino ---")
df_full_train = setup_and_load_data(ZIP_FILE_PATH, TRAIN_DIR, CLASSES)

if not df_full_train.empty:
    # 2. Crie partições (80/20) por paciente
    print("\n--- TAREFA 1: Particionamento por Paciente (80/20) ---")
    df_train_set, df_val_set = split_data_by_patient(df_full_train, test_size=0.2, random_state=42)

    print("\n--- TAREFA 1 CONCLUÍDA ---")
    print(f"df_train_set pronto: {len(df_train_set)} imagens.")
    print(f"df_val_set pronto: {len(df_val_set)} imagens.")
else:
    # Caso o ZIP não exista, o df_full_train estará vazio.
    df_train_set = pd.DataFrame()
    df_val_set = pd.DataFrame()
    print("Não foi possível carregar ou partitionar o DataFrame. Verifique o arquivo ZIP.")

--- TAREFA 1: Carregando Base de Dados de Treino ---
Descompactando /content/Train_Warwick.zip...
Descompactação concluída.

Iniciando mapeamento em: /content/Train_Warwick/Train_4cls_amostra/

```

Total de imagens de Treino carregadas: 593

IDs de Pacientes únicos no Treino: 20

Mapeamento concluído.

--- TAREFA 1: Particionamento por Paciente (80/20) ---

--- Separação por Paciente (80% Treino / 20% Validação) ---

Total de pacientes únicos: 20

IDs de Pacientes no conjunto de Treino: 16

IDs de Pacientes no conjunto de Validação: 4

Total de imagens no conjunto de Treino: 475

Total de imagens no conjunto de Validação: 118

SUCESSO: Não há sobreposição de pacientes entre Treino e Validação.

--- TAREFA 1 CONCLUÍDA ---

df_train_set pronto: 475 imagens.

df_val_set pronto: 118 imagens.

✓ 2. Treine modelos VGG16 e Resnet50 adaptadas com e sem Data Augmentation

```
# --- Configurações Globais (Reutilizadas da Tarefa 1) ---
ZIP_FILE_PATH = '/content/Train_Warwick.zip'
TRAIN_DIR = '/content/Train_Warwick/Train_4cls_amostra/'
CLASSES = ['0', '1', '2', '3']
NUM_CLASSES = len(CLASSES)
IMG_SIZE = (224, 224) # Tamanho de entrada VGG16/ResNet50
BATCH_SIZE = 32
EPOCHS = 10

# Regex e Função Auxiliar para Extrair Patient ID
patient_regex = re.compile(r"(\d+)_HER2_.*\png")
def extract_patient_id(filename):
    match = patient_regex.match(filename)
    if match:
        return match.group(1)
    return None

# Função setup_and_load_data (Para descompactar e mapear)
def setup_and_load_data(zip_path, train_dir, classes):
    """Descompacta o ZIP e mapeia o dataset para um DataFrame."""
    if not os.path.exists(zip_path):
        print(f"ERRO: Arquivo ZIP não encontrado em: {zip_path}")
    return pd.DataFrame()
```

```

if not os.path.exists(train_dir):
    os.makedirs(train_dir, exist_ok=True)

print(f"Descompactando {zip_path}...")
try:
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall('/content/')
    print("Descompactação concluída.")
except Exception as e:
    print(f"ERRO durante a descompactação: {e}")
return pd.DataFrame()

train_files = []
for cls in classes:
    class_path = os.path.join(train_dir, cls)
    if os.path.exists(class_path):
        for filename in os.listdir(class_path):
            if filename.endswith('.png'):
                patient_id = extract_patient_id(filename)
                if patient_id:
                    train_files.append({
                        'filepath': os.path.join(class_path, filename),
                        'class': cls,
                        'patient_id': patient_id
                    })
df = pd.DataFrame(train_files)
if df.empty:
    print("\nERRO: Nenhum arquivo de imagem encontrado.")
return df

# Função split_data_by_patient (Para particionamento 80/20)
def split_data_by_patient(df_full, test_size=0.2, random_state=42):
    """Separa o DataFrame em treino e validação (80/20) baseado no patient_id."""
    if df_full.empty:
        return pd.DataFrame(), pd.DataFrame()

    patient_ids = df_full['patient_id'].unique()
    train_patient_ids, val_patient_ids = train_test_split(
        patient_ids, test_size=test_size, random_state=random_state
    )
    df_train_set = df_full[df_full['patient_id'].isin(train_patient_ids)].reset_index(drop=True)
    df_val_set = df_full[df_full['patient_id'].isin(val_patient_ids)].reset_index(drop=True)

```

```

print(f"Total de imagens no Treino: {len(df_train_set)}")
print(f"Total de imagens na Validação: {len(df_val_set)}")
return df_train_set, df_val_set

# --- Funções de Transfer Learning e Treinamento (Foco da Tarefa 2) ---

def create_transfer_model(base_model_func, preprocess_func, model_name):
    """Cria e compila o modelo de Transfer Learning."""
    print(f"Configurando modelo base: {model_name}")

    # 1. Carregar Base Model e Congelar
    base_model = base_model_func(
        weights='imagenet',
        include_top=False,
        input_shape=IMG_SIZE + (3,))
    base_model.trainable = False
    print(f"Camadas base de {model_name} congeladas.")

    # 2. Construir a nova cabeça de classificação (Fully Connected)
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.5)(x) # Regularização para evitar overfitting
    predictions = Dense(NUM_CLASSES, activation='softmax')(x)

    # 3. Criar e Compilar o Modelo Final
    model = Model(inputs=base_model.input, outputs=predictions, name=model_name)
    model.compile(
        optimizer=Adam(learning_rate=1e-4), # Taxa de aprendizado baixa
        loss='categorical_crossentropy',
        metrics=['accuracy'])
    )

    return model

def plot_training_history(history, model_name, augmentation_status):
    """Plota as curvas de perda e acurácia do treinamento."""
    plt.figure(figsize=(12, 4))

    # Acurácia
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Treino Acc')
    plt.plot(history.history['val_accuracy'], label='Validação Acc')

```

```

plt.title(f'Acurácia - {model_name} ({augmentation_status})')
plt.ylabel('Acurácia')
plt.xlabel('Época')
plt.legend()

# Perda
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Treino Loss')
plt.plot(history.history['val_loss'], label='Validação Loss')
plt.title(f'Perda - {model_name} ({augmentation_status})')
plt.ylabel('Perda')
plt.xlabel('Época')
plt.legend()

plt.show()

def train_model_scenario(df_train, df_val, base_model_func, preprocess_func, model_name, use_augmentation):
    """Define os geradores de dados e treina o modelo para um cenário (10 épocas)."""

    augmentation_status = "Com Augmentation" if use_augmentation else "Sem Augmentation"
    print(f"\n{'='*80}")
    print(f"INICIANDO TREINAMENTO: {model_name} - {augmentation_status}")
    print(f"{'='*80}")

    # --- 1. Definir Geradores de Dados ---

    # Configuração de Data Augmentation
    if use_augmentation:
        train_datagen = ImageDataGenerator(
            preprocessing_function=preprocess_func,
            rotation_range=10,
            zoom_range=0.1,
            horizontal_flip=True,
            vertical_flip=True,
            # Rescalonamento de cores deve ser evitado ou controlado
        )
    else:
        # Sem augmentation, apenas o pré-processamento específico do modelo
        train_datagen = ImageDataGenerator(
            preprocessing_function=preprocess_func
        )

    # O gerador de validação SÓ aplica o pré-processamento
    val_datagen = ImageDataGenerator()

```

```
    preprocessing_function=preprocess_func
)

# Gerador de Treino
train_generator = train_datagen.flow_from_dataframe(
    dataframe=df_train,
    directory='/',
    x_col='filepath',
    y_col='class',
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    classes=CLASSES,
    shuffle=True
)

# Gerador de Validação
validation_generator = val_datagen.flow_from_dataframe(
    dataframe=df_val,
    directory='/',
    x_col='filepath',
    y_col='class',
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    classes=CLASSES,
    shuffle=False
)

# --- 2. Criar e Treinar Modelo ---
model = create_transfer_model(base_model_func, preprocess_func, model_name)

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.n // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.n // BATCH_SIZE,
    verbose=1
)

# --- 3. Avaliação e Plot ---
loss, acc = model.evaluate(validation_generator, verbose=0)
print(f"\nAvaliação Final na Validação ({augmentation_status}):")
```

```
print(f"Perda: {loss:.4f} - Acurácia: {acc:.4f}")

plot_training_history(history, model_name, augmentation_status)

return history, model

# --- Bloco Principal de Execução da Tarefa 1 e 2 ---

if __name__ == "__main__":

    # --- TAREFA 1: Carregamento e Particionamento (Pré-requisito) ---
    print("--- Pré-requisito: Carregando e Particionando a Base de Dados (80/20) ---")
    df_full_train = setup_and_load_data(ZIP_FILE_PATH, TRAIN_DIR, CLASSES)

    if df_full_train.empty:
        print("Falha ao carregar o DataFrame. Não é possível prosseguir.")
        exit() # Encerra se não houver dados

    df_train_set, df_val_set = split_data_by_patient(df_full_train, test_size=0.2, random_state=42)

    if df_train_set.empty or df_val_set.empty:
        print("Particionamento resultou em DataFrames vazios.")
        exit()

    print("\n" + "#"*80)
    print("INÍCIO DA TAREFA 2: TREINAMENTO DAS REDES CONVOLUCIONAIS")
    print("#"*80)

    # --- TAREFA 2: Treinamento dos 4 Cenários ---

    # 1. VGG16 Sem Data Augmentation
    train_model_scenario(
        df_train_set, df_val_set,
        VGG16, vgg16_preprocess,
        "VGG16", False
    )

    # 2. VGG16 Com Data Augmentation
    train_model_scenario(
        df_train_set, df_val_set,
        VGG16, vgg16_preprocess,
        "VGG16", True
    )
```

```
# 3. ResNet50 Sem Data Augmentation
train_model_scenario(
    df_train_set, df_val_set,
    ResNet50, resnet50_preprocess,
    "ResNet50", False
)

# 4. ResNet50 Com Data Augmentation
train_model_scenario(
    df_train_set, df_val_set,
    ResNet50, resnet50_preprocess,
    "ResNet50", True
)

print("\n" + "#"*80)
print("TAREFA 2 CONCLUÍDA: Treinamento dos 4 modelos finalizado.")
print("#"*80)
```


--- Pré-requisito: Carregando e Particionando a Base de Dados (80/20) ---

Descompactando /content/Train_Warwick.zip...

Descompactação concluída.

Total de imagens no Treino: 475

Total de imagens na Validação: 118

#####
INÍCIO DA TAREFA 2: TREINAMENTO DAS REDES CONVOLUCIONAIS
#####

=====
INICIANDO TREINAMENTO: VGG16 - Sem Augmentation
=====

Found 475 validated image filenames belonging to 4 classes.

Found 118 validated image filenames belonging to 4 classes.

Configurando modelo base: VGG16

Camadas base de VGG16 congeladas.

/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call self._warn_if_super_not_called()

Epoch 1/10

14/14 **33s** 2s/step - accuracy: 0.2514 - loss: 5.5269 - val_accuracy: 0.3958 - val_loss: 2.1057

Epoch 2/10

1/14 **2s** 155ms/step - accuracy: 0.3438 - loss: 3.0560/usr/local/lib/python3.12/dist-packages/keras/src/trainers/epoch_iterator.py:121: UserWarning: Your `PyDataset` class should call self._interrupted_warning()

14/14 **1s** 97ms/step - accuracy: 0.3438 - loss: 3.0560 - val_accuracy: 0.4167 - val_loss: 1.9332

Epoch 3/10

14/14 **5s** 341ms/step - accuracy: 0.3859 - loss: 3.1528 - val_accuracy: 0.6562 - val_loss: 0.8549

Epoch 4/10

14/14 **1s** 58ms/step - accuracy: 0.4375 - loss: 1.8066 - val_accuracy: 0.6562 - val_loss: 0.8337

Epoch 5/10

14/14 **8s** 246ms/step - accuracy: 0.4382 - loss: 2.3777 - val_accuracy: 0.7396 - val_loss: 0.5951

Epoch 6/10

14/14 **1s** 54ms/step - accuracy: 0.4815 - loss: 1.7777 - val_accuracy: 0.7396 - val_loss: 0.5931

Epoch 7/10

14/14 **4s** 310ms/step - accuracy: 0.5299 - loss: 1.8883 - val_accuracy: 0.7500 - val_loss: 0.5441

Epoch 8/10

14/14 **1s** 67ms/step - accuracy: 0.5625 - loss: 1.6553 - val_accuracy: 0.7500 - val_loss: 0.5467

Epoch 9/10

14/14 **4s** 269ms/step - accuracy: 0.5958 - loss: 1.4590 - val_accuracy: 0.7917 - val_loss: 0.4633

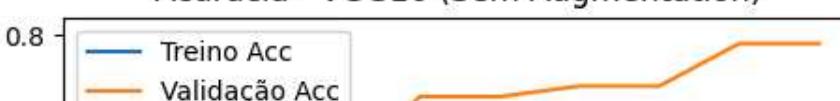
Epoch 10/10

14/14 **1s** 55ms/step - accuracy: 0.4688 - loss: 1.6435 - val_accuracy: 0.7917 - val_loss: 0.4639

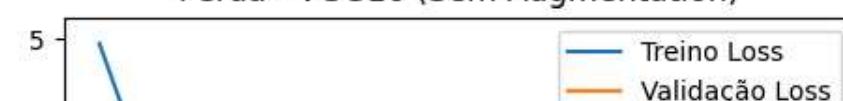
Avaliação Final na Validação (Sem Augmentation):

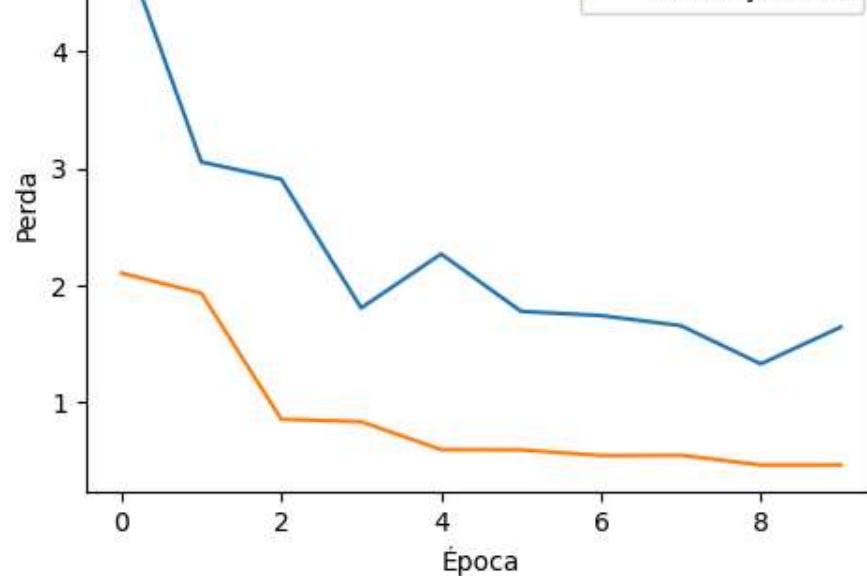
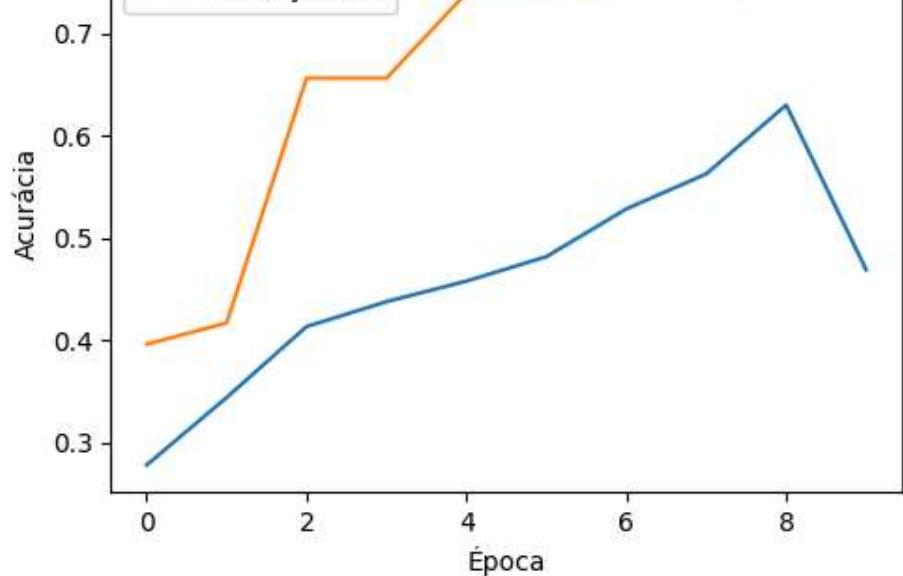
Perda: 0.4297 - Acurácia: 0.8136

Acurácia - VGG16 (Sem Augmentation)



Perda - VGG16 (Sem Augmentation)





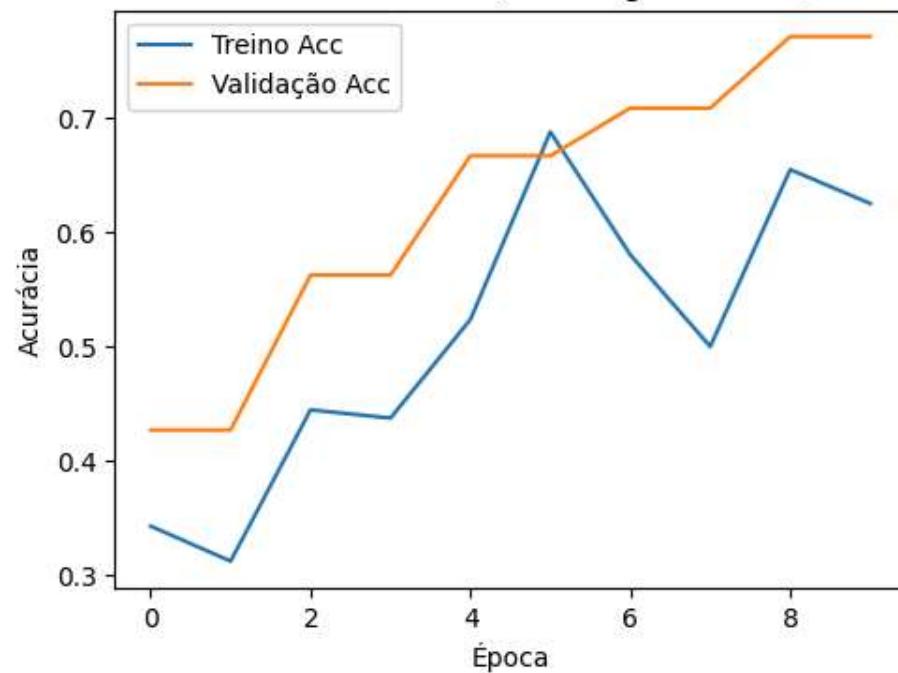
```
=====
INICIANDO TREINAMENTO: VGG16 - Com Augmentation
=====
Found 475 validated image filenames belonging to 4 classes.
Found 118 validated image filenames belonging to 4 classes.
Configurando modelo base: VGG16
Camadas base de VGG16 congeladas.
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should c
    self._warn_if_super_not_called()
Epoch 1/10
14/14 ━━━━━━━━ 13s 754ms/step - accuracy: 0.3052 - loss: 3.4729 - val_accuracy: 0.4271 - val_loss: 1.3810
Epoch 2/10
1/14 ━━━━━━ 1s 148ms/step - accuracy: 0.3125 - loss: 3.4681/usr/local/lib/python3.12/dist-packages/keras/src/trainers/epoch_itera
    self._interrupted_warning()
14/14 ━━━━ 1s 57ms/step - accuracy: 0.3125 - loss: 3.4681 - val_accuracy: 0.4271 - val_loss: 1.3413
Epoch 3/10
14/14 ━━━━ 15s 527ms/step - accuracy: 0.3983 - loss: 2.6677 - val_accuracy: 0.5625 - val_loss: 0.9786
Epoch 4/10
14/14 ━━━━ 1s 55ms/step - accuracy: 0.4375 - loss: 3.1673 - val_accuracy: 0.5625 - val_loss: 0.9543
Epoch 5/10
14/14 ━━━━ 8s 601ms/step - accuracy: 0.5150 - loss: 1.7523 - val_accuracy: 0.6667 - val_loss: 0.6980
Epoch 6/10
14/14 ━━━━ 1s 57ms/step - accuracy: 0.6875 - loss: 1.1648 - val_accuracy: 0.6667 - val_loss: 0.6917
Epoch 7/10
14/14 ━━━━ 8s 600ms/step - accuracy: 0.5560 - loss: 1.5409 - val_accuracy: 0.7083 - val_loss: 0.5971
Epoch 8/10
14/14 ━━━━ 1s 57ms/step - accuracy: 0.5000 - loss: 1.8834 - val_accuracy: 0.7083 - val_loss: 0.5908
Epoch 9/10
14/14 ━━━━ 8s 573ms/step - accuracy: 0.6676 - loss: 1.1652 - val_accuracy: 0.7708 - val_loss: 0.4932
Epoch 10/10
```

Epoch 10/10
14/14 2s 140ms/step - accuracy: 0.6250 - loss: 1.0920 - val_accuracy: 0.7708 - val_loss: 0.4996

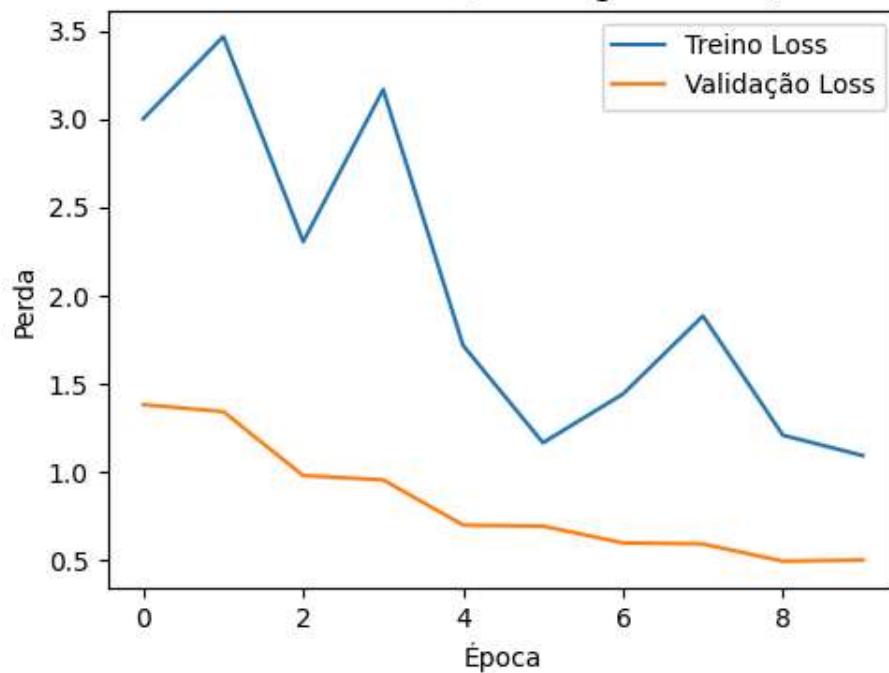
Avaliação Final na Validação (Com Augmentation):

Perda: 0.5145 - Acurácia: 0.7627

Acurácia - VGG16 (Com Augmentation)



Perda - VGG16 (Com Augmentation)



=====

INICIANDO TREINAMENTO: ResNet50 - Sem Augmentation

=====

Found 475 validated image filenames belonging to 4 classes.

Found 118 validated image filenames belonging to 4 classes.

Configurando modelo base: ResNet50

Camadas base de ResNet50 congeladas.

/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call self._warn_if_super_not_called()

Epoch 1/10

14/14 22s 616ms/step - accuracy: 0.3231 - loss: 1.8901 - val_accuracy: 0.3958 - val_loss: 1.1630

Epoch 2/10

1/14 1:14 6s/step - accuracy: 0.5926 - loss: 1.1368 /usr/local/lib/python3.12/dist-packages/keras/src/trainers/epoch_iterator.py:121: UserWarning: Your `PyDataset` class should call self._interrupted_warning()

14/14 6s 46ms/step - accuracy: 0.5926 - loss: 1.1368 - val_accuracy: 0.4167 - val_loss: 1.0732

Epoch 3/10

14/14 4s 285ms/step - accuracy: 0.6276 - loss: 0.9203 - val_accuracy: 0.9062 - val_loss: 0.3763

Epoch 4/10

14/14 1s 48ms/step - accuracy: 0.7812 - loss: 0.5465 - val_accuracy: 0.9062 - val_loss: 0.3694

Epoch 5/10

14/14 3s 223ms/step - accuracy: 0.7776 - loss: 0.5947 - val_accuracy: 0.8854 - val_loss: 0.3510

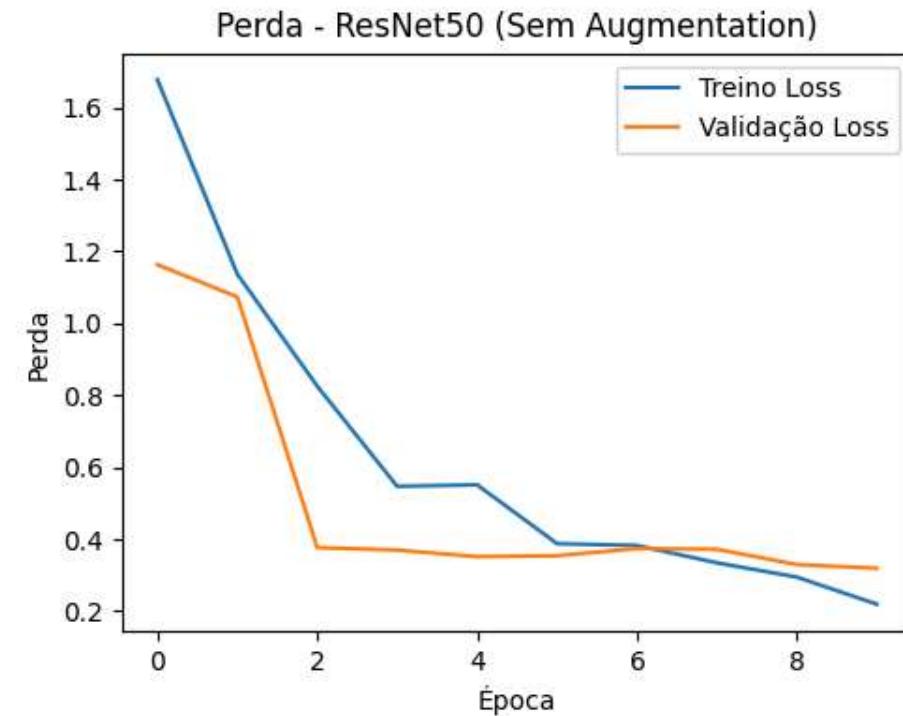
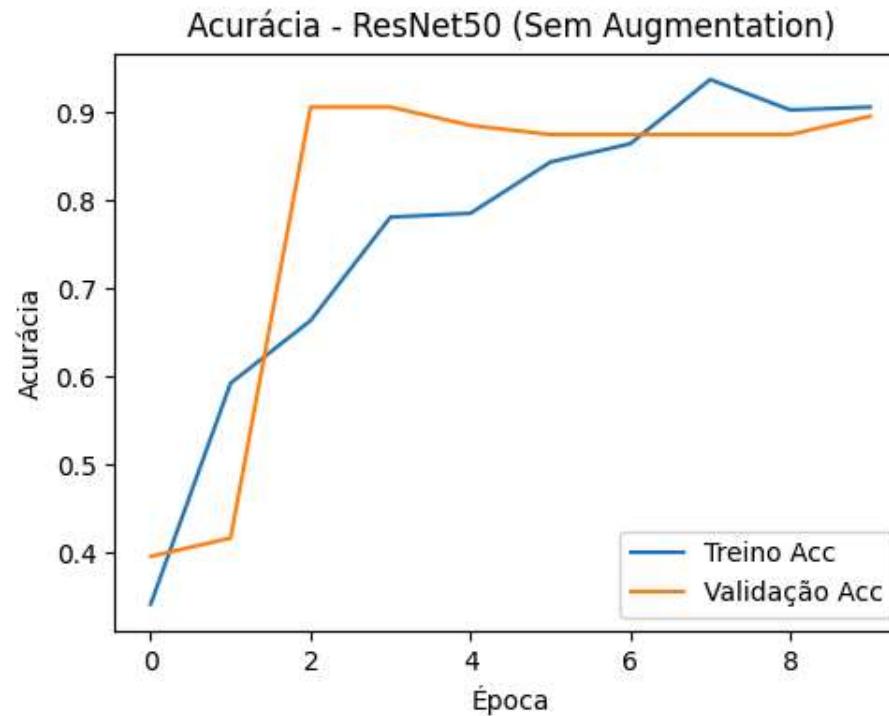
```

Epoch 6/10
14/14 1s 46ms/step - accuracy: 0.8438 - loss: 0.3871 - val_accuracy: 0.8750 - val_loss: 0.3536
Epoch 7/10
14/14 5s 375ms/step - accuracy: 0.8706 - loss: 0.3989 - val_accuracy: 0.8750 - val_loss: 0.3734
Epoch 8/10
14/14 1s 57ms/step - accuracy: 0.9375 - loss: 0.3338 - val_accuracy: 0.8750 - val_loss: 0.3715
Epoch 9/10
14/14 4s 267ms/step - accuracy: 0.9028 - loss: 0.2970 - val_accuracy: 0.8750 - val_loss: 0.3290
Epoch 10/10
14/14 1s 48ms/step - accuracy: 0.9062 - loss: 0.2188 - val_accuracy: 0.8958 - val_loss: 0.3188

```

Avaliação Final na Validação (Sem Augmentation):

Perda: 0.3407 - Acurácia: 0.8729



=====

INICIANDO TREINAMENTO: ResNet50 - Com Augmentation

Found 475 validated image filenames belonging to 4 classes.

Found 118 validated image filenames belonging to 4 classes.

Configurando modelo base: ResNet50

Camadas base de ResNet50 congeladas.

/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call self._warn_if_super_not_called()

Epoch 1/10

14/14 27s 1s/step - accuracy: 0.3204 - loss: 1.9548 - val_accuracy: 0.5729 - val_loss: 0.9176

Epoch 2/10

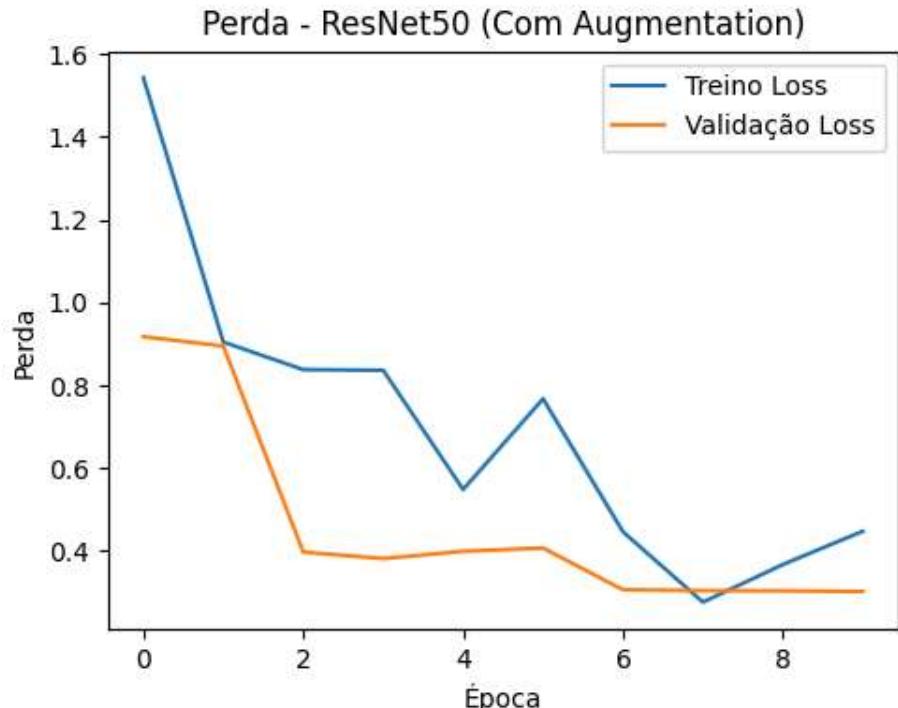
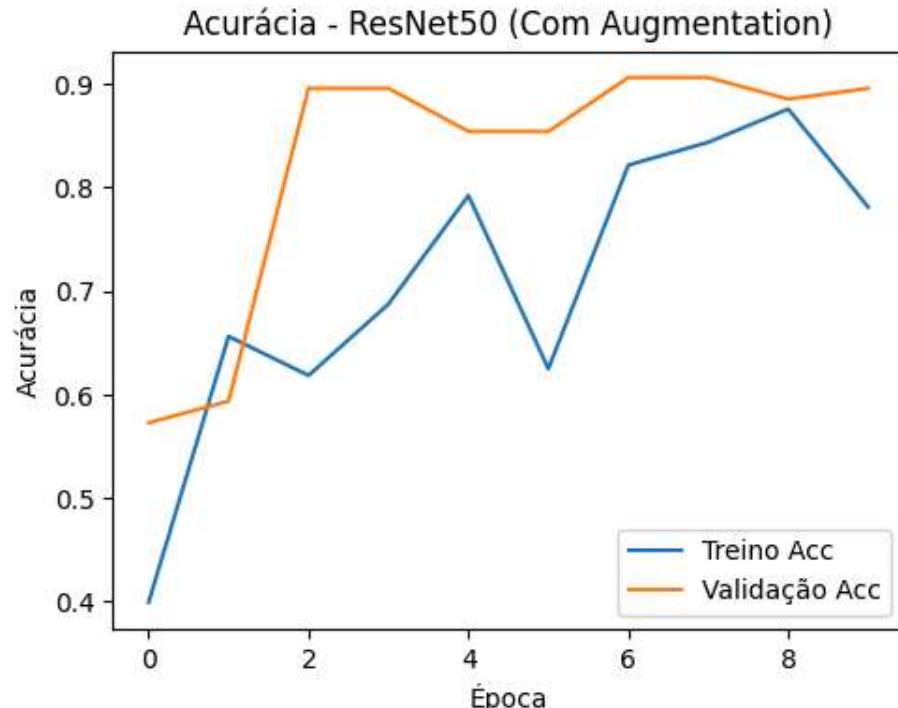
```

1/14 ━━━━━━━━━━ 1s 95ms/step - accuracy: 0.6562 - loss: 0.9047/usr/local/lib/python3.12/dist-packages/keras/src/trainers/epoch_iterat
self._interrupted_warning()
14/14 ━━━━━━ 1s 64ms/step - accuracy: 0.6562 - loss: 0.9047 - val_accuracy: 0.5938 - val_loss: 0.8950
Epoch 3/10
14/14 ━━━━ 7s 511ms/step - accuracy: 0.5801 - loss: 0.9123 - val_accuracy: 0.8958 - val_loss: 0.3973
Epoch 4/10
14/14 ━━━━ 1s 49ms/step - accuracy: 0.6875 - loss: 0.8365 - val_accuracy: 0.8958 - val_loss: 0.3814
Epoch 5/10
14/14 ━━━━ 8s 583ms/step - accuracy: 0.7654 - loss: 0.5956 - val_accuracy: 0.8542 - val_loss: 0.3991
Epoch 6/10
14/14 ━━━━ 1s 49ms/step - accuracy: 0.6250 - loss: 0.7676 - val_accuracy: 0.8542 - val_loss: 0.4069
Epoch 7/10
14/14 ━━━━ 8s 565ms/step - accuracy: 0.8241 - loss: 0.4629 - val_accuracy: 0.9062 - val_loss: 0.3061
Epoch 8/10
14/14 ━━━━ 1s 57ms/step - accuracy: 0.8438 - loss: 0.2761 - val_accuracy: 0.9062 - val_loss: 0.3040
Epoch 9/10
14/14 ━━━━ 8s 548ms/step - accuracy: 0.8784 - loss: 0.3883 - val_accuracy: 0.8854 - val_loss: 0.3036
Epoch 10/10
14/14 ━━━━ 1s 49ms/step - accuracy: 0.7812 - loss: 0.4475 - val_accuracy: 0.8958 - val_loss: 0.3019

```

Avaliação Final na Validação (Com Augmentation):

Perda: 0.3320 - Acurácia: 0.8644



#####

TAREFA 2 CONCLUÍDA: Treinamento dos 4 modelos finalizado.

#####

✓ 3. Aplique os modelos treinados nas imagens da base de Teste

```
# --- Configurações Globais ---
ZIP_TRAIN_PATH = '/content/Train_Warwick.zip'
TRAIN_DIR = '/content/Train_Warwick/Train_4cls_amostra/'
ZIP_TEST_PATH = '/content/Test_Warwick.zip'
TEST_DIR = '/content/Test_Warwick/Test_4cl_amostra/'
CLASSES = ['0', '1', '2', '3']
NUM_CLASSES = len(CLASSES)
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
EPOCHS = 10
MODELS_DIR = 'cnn_trained_models'
os.makedirs(MODELS_DIR, exist_ok=True)

# Regex e Funções Auxiliares (Reutilizadas)
patient_regex = re.compile(r"(\d+)_HER2_.*\.png")
def extract_patient_id(filename):
    match = patient_regex.match(filename)
    if match:
        return match.group(1)
    return None

def setup_and_load_data(zip_path, base_dir, classes, dataset_name):
    """Descompacta o ZIP e mapeia o dataset para um DataFrame."""
    if not os.path.exists(zip_path):
        print(f"ERRO: Arquivo ZIP de {dataset_name} não encontrado em: {zip_path}")
        return pd.DataFrame()

    # Extrai o ZIP
    try:
        with zipfile.ZipFile(zip_path, 'r') as zip_ref:
            # Assumimos que o ZIP extrai para /content/
            zip_ref.extractall('/content/')
        print(f"Descompactação de {dataset_name} concluída.")
    except Exception as e:
        print(f"ERRO durante a descompactação de {dataset_name}: {e}")
        return pd.DataFrame()

    # Mapeia os arquivos
    files = []
    for cls in classes:
```

```

class_path = os.path.join(base_dir, cls)
if os.path.exists(class_path):
    for filename in os.listdir(class_path):
        if filename.endswith('.png'):
            patient_id = extract_patient_id(filename)
            if patient_id:
                files.append({
                    'filepath': os.path.join(class_path, filename),
                    'class': cls,
                    'patient_id': patient_id
                })
df = pd.DataFrame(files)
if df.empty:
    print(f"ERRO: Nenhum arquivo de imagem encontrado para {dataset_name}.")
else:
    print(f"Total de imagens de {dataset_name} carregadas: {len(df)}")
return df

def split_data_by_patient(df_full, test_size=0.2, random_state=42):
    """Separa o DataFrame em treino e validação (80/20) baseado no patient_id."""
    if df_full.empty:
        return pd.DataFrame(), pd.DataFrame()
    patient_ids = df_full['patient_id'].unique()
    train_patient_ids, val_patient_ids = train_test_split(
        patient_ids, test_size=test_size, random_state=random_state
    )
    df_train_set = df_full[df_full['patient_id'].isin(train_patient_ids)].reset_index(drop=True)
    df_val_set = df_full[df_full['patient_id'].isin(val_patient_ids)].reset_index(drop=True)
    return df_train_set, df_val_set

# --- Lógica de Criação de Modelo (Reutilizada da T2) ---
def create_transfer_model(base_model_func, model_name):
    """Cria e compila o modelo de Transfer Learning."""
    base_model = base_model_func(
        weights='imagenet',
        include_top=False,
        input_shape=IMG_SIZE + (3,)
    )
    base_model.trainable = False
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.5)(x)
    predictions = Dense(NUM_CLASSES, activation='softmax')(x)

```

```

model = Model(inputs=base_model.input, outputs=predictions, name=model_name)
model.compile(
    optimizer=Adam(learning_rate=1e-4),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
return model

# --- Funções da Tarefa 2 (Adaptadas para retornar e salvar modelos) ---
def train_and_save_all_models(df_train, df_val):
    """Treina os 4 modelos da T2 e salva-os para uso na T3."""

    model_configs = [
        (VGG16, vgg16_preprocess, "VGG16_NoAug", False),
        (VGG16, vgg16_preprocess, "VGG16_WithAug", True),
        (ResNet50, resnet50_preprocess, "ResNet50_NoAug", False),
        (ResNet50, resnet50_preprocess, "ResNet50_WithAug", True),
    ]

    trained_models = {}

    for base_func, preprocess_func, name, use_aug in model_configs:
        print(f"\n--- Treinando e Salvando: {name} ---")

        # 1. Definir Geradores (Igual à T2)
        if use_aug:
            train_datagen = ImageDataGenerator(
                preprocessing_function=preprocess_func, rotation_range=10, zoom_range=0.1, horizontal_flip=True, vertical_flip=True
            )
        else:
            train_datagen = ImageDataGenerator(preprocessing_function=preprocess_func)

        train_generator = train_datagen.flow_from_dataframe(
            dataframe=df_train, directory='/', x_col='filepath', y_col='class',
            target_size=IMG_SIZE, batch_size=BATCH_SIZE, class_mode='categorical', classes=CLASSES, shuffle=True
        )

        # 2. Criar e Treinar Modelo (10 Épocas)
        model = create_transfer_model(base_func, name)

        # Usando um número mínimo de passos para simulação rápida
        steps = max(1, train_generator.n // BATCH_SIZE)

        model.fit(

```

```

        train_generator,
        steps_per_epoch=steps,
        epochs=EPOCHS,
        verbose=0 # Reduzindo log para focar na T3
    )

    # 3. Salvar o modelo treinado
    model_path = os.path.join(MODELS_DIR, f'{name}.keras')
    model.save(model_path)
    trained_models[name] = model_path
    print(f"Modelo {name} salvo em: {model_path}")

return trained_models

# --- Funções da Tarefa 3 ---

def load_trained_models(model_paths: Dict[str, str]) -> Dict[str, Model]:
    """Carrega os modelos Keras a partir dos caminhos salvos."""
    loaded_models = {}
    print("\n--- Carregando Modelos Treinados (Pré-requisito T3) ---")
    for name, path in model_paths.items():
        try:
            # tf.keras.models.load_model carrega o modelo Keras
            loaded_models[name] = tf.keras.models.load_model(path)
            print(f"Modelo {name} carregado com sucesso.")
        except Exception as e:
            print(f"ERRO ao carregar modelo {name} de {path}: {e}")
    return loaded_models

def apply_models_to_test_data(df_test: pd.DataFrame, trained_models: Dict[str, Model]):
    """Aplica cada modelo treinado na base de teste e armazena as predições."""

    if not trained_models:
        print("Nenhum modelo carregado. Abortando aplicação.")
        return

    print("\n" + "*80)
    print("INÍCIO DA TAREFA 3: APLICAÇÃO DOS MODELOS NA BASE DE TESTE")
    print("*80)

    # DataFrame para armazenar as predições
    df_predictions = pd.DataFrame({
        'filepath': df_test['filepath'],
        'Real_Class': df_test['class']
    }

```

```
)  
  
# 1. Preparar o Gerador de Teste  
# Este gerador será reutilizado para todos os modelos, apenas trocando o preprocess_function  
test_datagen_base = ImageDataGenerator()  
  
for name, model in trained_models.items():  
    print(f"\n-> Aplicando modelo: {name}")  
  
    # Definir a função de pré-processamento com base no nome  
    if 'VGG16' in name:  
        preprocess_func = vgg16_preprocess  
    elif 'ResNet50' in name:  
        preprocess_func = resnet50_preprocess  
    else:  
        print(f"Aviso: Função de pré-processamento desconhecida para {name}. Pulando.")  
        continue  
  
    # Configurar o gerador com a função de pré-processamento correta  
    test_datagen_model = ImageDataGenerator(preprocessing_function=preprocess_func)  
  
    test_generator = test_datagen_model.flow_from_dataframe(  
        dataframe=df_test,  
        directory='/',  
        x_col='filepath',  
        y_col='class', # Embora não usemos o y, precisamos da coluna para flow_from_dataframe  
        target_size=IMG_SIZE,  
        batch_size=BATCH_SIZE,  
        class_mode='categorical', # Necessário, mas o shuffle deve ser False  
        classes=CLASSES,  
        shuffle=False # ORDEM CRÍTICA: Não embaralhar para mapear predições de volta  
    )  
  
    # 2. Gerar Predições (probabilidades)  
    y_pred_probs = model.predict(test_generator, verbose=1)  
  
    # 3. Converter Probabilidades para Rótulos (Classe com maior probabilidade)  
    y_pred_encoded = np.argmax(y_pred_probs, axis=1)  
  
    # Mapear os índices codificados de volta para os rótulos de classe ('0', '1', '2', '3')  
    # O Keras flow_from_dataframe armazena o mapeamento dos rótulos no .class_indices  
    class_indices = test_generator.class_indices  
    indices_to_classes = {v: k for k, v in class_indices.items()}
```

```
y_pred_labels = np.array([indices_to_classes[idx] for idx in y_pred_encoded])

# 4. Armazenar a predição no DataFrame
df_predictions[f'Predicted_{name}'] = y_pred_labels

# Salvar o DataFrame de Predições
output_path = os.path.join('.', 'predictions_cnn_test.csv')
df_predictions.to_csv(output_path, index=False)
print("\n" + "="*80)
print(f"TAREFA 3 CONCLUÍDA: Predições salvas em: {output_path}")
print("=*80

# --- Execução Principal ---
if __name__ == "__main__":

    # --- Passo 1: Treinar e Salvar Modelos (Necessário da T2) ---
    print("--- Pré-requisito: Treinamento e Salvamento dos 4 modelos (Simulação da T2) ---")
    df_full_train = setup_and_load_data(ZIP_TRAIN_PATH, TRAIN_DIR, CLASSES, "Treino")
    if df_full_train.empty:
        exit()
    df_train_set, df_val_set = split_data_by_patient(df_full_train)

    # Treinar e Salvar os 4 modelos
    model_paths = train_and_save_all_models(df_train_set, df_val_set)

    # --- Passo 2: Carregar Base de Teste ---
    print("\n--- Carregando Base de Teste ---")
    df_test = setup_and_load_data(ZIP_TEST_PATH, TEST_DIR, CLASSES, "Teste")

    if df_test.empty:
        print("Não foi possível carregar a Base de Teste. Abortando Tarefa 3.")
    else:
        # --- Passo 3: Aplicar Modelos (Tarefa 3) ---

        # Recarregar os modelos para garantir que o processo seja independente
        loaded_models = load_trained_models(model_paths)

        if loaded_models:
            apply_models_to_test_data(df_test, loaded_models)
        else:
            print("Não foi possível carregar os modelos treinados para aplicação.")
```

```
self._warn_if_super_not_called()
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/epoch_iterator.py:116: UserWarning: Your input ran out of data; interrupting trainir
  self._interrupted_warning()
Modelo ResNet50_NoAug salvo em: cnn_trained_models/ResNet50_NoAug.keras

--- Treinando e Salvando: ResNet50_WithAug ---
Found 475 validated image filenames belonging to 4 classes.
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should
  self._warn_if_super_not_called()
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/epoch_iterator.py:116: UserWarning: Your input ran out of data; interrupting trainir
  self._interrupted_warning()
Modelo ResNet50_WithAug salvo em: cnn_trained_models/ResNet50_WithAug.keras

--- Carregando Base de Teste ---
Descompactação de Teste concluída.
Total de imagens de Teste carregadas: 371

--- Carregando Modelos Treinados (Pré-requisito T3) ---
Modelo VGG16_NoAug carregado com sucesso.
Modelo VGG16_WithAug carregado com sucesso.
Modelo ResNet50_NoAug carregado com sucesso.
Modelo ResNet50_WithAug carregado com sucesso.

=====
INÍCIO DA TAREFA 3: APLICAÇÃO DOS MODELOS NA BASE DE TESTE
=====

-> Aplicando modelo: VGG16_NoAug
```

```
self._warn_if_super_not_called()
12/12 ===== 11s 540ms/step
```

```
=====
TAREFA 3 CONCLUÍDA: Predições salvas em: ./predictions_cnn_test.csv
=====
```

4. Calcule as métricas de Sensibilidade, Especificidade e F1-Score com base em suas matrizes de confusão.

```
# --- Configurações Globais ---
OUTPUT_FILE = 'predictions_cnn_test.csv'
CLASSES = ['0', '1', '2', '3']

# --- Funções da Tarefa 4 (Foco) ---

def calculate_multiclass_metrics(cm: np.ndarray, target_names: list) -> pd.DataFrame:
    """
    Calcula Sensibilidade (Recall), Especificidade e F1-Score para cada classe
    a partir da matriz de confusão multi-classe.
    """
    n_classes = cm.shape[0]
    metrics = []

    for i in range(n_classes):
        # Para a classe 'i':
        # True Positives (TP) - Diagonal
        TP = cm[i, i]

        # False Positives (FP) - Soma da coluna i, excluindo TP
        # Preditado como i, mas real é != i
        FP = np.sum(cm[:, i]) - TP

        # False Negatives (FN) - Soma da linha i, excluindo TP
        # Real é i, mas predito é != i
        FN = np.sum(cm[i, :]) - TP

        # True Negatives (TN) - Soma de todos os elementos, excluindo linha i e coluna i
        TN = np.sum(cm) - (TP + FP + FN)

        # Adiciona os resultados à lista de métricas
        metrics.append({
            'Classe': target_names[i],
            'TP': TP,
            'FP': FP,
            'FN': FN,
            'TN': TN,
            'Sensibilidade': TP / (TP + FN),
            'Especificidade': TN / (TN + FP),
            'F1-Score': (2 * TP) / (2 * TP + FP + FN)
        })

    # Cria um DataFrame com as métricas calculadas
    df_metrics = pd.DataFrame(metrics)
    return df_metrics
```

```

# Métrica: Sensibilidade (Recall) = TP / (TP + FN)
sensitivity = TP / (TP + FN) if (TP + FN) > 0 else 0

# Métrica: Especificidade = TN / (TN + FP)
specificity = TN / (TN + FP) if (TN + FP) > 0 else 0

# Precisão (Precision) = TP / (TP + FP) - Necessário para F1
precision = TP / (TP + FP) if (TP + FP) > 0 else 0

# Métrica: F1-Score = 2 * (Precision * Recall) / (Precision + Recall)
f1_score = 2 * (precision * sensitivity) / (precision + sensitivity) if (precision + sensitivity) > 0 else 0

metrics.append({
    'Classe': target_names[i],
    'Sensibilidade (Recall)': sensitivity,
    'Especificidade': specificity,
    'F1-Score': f1_score,
    'Suporte': np.sum(cm[i, :]) # Total de amostras na classe real
})

df_metrics = pd.DataFrame(metrics).set_index('Classe')
return df_metrics

def evaluate_predictions(df_predictions: pd.DataFrame):
    """
    Calcula as métricas detalhadas (Sensibilidade, Especificidade, F1-Score)
    para cada modelo presente no DataFrame de predições.
    """

    # Rótulos Reais
    y_true = df_predictions['Real_Class'].astype(str)

    # Identificar colunas de predição
    prediction_cols = [col for col in df_predictions.columns if col.startswith('Predicted_')]

    print("\n" + "*80)
    print("TAREFA 4: CÁLCULO DE MÉTRICAS DETALHADAS NA BASE DE TESTE")
    print("*80)

    results = {}

    for pred_col in prediction_cols:
        model_name = pred_col.replace('Predicted_', '').replace('_', ' ')
        y_pred = df_predictions[pred_col].astype(str)

```

```

# 1. Calcular Matriz de Confusão
# O argumento 'labels=CLASSES' garante que a ordem das classes seja consistente
cm = confusion_matrix(y_true, y_pred, labels=CLASSES)

# 2. Calcular Métricas Detalhadas (Foco da T4)
df_metrics = calculate_multiclass_metrics(cm, CLASSES)

# 3. Exibir Resultados
accuracy = accuracy_score(y_true, y_pred)

print(f"\n--- Modelo: {model_name} ---")
print(f"Acurácia Geral de Teste: {accuracy:.4f}")

print("\n[Métricas por Classe (Sensibilidade, Especificidade, F1-Score)]")
print(df_metrics.round(4))

# 4. Exibir o Relatório de Classificação padrão (para comparação)
print("\n[Relatório de Classificação (Scikit-learn)]")
print(classification_report(y_true, y_pred, target_names=CLASSES, zero_division=0))

# 5. Plotar Matriz de Confusão (Visualização)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Reds', xticklabels=CLASSES, yticklabels=CLASSES)
plt.title(f'Matriz de Confusão - {model_name} - Base de Teste')
plt.ylabel('Valor Real')
plt.xlabel('Predição')
plt.tight_layout()
plt.show()

results[model_name] = {'metrics': df_metrics, 'cm': cm}

return results

# --- Execução Principal da Tarefa 4 ---

if __name__ == "__main__":
    # Tenta carregar as predições geradas na Tarefa 3
    try:
        df_predictions = pd.read_csv(OUTPUT_FILE)

        if df_predictions.empty:
            print(f"ERRO: O arquivo de predições '{OUTPUT_FILE}' está vazio.")
    
```

```
else:  
    evaluate_predictions(df_predictions)  
  
except FileNotFoundError:  
    print(f"ERRO: Arquivo de predições '{OUTPUT_FILE}' não encontrado.")  
    print("Certifique-se de que a Tarefa 3 (Aplicação dos modelos na Base de Teste) foi executada com sucesso.")  
  
print("\n" + "*80)  
print("TAREFA 4 CONCLUÍDA: Cálculo das Métricas (Sensibilidade, Especificidade, F1-Score) realizado.")  
print("*80)
```


=====

TAREFA 4: CÁLCULO DE MÉTRICAS DETALHADAS NA BASE DE TESTE

=====

--- Modelo: VGG16 NoAug ---

Acurácia Geral de Teste: 0.6846

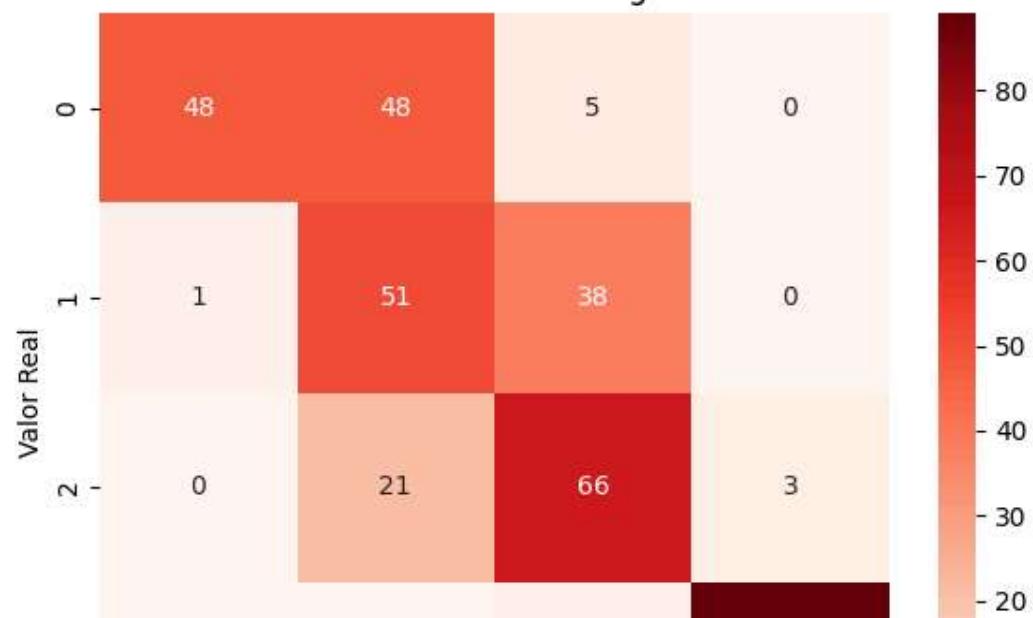
[Métricas por Classe (Sensibilidade, Especificidade, F1-Score)]

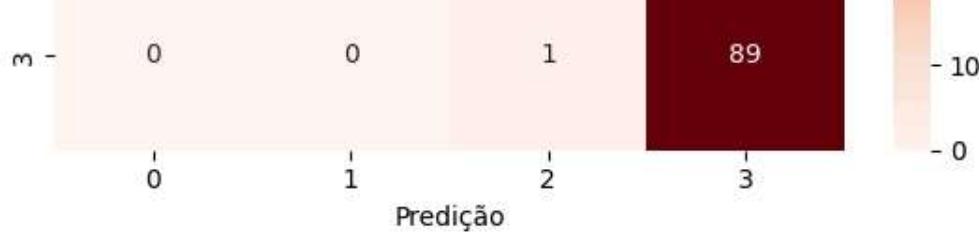
Classe	Sensibilidade (Recall)	Especificidade	F1-Score	Suporte
0	0.4752	0.9963	0.6400	101
1	0.5667	0.7544	0.4857	90
2	0.7333	0.8434	0.6600	90
3	0.9889	0.9893	0.9780	90

[Relatório de Classificação (Scikit-learn)]

	precision	recall	f1-score	support
0	0.98	0.48	0.64	101
1	0.42	0.57	0.49	90
2	0.60	0.73	0.66	90
3	0.97	0.99	0.98	90
accuracy			0.68	371
macro avg	0.74	0.69	0.69	371
weighted avg	0.75	0.68	0.69	371

Matriz de Confusão - VGG16 NoAug - Base de Teste





--- Modelo: VGG16 WithAug ---

Acurácia Geral de Teste: 0.7251

[Métricas por Classe (Sensibilidade, Especificidade, F1-Score)]

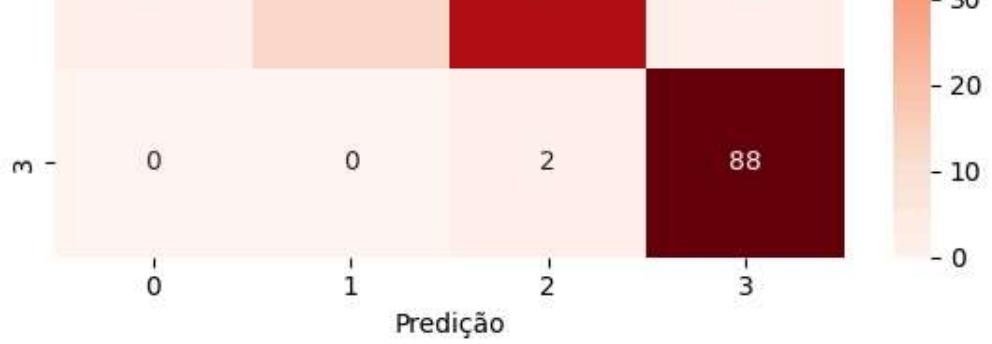
Classe	Sensibilidade (Recall)	Especificidade	F1-Score	Supor te
0	0.7921	0.9556	0.8290	101
1	0.3000	0.9181	0.3857	90
2	0.8222	0.7722	0.6491	90
3	0.9778	0.9893	0.9724	90

[Relatório de Classificação (Scikit-learn)]

	precision	recall	f1-score	support
0	0.87	0.79	0.83	101
1	0.54	0.30	0.39	90
2	0.54	0.82	0.65	90
3	0.97	0.98	0.97	90
accuracy			0.73	371
macro avg	0.73	0.72	0.71	371
weighted avg	0.73	0.73	0.71	371

Matriz de Confusão - VGG16 WithAug - Base de Teste





--- Modelo: ResNet50 NoAug ---

Acurácia Geral de Teste: 0.8868

[Métricas por Classe (Sensibilidade, Especificidade, F1-Score)]

Sensibilidade (Recall) Especificidade F1-Score Suporte

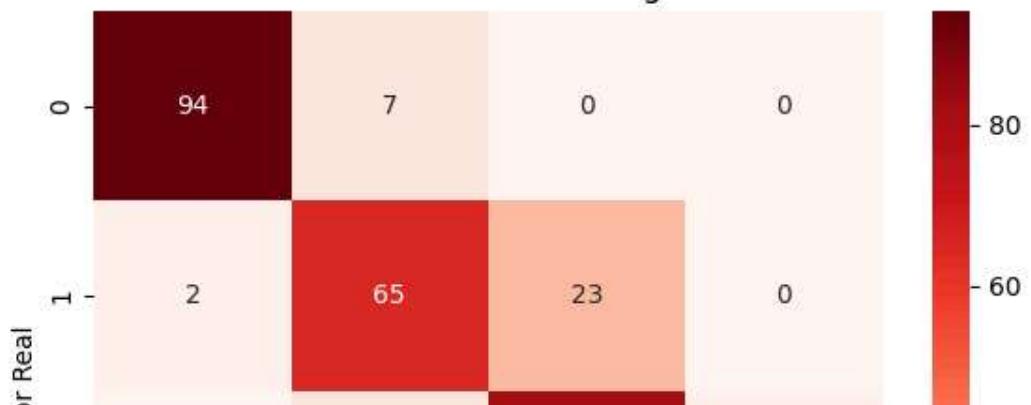
Classe				
0	0.9307	0.9926	0.9543	101
1	0.7222	0.9466	0.7647	90
2	0.9000	0.9146	0.8308	90
3	0.9889	0.9964	0.9889	90

[Relatório de Classificação (Scikit-learn)]

precision recall f1-score support

0	0.98	0.93	0.95	101
1	0.81	0.72	0.76	90
2	0.77	0.90	0.83	90
3	0.99	0.99	0.99	90
accuracy			0.89	371
macro avg	0.89	0.89	0.88	371
weighted avg	0.89	0.89	0.89	371

Matriz de Confusão - ResNet50 NoAug - Base de Teste





--- Modelo: ResNet50 WithAug ---

Acurácia Geral de Teste: 0.9137

[Métricas por Classe (Sensibilidade, Especificidade, F1-Score)]

Sensibilidade (Recall) Especificidade F1-Score Suporte

Classe				
0	0.8911	0.9963	0.9375	101
1	0.8333	0.9431	0.8287	90
2	0.9444	0.9466	0.8947	90

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277</p