1 - Extração de Caracteristicas

```
import numpy as np
import matplotlib.pyplot as plt
import zipfile
from google.colab.patches import cv2_imshow
import os
import cv2 as cv
from skimage.feature import local_binary_pattern
import pandas as pd
```

```
!wget /content/Train_Warwick.zip
!wget /content/Test_Warwick.zip

# Extrai o conteúdo do arquivo zip
!unzip Train_Warwick.zip
!unzip Test_Warwick.zip
```

Mostrar saída oculta

Carregar base de dados

```
# prompt: carregue a pasta Test_4cl_amostra em uma base de dados de imagens
# ## Carregar base de dados


def load_images_from_folder(folder):
  images = []
  for filename in os.listdir(folder):
    img = cv.imread(os.path.join(folder,filename))
    if img is not None:
      images.append((img, filename, i))
  return images

train_images = []

for i in range(4):
  train_images += load_images_from_folder('Train_4cls_amostra/' + str(i) + '/')

#train_images

```

```
''' separando em treino de validação, separando as imagens de 1 paciente por
categoria (aprox. 30 imagens por paciente) temos ~20% da base de treino
Pacientes escolhidos: 01, 14, 04 e 06'''

dados_treino = []
dados_val = []

for image, filename, cat in train_images:
  if filename.startswith('01'):
    dados_val.append((image, cat))
  elif filename.startswith('14'):
    dados_val.append((image, cat))
  elif filename.startswith('04'):
    dados_val.append((image, cat))
  elif filename.startswith('06'):
    dados_val.append((image, cat))
  else:
    dados_treino.append((image, cat))

len(dados_treino), len(dados_val)
```

```
(477, 116)
```

```
# exibindo as imagens com o nome do arquivo e categoria

import matplotlib.pyplot as plt

# Assuming your tuple structure is (image, filename, label)
for image, filename, cat in train_images:
    # Display the image
    plt.imshow(cv.cvtColor(image, cv.COLOR_BGR2RGB))  # Convert to RGB for Matplotlib
    plt.title(f"Filename: {filename}, Label: {cat}")
```

```
    plt.show()

    # Print other information
    print(f"Image shape: {image.shape}")
    print(f"Filename: {filename}")
    print(f"Label: {cat}")
    print("-" * 20)  # Separator
```

Mostrar saída oculta

```
#Extrai características utilizando LBP

def extract_lbp_features(image):
    gray_image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    radius = 3  # Adjust as needed
    n_points = 8 * radius  # Adjust as needed
    lbp = local_binary_pattern(gray_image, n_points, radius, method='uniform')
    # Calculate histogram to represent the LBP features
    n_bins = int(lbp.max() + 1)
    hist, _ = np.histogram(lbp, density=True, bins=n_bins, range=(0, n_bins))
    return hist

# Extrai caracteristica e salva em lista junto com a categoria
data_treino = []
data_val = []

#Base treino
for image, cat in dados_treino:
    features_lbp = extract_lbp_features(image)
    data_treino.append(np.concatenate(([cat], features_lbp)))

#Base validação
for image, cat in dados_val:
    features_lbp_val = extract_lbp_features(image)
    data_val.append(np.concatenate(([cat], features_lbp_val)))


#cria um dataframe pandas e salva em csv
#base treino
colunas = ['cat'] + [f'lbp_{i}' for i in range(len(features_lbp))]
df_treino_lbp = pd.DataFrame(data_treino, columns=colunas)

df_treino_lbp.to_csv('lbp_features.csv', index=False)

#base validação
colunas = ['cat'] + [f'lbp_{i}' for i in range(len(features_lbp_val))]
df_val_lbp = pd.DataFrame(data_val, columns=colunas)

df_val_lbp.to_csv('lbp_features_val.csv', index=False)
```

```
#Extrai características utilizando CNN VGG16

from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.models import Model


# Carrega o modelo VGG16 pré-treinado (sem a camada de classificação)
base_model = VGG16(weights='imagenet', include_top=True)

# Cria um novo modelo que gera as características da camada 'fc2'
model = Model(inputs=base_model.input, outputs=base_model.get_layer('fc2').output)

def extract_features(img):
    # Redimensiona a imagem para o tamanho de entrada do VGG16 (224x224)
    img = cv.resize(img, (224, 224))
    # Converte a imagem para um array NumPy
    #x = image.img_to_array(img)
    # Expande as dimensões para criar um batch de uma única imagem
    x = np.expand_dims(img, axis=0)
    # Pré-processa a imagem de acordo com o VGG16
    x = preprocess_input(x)
    # Extrai as características usando o modelo
    features = model.predict(x)
    # Retorna as características como um array NumPy
    return features[0]

features_list = []
features_list_val = []

```

```
# BASE DE TREINO
for image, cat in dados_treino:
    features_cnn = extract_features(image)
    features_list.append(np.concatenate(([cat], features_cnn)))

# BASE DE VALIDAÇÃO
for image, cat in dados_val:
    features_cnn_val = extract_features(image)
    features_list_val.append(np.concatenate(([cat], features_cnn_val)))

```

Mostrar saída oculta

```
# Create DataFrame and save to CSV
#BASE TREINO
num_features = features_cnn.shape[0] # Get the number of features from features_cnn
colunas_cnn = ['cat'] + [f'cnn_{i}' for i in range(num_features)]
df_treino_cnn = pd.DataFrame(features_list, columns=colunas_cnn)

df_treino_cnn.to_csv('cnn_features.csv', index=False)

# BASE VALIDAÇÃO
num_features = features_cnn_val.shape[0] # Get the number of features from features_cnn
colunas_cnn = ['cat'] + [f'cnn_{i}' for i in range(num_features)]
df_val_cnn = pd.DataFrame(features_list_val, columns=colunas_cnn)

df_val_cnn.to_csv('cnn_features_val.csv', index=False)
```

TREINAMENTO DOS MODELOS

As metricas calculadas nesta etapa foram usadas para o ajuste dos hiperparametros e não serão necessariamente usadas para a escolha do melhor modelo.

```
#função para avaliação de resultados

def avaliar_resultados(y_real, y_pred):
    # Avaliando a acurácia
    accuracy = accuracy_score(y_real, y_pred)
    print(f"Acurácia do modelo: {accuracy}")

    # Exibindo a matriz de confusão
    cm = confusion_matrix(y_real, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot(cmap=plt.cm.Blues)
    plt.show()

    # Calculando as métricas por classe e a média macro
    report = classification_report(y_real, y_pred, output_dict=True)
    sensibilidade_macro = report['macro avg']['recall']  # Sensibilidade (Recall) macro
    especificidade_macro = report['macro avg']['precision']  # Especificidade (Precision) macro
    f1_score_macro = report['macro avg']['f1-score']  # F1-Score macro

    print(f"Sensibilidade (Macro): {sensibilidade_macro}")
    print(f"Especificidade (Macro): {especificidade_macro}")
    print(f"F1-Score (Macro): {f1_score_macro}")

    return {
        "acuracia": accuracy,
        "sensibilidade_macro": sensibilidade_macro,
        "especificidade_macro": especificidade_macro,
        "f1_score_macro": f1_score_macro
    }
```

RANDOM FOREST

LBP

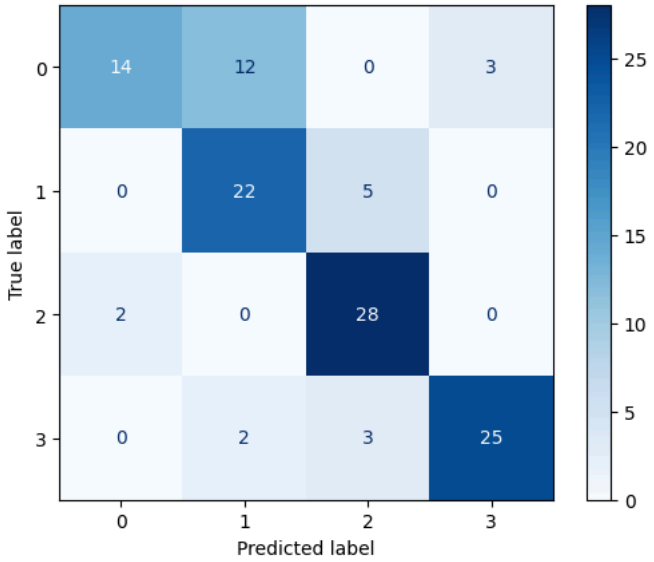
```
#LBP

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report

#df_treino_lbp = pd.read_csv('lbp_features.csv')

```

```
# Separate features (X) and target (y)
X_train = df_treino_lbp.drop('cat', axis=1)  # Caracteristicas
y_train = df_treino_lbp['cat']     # alvo

# criando e treinando o modelo
rf_model_lbp = RandomForestClassifier(n_estimators=300,
                                      max_depth=15,
                                      min_samples_split=8,
                                      min_samples_leaf=5,
                                      max_features='sqrt')
#rf_model = RandomForestClassifier()

rf_model_lbp.fit(X_train, y_train)


```

```
RandomForestClassifier(max_depth=15, min_samples_leaf=5, min_samples_split=8,
                       n_estimators=300)
```

```
# Avaliando o modelo para o tunning dos hiperparametros
y_pred = rf_model_lbp.predict(df_val_lbp.drop('cat', axis=1))
resultado_tunning_rf_lbp = avaliar_resultados(df_val_lbp['cat'], y_pred)

```

```
Acurácia do modelo: 0.7672413793103449
```




```
Sensibilidade (Macro): 0.7660600255427842
Especificidade (Macro): 0.7891865079365079
F1-Score (Macro): 0.7577971836592525
```

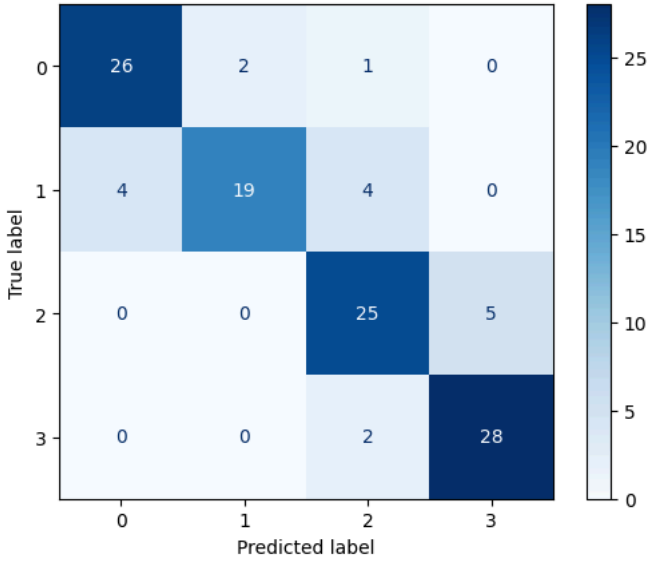
CNN

```
# CNN
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

#df_treino_cnn = pd.read_csv('cnn_features.csv')

# Separate features (X) and target (y)
X_train = df_treino_cnn.drop('cat', axis=1)  # Caracteristicas
y_train = df_treino_cnn['cat']     # alvo

# instanciando com hiperparametros
rf_model_cnn = RandomForestClassifier(n_estimators=250,
                                      max_depth=12,
                                      min_samples_split=10,
                                      min_samples_leaf=5,
                                      max_features='sqrt')
#rf_model = RandomForestClassifier()

#trainando o modelo
rf_model_cnn.fit(X_train, y_train)

```

```
RandomForestClassifier(max_depth=12, min_samples_leaf=5, min_samples_split=10,
                       n_estimators=250)
```

```
# Avaliando o modelo para o tunning dos hiperparametros
y_pred = rf_model_cnn.predict(df_val_cnn.drop('cat', axis=1))
resultado_tunning_rf_cnn = avaliar_resultados(df_val_cnn['cat'], y_pred)
```

```
Acurácia do modelo: 0.8448275862068966
```




```
Sensibilidade (Macro): 0.8417305236270753
Especificidade (Macro): 0.850290854978355
F1-Score (Macro): 0.8420907751655428
```

SVM

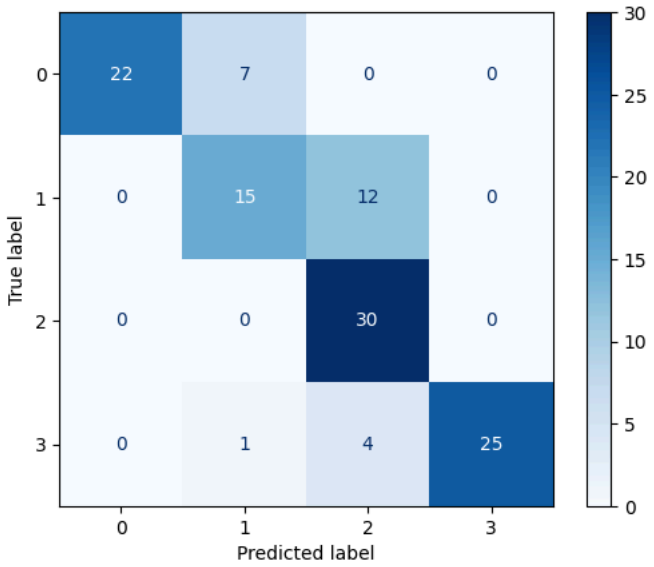
LBP

```
# LBP
from sklearn.svm import SVC

# Separate features (X) and target (y)
X_train = df_treino_lbp.drop('cat', axis=1)  # Caracteristicas
y_train = df_treino_lbp['cat']     # alvo

#instanciando com hiperparametros
svm_model_lbp = SVC(kernel='rbf', C=100, gamma = 5)


#treinando o modelo
svm_model_lbp.fit(X_train, y_train)


```

```
SVC(C=100, gamma=5)
```

```
# Avaliando o modelo para o tunning dos hiperparametros
y_pred = svm_model_lbp.predict(df_val_lbp.drop('cat', axis=1))
resultado_tunning_svm_lbp = avaliar_resultados(df_val_lbp['cat'], y_pred)
```

```
Acurácia do modelo: 0.7931034482758621
```




```
Sensibilidade (Macro): 0.7868773946360154
Especificidade (Macro): 0.8260869565217391
F1-Score (Macro): 0.7903274228351629
```

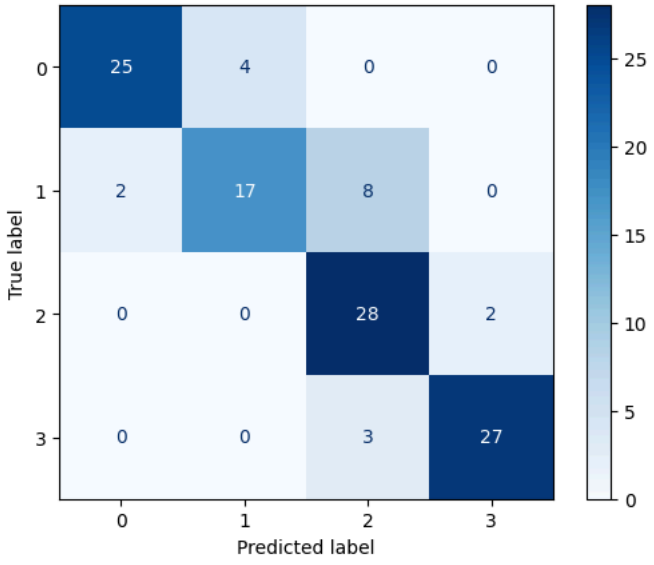
CNN

```
#CNN
# Separate features (X) and target (y)
X_train = df_treino_cnn.drop('cat', axis=1)  # Caracteristicas
y_train = df_treino_cnn['cat']     # alvo

#instanciando com hiperparametros
svm_model_cnn = SVC(kernel='linear', C=1)

#treinando o modelo
svm_model_cnn.fit(X_train, y_train)
```

```
SVC
SVC(C=1, kernel='linear')
```

```
# Avaliando o modelo para o tunning dos hiperparametros
y_pred = svm_model_cnn.predict(df_val_cnn.drop('cat', axis=1))
resultado_tunning_svm_cnn = avaliar_resultados(df_val_cnn['cat'], y_pred)
```

```
Acurácia do modelo: 0.8362068965517241
```




```
Sensibilidade (Macro): 0.8312579821200511
Especificidade (Macro): 0.8461082340392685
F1-Score (Macro): 0.8320097290942907
```

RNA

LBP

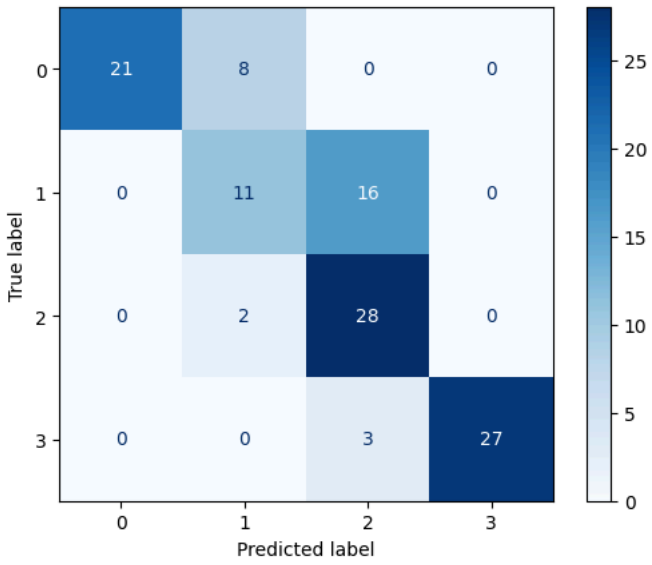
```
#LBP
from sklearn.neural_network import MLPClassifier

# Separate features (X) and target (y) - treinamento
X_train = df_treino_lbp.drop('cat', axis=1)  # Caracteristicas
y_train = df_treino_lbp['cat']     # alvo

#instanciando com hiperparametros
rna_model_lbp = MLPClassifier(hidden_layer_sizes=(100,),
                          activation='relu',
                          solver='adam',
                          alpha=0.0001,
                          batch_size='auto',
                          max_iter=1000)

#treinando modelo
rna_model_lbp.fit(X_train, y_train)

```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:690: ConvergenceWarning: Stochastic Optimiz
  warnings.warn(
```

```
MLPClassifier(max_iter=1000)
```

```
# Avaliando o modelo para o tunning dos hiperparametros
y_pred = rna_model_lbp.predict(df_val_lbp.drop('cat', axis=1))
resultado_tunning_rna_lbp = avaliar_resultados(df_val_lbp['cat'], y_pred)
```

```
Acurácia do modelo: 0.75
```




```
Sensibilidade (Macro): 0.7412196679438058
Especificidade (Macro): 0.7798885511651469
F1-Score (Macro): 0.743243620414673
```

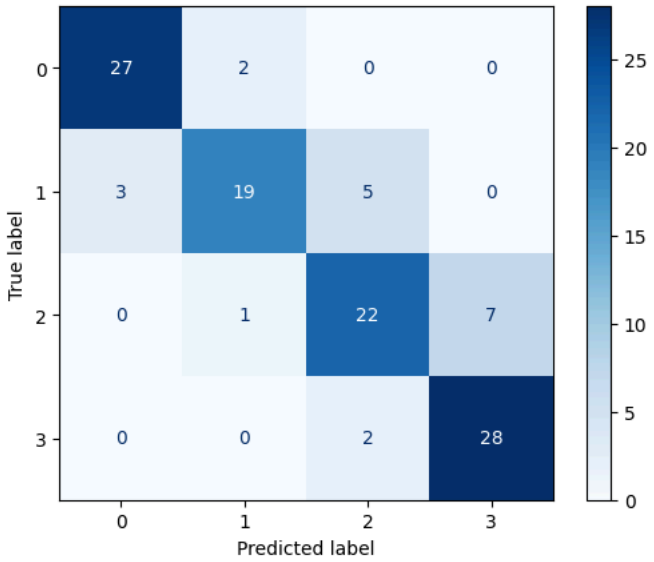
CNN

```
# CNN
# Separate features (X) and target (y) - treinamento
X_train = df_treino_cnn.drop('cat', axis=1)  # Caracteristicas
y_train = df_treino_cnn['cat']     # alvo

#instanciando com hiperparametros
rna_model_cnn = MLPClassifier(hidden_layer_sizes=(100,),
                          activation='relu',
                          solver='adam',
                          alpha=0.0001,
                          batch_size='auto',
                          max_iter=1000)
```

```
13
14 #treinando modelo
15 rna_model_cnn.fit(X_train, y_train)
16
```

```
1 # Avaliando o modelo para o tunning dos hiperparametros
2 y_pred = rna_model_cnn.predict(df_val_cnn.drop('cat', axis=1))
3 resultado_tunning_rna_cnn = avaliar_resultados(df_val_cnn['cat'], y_pred)
```

```
Acurácia do modelo: 0.8275862068965517
```




```
Sensibilidade (Macro): 0.8253512132822478
Especificidade (Macro): 0.830564263322884
F1-Score (Macro): 0.8245164036931592
```

APLICAÇÃO NA BASE DE TESTE

```
1 # Carregamento da base de testes
2 test_images = []
3
4 for i in range(4):
5   test_images += load_images_from_folder('Test_4cl_amostra/' + str(i) + '/')
6
7 #test_images
```

```
1 #Extrai características utilizando LBP e salva em lista junto com a categoria
2 data_test = []
3
4 for image, filename, cat in test_images:
5     features_lbp_test = extract_lbp_features(image)
6     data_test.append(np.concatenate(([cat], features_lbp_test)))
7
8 #cria um dataframe pandas e salva em csv
9 colunas = ['cat'] + [f'lbp_{i}' for i in range(len(features_lbp_test))]
10 df_test_lbp = pd.DataFrame(data_test, columns=colunas)
11 df_test_lbp.to_csv('lbp_features_test.csv', index=False)
```

```
1 #Extrai características utilizando CNN e salva em lista junto com a categoria
2 features_list_test = []
3
4 for image, filename, cat in test_images:
5     features_cnn_test = extract_features(image)
6     features_list_test.append(np.concatenate(([cat], features_cnn_test)))
7
8 # Create DataFrame and save to CSV
9 num_features = features_cnn.shape[0] # Get the number of features from features_cnn
10 colunas_cnn = ['cat'] + [f'cnn_{i}' for i in range(num_features)]
11 df_test_cnn = pd.DataFrame(features_list_test, columns=colunas_cnn)
12
13 df_test_cnn.to_csv('cnn_features_test.csv', index=False)
```

Mostrar saída oculta

```
1
2 #faz as predições nas bases de teste
3 y_pred_rf_lbp = rf_model_lbp.predict(df_test_lbp.drop('cat', axis=1))
```

```
y_pred_rf_cnn = rf_model_cnn.predict(df_test_cnn.drop('cat', axis=1))

y_pred_svm_lbp = svm_model_lbp.predict(df_test_lbp.drop('cat', axis=1))
y_pred_svm_cnn = svm_model_cnn.predict(df_test_cnn.drop('cat', axis=1))

y_pred_rna_lbp = rna_model_lbp.predict(df_test_lbp.drop('cat', axis=1))
y_pred_rna_cnn = rna_model_cnn.predict(df_test_cnn.drop('cat', axis=1))

# avaliação dos resultados
print("Random Forest - LBP")
resultados_rf_lbp = avaliar_resultados(df_test_lbp['cat'], y_pred_rf_lbp)
print("\nRandom Forest - CNN")
resultados_rf_cnn = avaliar_resultados(df_test_cnn['cat'], y_pred_rf_cnn)

print("\nSVM - LBP")
resultados_svm_lbp = avaliar_resultados(df_test_lbp['cat'], y_pred_svm_lbp)
print("\nSVM - CNN")
resultados_svm_cnn = avaliar_resultados(df_test_cnn['cat'], y_pred_svm_cnn)

print("\nRNA - LBP")
resultados_rna_lbp = avaliar_resultados(df_test_lbp['cat'], y_pred_rna_lbp)
print("\nRNA - CNN")
resultados_rna_cnn = avaliar_resultados(df_test_cnn['cat'], y_pred_rna_cnn)
```

2 - Redes Neurais

```
##Mudar para essa versão do Tensorflow
%pip install tensorflow==2.15
```

Mostrar saída oculta

```
from tensorflow.keras.applications import VGG16, ResNet50
from tensorflow.keras import layers, models
from keras.layers import Dense, Dropout, Flatten
from keras.models import Model
import cv2 as cv
import zipfile
import os



# Módulo para imprimir os gráficos de treinamento de forma dinâmica.
!pip install livelossplot
```

Mostrar saída oculta

```
!wget /content/Train_Warwick.zip
!wget /content/Test_Warwick.zip

# Extrai o conteúdo do arquivo zip
!unzip Train_Warwick.zip
!unzip Test_Warwick.zip
```

Mostrar saída oculta

```
def load_images_from_folder(folder):
  '''Carrega as imagens dos diretorios em uma lista com o nome do arquivo e classe'''
  images = []
  for filename in os.listdir(folder):
    img = cv.imread(os.path.join(folder,filename))
    if img is not None:
      images.append((img, filename, i))
  return images

def resize_images(images):
    '''Redimensiona as imagens para 224x224 como esperado por VGG16, ResNet50'''
    resized_images = []
    for image, filename, i in images:
      resized_image = cv.resize(image, (224, 224))
      resized_images.append((resized_image, filename, i))
    return resized_images
```

```
# carregando as imagens em 2 listas
# Move os dados do diretório images para images/train ou images/test:
import shutil
from collections import defaultdict
import json
from pathlib import Path
import os

os.makedirs('images/train', exist_ok=True)
os.makedirs('images/test', exist_ok=True)

temp_train_images = [[],[],[],[]]
test_images = [[],[],[],[]]

for i in range(4):
  temp_train_images[i].append(load_images_from_folder('Train_4cls_amostra/' + str(i) + '/'))

for i in range(4):
  test_images[i].append(load_images_from_folder('Test_4cl_amostra/' + str(i) + '/'))

  ''' separando em treino de validação, separando as imagens de 1 paciente por
categoria (aprox. 30 imagens por paciente) temos ~20% da base de treino
Pacientes escolhidos: 01, 14, 04 e 06'''


for i in range(4):
  for image in temp_train_images[i]:
    image = resize_images(image)
```

```
    for img, filename, cat in image:
      os.makedirs('images/test/' + str(i) + '/', exist_ok=True)
      os.makedirs('images/train/'+ str(i) + '/', exist_ok=True)
      if filename.startswith('01'):
        cv.imwrite(os.path.join('images/test/' + str(i) + '/', filename), img)
      elif filename.startswith('14'):
        cv.imwrite(os.path.join('images/test/' + str(i) + '/', filename), img)
      elif filename.startswith('04'):
        cv.imwrite(os.path.join('images/test/' + str(i) + '/', filename), img)
      elif filename.startswith('06'):
        cv.imwrite(os.path.join('images/test/' + str(i) + '/', filename), img)
      else:
        cv.imwrite(os.path.join('images/train/' + str(i) + '/', filename), img)

```

```
import os

class_subset = sorted(os.listdir('Train_4cls_amostra/'))
print(class_subset)
```

```
['0', '1', '2', '3']
```

```
# carregando as redes com transfer learning sem as camadas fuly connected

base_model_vgg16 = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model_resnet50 = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

for layer in base_model_vgg16.layers:
  layer.trainable = False

for layer in base_model_resnet50.layers:
  layer.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_no
58889256/58889256 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernel
94765736/94765736 [==============================] - 0s 0us/step
```

```
# Adicionar camadas fully connected no VGG16

# camadas próprias - você pode colocar mais se quiser
# A saída da resnet será a entrada da camada criada
x = Flatten()(base_model_vgg16.output)

# camada de classificação com as 4 classes utilizadas
prediction = Dense(4, activation='softmax')(x)

# Criação do Objeto Modelo (a parte da resnet + as camadas Fully connected criadas)
model_vgg16 = Model(inputs=base_model_vgg16.input, outputs=prediction)

```

```
# Adicionar camadas fully connected no resnet50

# camadas próprias - você pode colocar mais se quiser
# A saída da resnet será a entrada da camada criada
x = Flatten()(base_model_resnet50.output)

# camada de classificação com as 4 classes utilizadas
prediction = Dense(len(class_subset), activation='softmax')(x)

# Criação do Objeto Modelo (a parte da resnet + as camadas Fully connected criadas)
model_resnet50 = Model(inputs=base_model_resnet50.input, outputs=prediction)
```

Data Augmentation

```
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_generator = ImageDataGenerator(
                                     rotation_range=90,
                                     brightness_range=[0.1, 0.7],
                                     width_shift_range=0.5,
                                     height_shift_range=0.5,
                                     horizontal_flip=True,
                                     vertical_flip=True,
                                     validation_split=0.2,
                                     preprocessing_function=preprocess_input)
```

```

test_generator = ImageDataGenerator(preprocessing_function=preprocess_input)
```

```
BATCH_SIZE = 64 # quantidade de imagens criadas em cada ciclo

traingen = train_generator.flow_from_directory('images/train',
                                               target_size=(224, 224),
                                               batch_size=BATCH_SIZE,
                                               class_mode='categorical',
                                               classes=class_subset,
                                               subset='training',
                                               shuffle=True,
                                               seed=42)

validgen = train_generator.flow_from_directory('images/train',
                                               target_size=(224, 224),
                                               batch_size=BATCH_SIZE,
                                               class_mode='categorical',
                                               classes=class_subset,
                                               subset='validation',
                                               shuffle=True,
                                               seed=42)
```

```
Found 382 images belonging to 4 classes.
Found 95 images belonging to 4 classes.
```

```
testgen = test_generator.flow_from_directory('images/test',
                                             target_size=(224, 224),
                                             batch_size=BATCH_SIZE,
                                             class_mode=None,
                                             classes=class_subset,
                                             shuffle=False,
                                             seed=42)
```

```
Found 116 images belonging to 4 classes.
```

```
import tensorflow as tf
train_noaug = tf.keras.utils.image_dataset_from_directory('images/train',
                                          labels='inferred',
                                          label_mode='categorical',
                                          subset="training",
                                          class_names=class_subset,
                                          validation_split = 0.2,
                                          image_size=(224, 224),
                                          batch_size=BATCH_SIZE,
                                          shuffle=True,
                                          seed=42)

validation_noaug = tf.keras.utils.image_dataset_from_directory('images/train',
                                          labels='inferred',
                                          label_mode='categorical',
                                          subset="validation",
                                          class_names=class_subset,
                                          validation_split = 0.2,
                                          image_size=(224, 224),
                                          batch_size=BATCH_SIZE,
                                          shuffle=True,
                                          seed=42)
```

```
Found 477 files belonging to 4 classes.
Using 382 files for training.
Found 477 files belonging to 4 classes.
Using 95 files for validation.
```

```
test_noaug = tf.keras.utils.image_dataset_from_directory('images/test',
                                          labels='inferred',
                                          label_mode='categorical',
                                          class_names=class_subset,
                                          image_size=(224, 224),
                                          batch_size=BATCH_SIZE,
                                          shuffle=False,
                                          seed=42)
```

```
Found 116 files belonging to 4 classes.
```

Treino da VGG16

```
model_vgg16.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 4)                 100356

=================================================================
Total params: 14815044 (56.51 MB)
Trainable params: 100356 (392.02 KB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

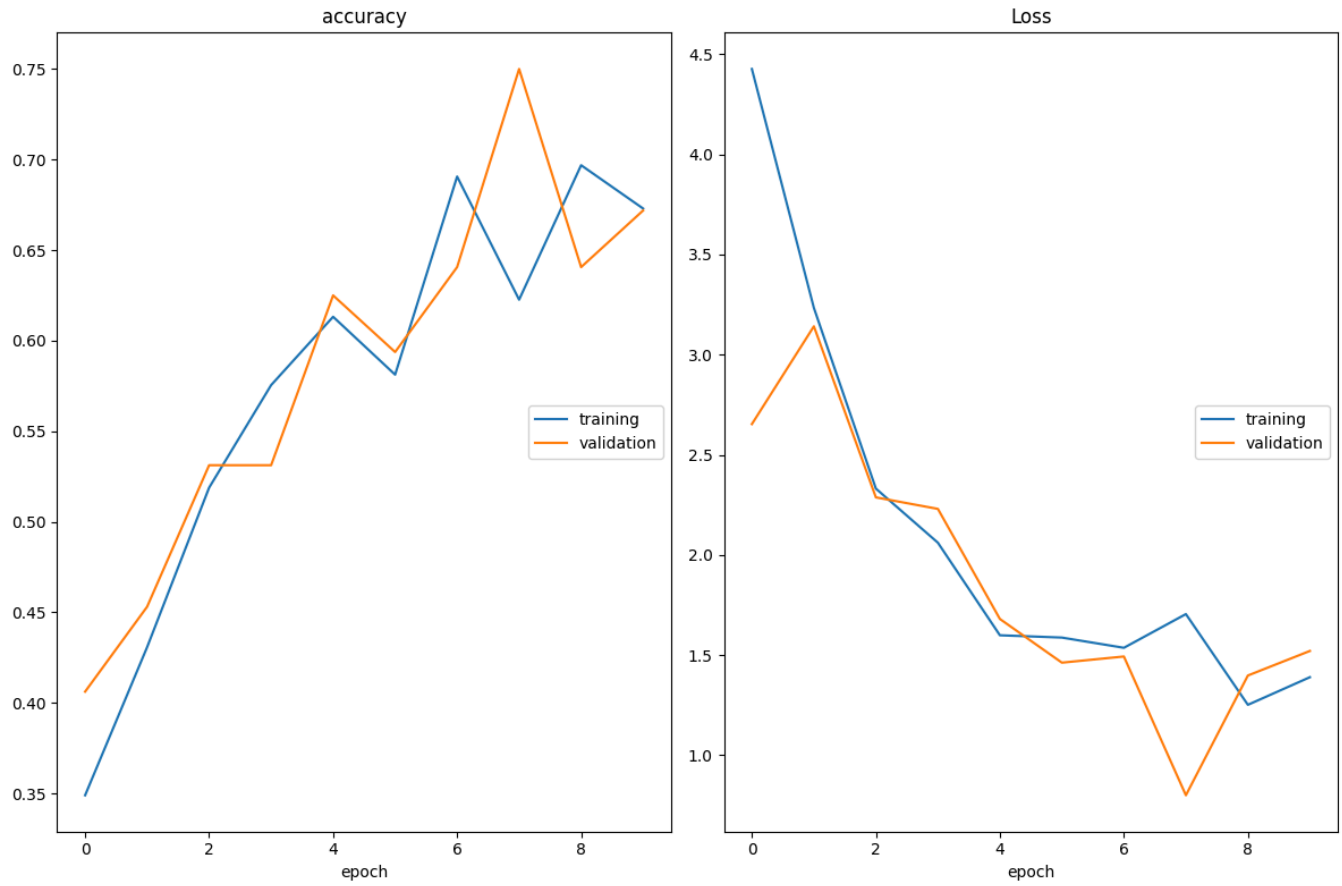
```
model_resnet50.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]        0         []

 conv1_pad (ZeroPadding2D)   (None, 230, 230, 3)          0         ['input_2[0][0]']

 conv1_conv (Conv2D)         (None, 112, 112, 64)         9472      ['conv1_pad[0][0]']

 conv1_bn (BatchNormalizati  (None, 112, 112, 64)         256       ['conv1_conv[0][0]']
 on)

 conv1_relu (Activation)     (None, 112, 112, 64)         0         ['conv1_bn[0][0]']

 pool1_pad (ZeroPadding2D)   (None, 114, 114, 64)         0         ['conv1_relu[0][0]']

 pool1_pool (MaxPooling2D)   (None, 56, 56, 64)           0         ['pool1_pad[0][0]']

 conv2_block1_1_conv (Conv2  (None, 56, 56, 64)           4160      ['pool1_pool[0][0]']
 D)

 conv2_block1_1_bn (BatchNo  (None, 56, 56, 64)           256       ['conv2_block1_1_conv[0][0]']
 rmalization)

 conv2_block1_1_relu (Activ  (None, 56, 56, 64)           0         ['conv2_block1_1_bn[0][0]']
 ation)

 conv2_block1_2_conv (Conv2  (None, 56, 56, 64)           36928     ['conv2_block1_1_relu[0][0]']
 D)

 conv2_block1_2_bn (BatchNo  (None, 56, 56, 64)           256       ['conv2_block1_2_conv[0][0]']
 rmalization)
```

```
conv2_block1_2_relu (Activ  (None, 56, 56, 64)          0         ['conv2_block1_2_bn[0][0]']
ation)

conv2_block1_0_conv (Conv2  (None, 56, 56, 256)         16640     ['pool1_pool[0][0]']
D)

conv2_block1_3_conv (Conv2  (None, 56, 56, 256)         16640     ['conv2_block1_2_relu[0][0]']
D)

conv2_block1_0_bn (BatchNo  (None, 56, 56, 256)         1024      ['conv2_block1_0_conv[0][0]']
rmalization)

conv2_block1_3_bn (BatchNo  (None, 56, 56, 256)         1024      ['conv2_block1_3_conv[0][0]']
rmalization)

conv2_block1_add (Add)      (None, 56, 56, 256)         0         ['conv2_block1_0_bn[0][0]',
                                                                   'conv2_block1_3_bn[0][0]']

conv2_block1_out (Activati  (None, 56, 56, 256)         0         ['conv2_block1_add[0][0]']
on)

conv2_block2_1_conv (Conv2  (None, 56, 56, 64)          16448     ['conv2_block1_out[0][0]']
D)
```

Treinamento VGG16 com Data Augmentation

```
%%time
from keras.optimizers import RMSprop
from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard
from livelossplot import PlotLossesKeras

steps_per_epoch = traingen.samples // BATCH_SIZE
val_steps = validgen.samples // BATCH_SIZE

n_epochs = 10

optimizer = RMSprop(learning_rate=0.0001)

model_vgg16.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])

# Salva o modelo Keras após cada época, porém só o de melhor resultado
checkpointer = ModelCheckpoint(filepath='img_model_vgg16.weights.best.keras',
                               verbose=1,
                               save_best_only=True)

# Para o treinamento para prevenir o overfitting
# Não utilizei aqui, pois queria que rodasse todas as 30 épocas
early_stop = EarlyStopping(monitor='val_loss',
                           patience=10,
                           restore_best_weights=True,
                           mode='min')

# Treinamento do Modelo
history_tl = model_vgg16.fit(traingen,
                    epochs=n_epochs,
                    steps_per_epoch=steps_per_epoch,
                    validation_data=validgen,
                    validation_steps=val_steps,
                    callbacks=[checkpointer, PlotLossesKeras()],
                    #callbacks=[early_stop, checkpointer, PlotLossesKeras()],
                    verbose=False)
```



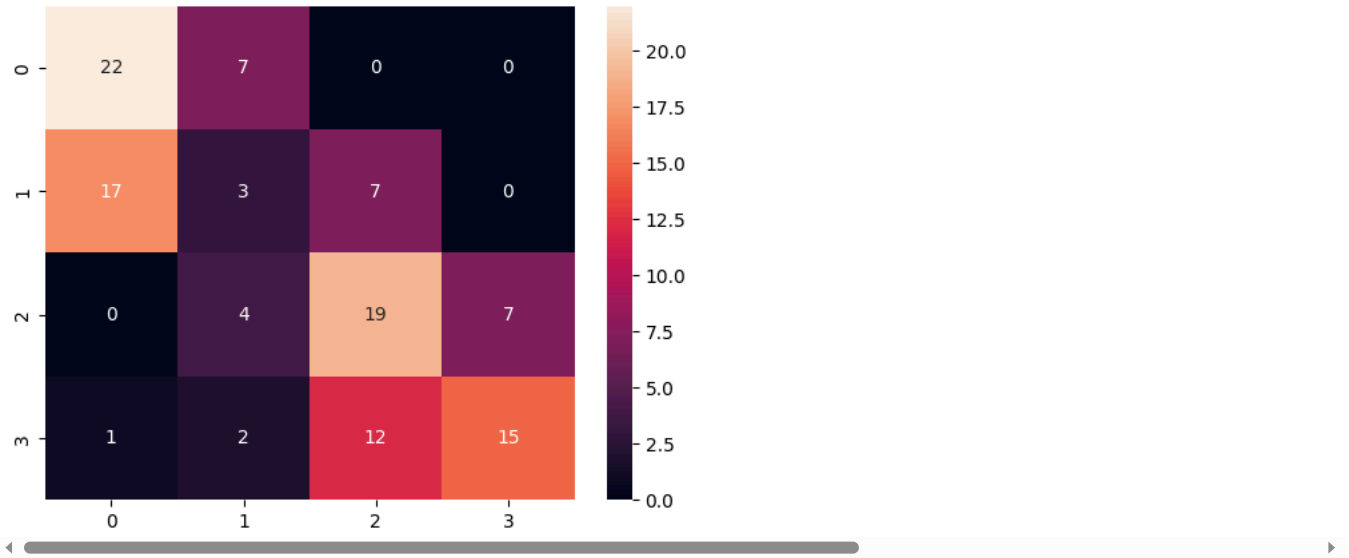

```
accuracy
        training                (min:    0.349, max:    0.697, cur:    0.673)
        validation              (min:    0.406, max:    0.750, cur:    0.672)
Loss
        training                (min:    1.252, max:    4.426, cur:    1.390)
        validation              (min:    0.800, max:    3.141, cur:    1.521)
```

Aplicar Modelo treinado na base de teste

Calcular Métrica de Sensibilidade,, Especificidade e F1-Score com base na matriz de confusão

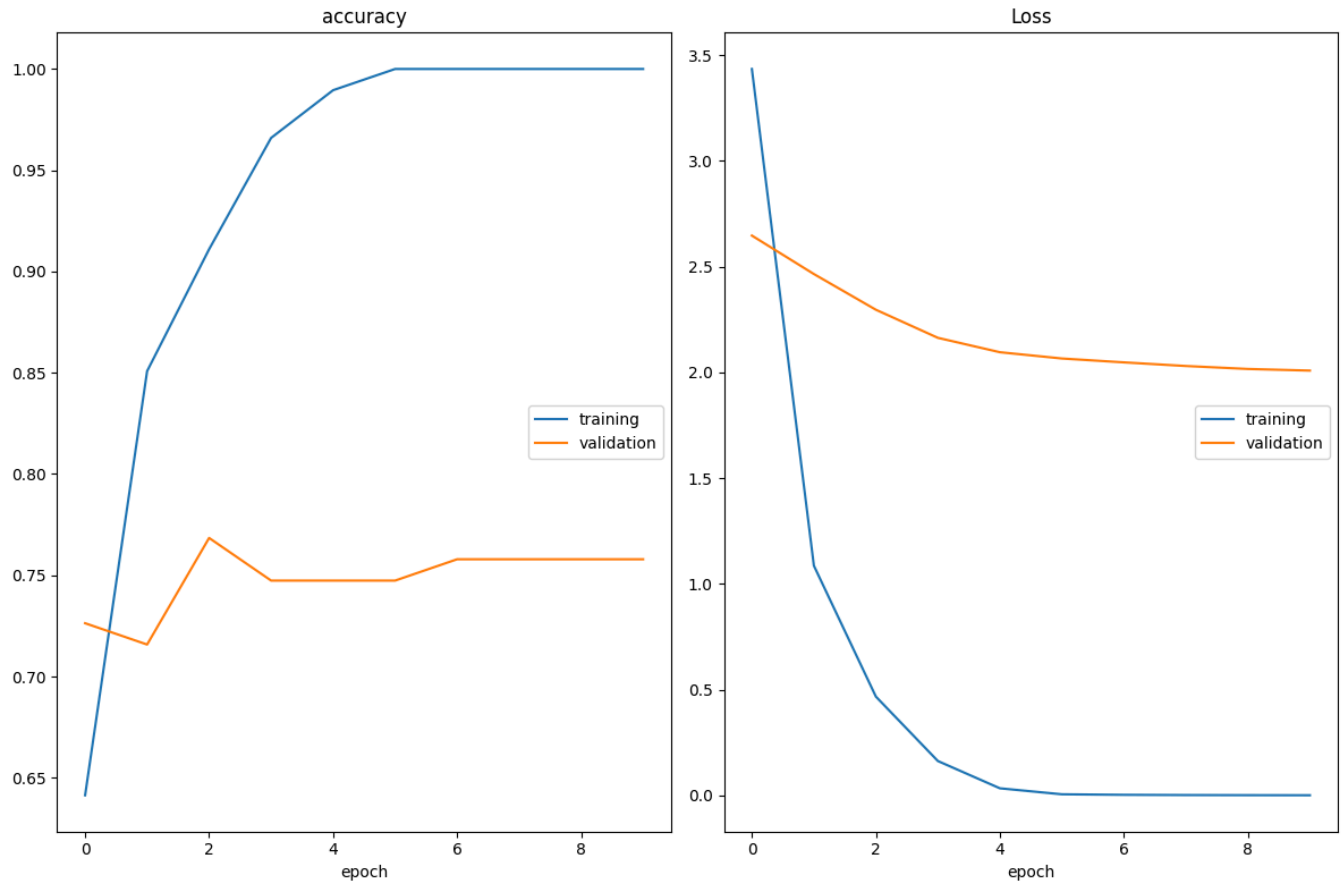
```
##Aplicar Modelo treinado na base de teste
### Calcular Métrica de Sensibilidade,, Especificidade e F1-Score com base na matriz de confusão
from sklearn.metrics import accuracy_score
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report, confusion_matrix

model_vgg16.load_weights('img_model_vgg16.weights.best.keras') ## Inicializa o peso com o melhor treino

true_classes = testgen.classes
class_indices = traingen.class_indices
class_indices = dict((v,k) for k,v in class_indices.items())

predictions = model_vgg16.predict(testgen)
predicted_classes = np.argmax(predictions, axis=1)


accuracy = accuracy_score(true_classes, predicted_classes)
precision = precision_score(true_classes, predicted_classes, average='weighted')
f1 = f1_score(true_classes, predicted_classes, average='weighted')
print("Acurácia Modelo VGG16 com Data Augmentation treinado {:.2f}%".format(accuracy * 100))
print("Precisão Modelo VGG16 com Data Augmentation treinado {:.2f}%".format(precision * 100))
print("F1_Score Modelo VGG16 com Data Augmentation treinado {:.2f}%".format(f1 * 100))

# Get the names of the ten classes
class_names = testgen.class_indices.keys()

#TODO: Imprimir Matriz de confusão
plot  = confusion_matrix(true_classes, predicted_classes)
sns.heatmap(plot, annot=True, fmt='d', xticklabels=class_names, yticklabels=class_names)
```

```
2/2 [==============================] - 10s 10s/step
Acurácia Modelo VGG16 com Data Augmentation treinado 50.86%
Precisão Modelo VGG16 com Data Augmentation treinado 48.68%
F1_Score Modelo VGG16 com Data Augmentation treinado 48.56%
<Axes: >
```



Treino do modelo com dados sem Data Augmentation

```
%%time
from keras.optimizers import RMSprop
from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard
from livelossplot import PlotLossesKeras

#steps_per_epoch = 32 // BATCH_SIZE
#val_steps = 32 // BATCH_SIZE

n_epochs = 10

optimizer = RMSprop(learning_rate=0.0001)

model_vgg16.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])

# Salva o modelo Keras após cada época, porém só o de melhor resultado
checkpointer = ModelCheckpoint(filepath='img_model_vgg16_noaug.weights.best.keras',
                               verbose=1,
                               save_best_only=True)

# Para o treinamento para prevenir o overfitting
# Não utilizei aqui, pois queria que rodasse todas as 30 épocas
early_stop = EarlyStopping(monitor='val_loss',
                           patience=10,
                           restore_best_weights=True,
                           mode='min')

# Treinamento do Modelo
history_tl = model_vgg16.fit(train_noaug,
                    epochs=n_epochs,
                    #steps_per_epoch=steps_per_epoch,
                    validation_data=validation_noaug,
                    #validation_steps=val_steps,
                    callbacks=[checkpointer, PlotLossesKeras()],
                    #callbacks=[early_stop, checkpointer, PlotLossesKeras()],
                    verbose=False)
```




```
accuracy
        training                (min:    0.641, max:    1.000, cur:    1.000)
        validation              (min:    0.716, max:    0.768, cur:    0.758)
Loss
        training                (min:    0.001, max:    3.434, cur:    0.001)
        validation              (min:    2.008, max:    2.647, cur:    2.008)
```

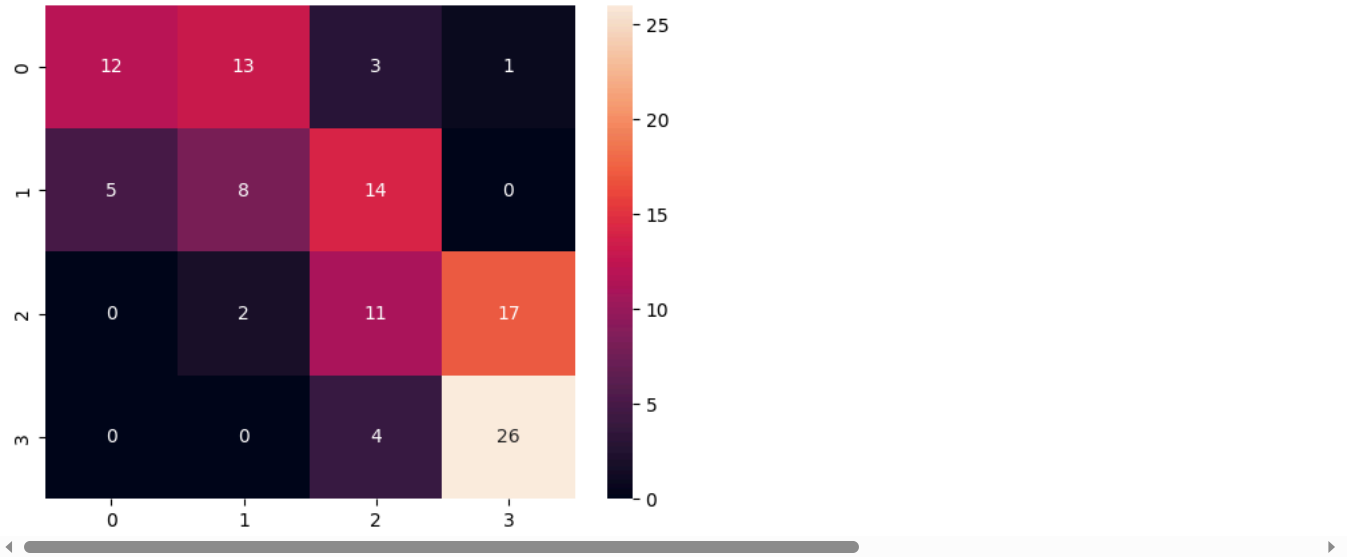
Aplicar Modelo treinado na base de teste

Calcular Métrica de Sensibilidade,, Especificidade e F1-Score com base na matriz de confusão

```
##Aplicar Modelo treinado na base de teste
### Calcular Métrica de Sensibilidade,, Especificidade e F1-Score com base na matriz de confusão

##Aplicar Modelo treinado na base de teste
### Calcular Métrica de Sensibilidade,, Especificidade e F1-Score com base na matriz de confusão
from sklearn.metrics import accuracy_score
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report, confusion_matrix

model_vgg16.load_weights('img_model_vgg16_noaug.weights.best.keras') ## Inicializa o peso com o melhor treino

true_classes = testgen.classes
class_indices = traingen.class_indices
class_indices = dict((v,k) for k,v in class_indices.items())

predictions = model_vgg16.predict(testgen)
predicted_classes = np.argmax(predictions, axis=1)


accuracy = accuracy_score(true_classes, predicted_classes)
precision = precision_score(true_classes, predicted_classes, average='weighted')
f1 = f1_score(true_classes, predicted_classes, average='weighted')
print("Acurácia Modelo VGG16 sem Data Augmentation treinado {:.2f}%".format(accuracy * 100))
print("Precisão Modelo VGG16 sem Data Augmentation treinado {:.2f}%".format(precision * 100))
print("F1_Score Modelo VGG16 sem Data Augmentation treinado {:.2f}%".format(f1 * 100))

# Get the names of the ten classes
class_names = testgen.class_indices.keys()

#TODO: Imprimir Matriz de confusão
```

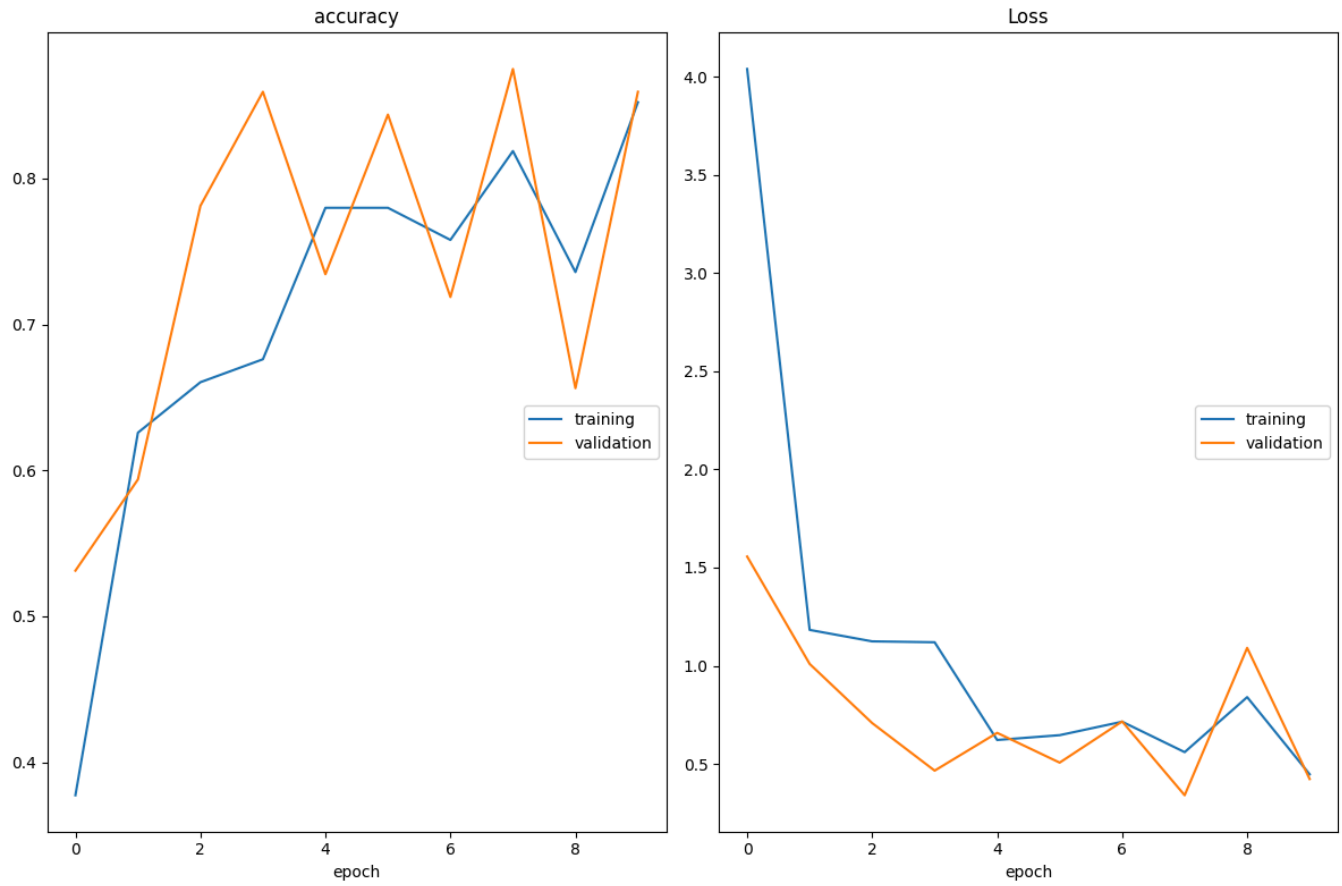
```
plot  = confusion_matrix(true_classes, predicted_classes)
sns.heatmap(plot, annot=True, fmt='d', xticklabels=class_names, yticklabels=class_names)
```

```
2/2 [==============================] - 1s 305ms/step
Acurácia Modelo VGG16 sem Data Augmentation treinado 49.14%
Precisão Modelo VGG16 sem Data Augmentation treinado 49.92%
F1_Score Modelo VGG16 sem Data Augmentation treinado 47.84%
<Axes: >
```



Treino do Resnet50 com Data Augmentation

```
%%time
from keras.optimizers import RMSprop
from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard
from livelossplot import PlotLossesKeras

steps_per_epoch = traingen.samples // BATCH_SIZE
val_steps = validgen.samples // BATCH_SIZE

n_epochs = 10

optimizer = RMSprop(learning_rate=0.0001)

model_resnet50.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])

# Salva o modelo Keras após cada época, porém só o de melhor resultado
checkpointer = ModelCheckpoint(filepath='img_model_resnet50.weights.best.keras',
                               verbose=1,
                               save_best_only=True)

# Para o treinamento para prevenir o overfitting
# Não utilizei aqui, pois queria que rodasse todas as 30 épocas
early_stop = EarlyStopping(monitor='val_loss',
                           patience=10,
                           restore_best_weights=True,
                           mode='min')

# Treinamento do Modelo
history_tl = model_resnet50.fit(traingen,
                    epochs=n_epochs,
                    steps_per_epoch=steps_per_epoch,
                    validation_data=validgen,
                    validation_steps=val_steps,
                    callbacks=[checkpointer, PlotLossesKeras()],
                    #callbacks=[early_stop, checkpointer, PlotLossesKeras()],
                    verbose=False)
```



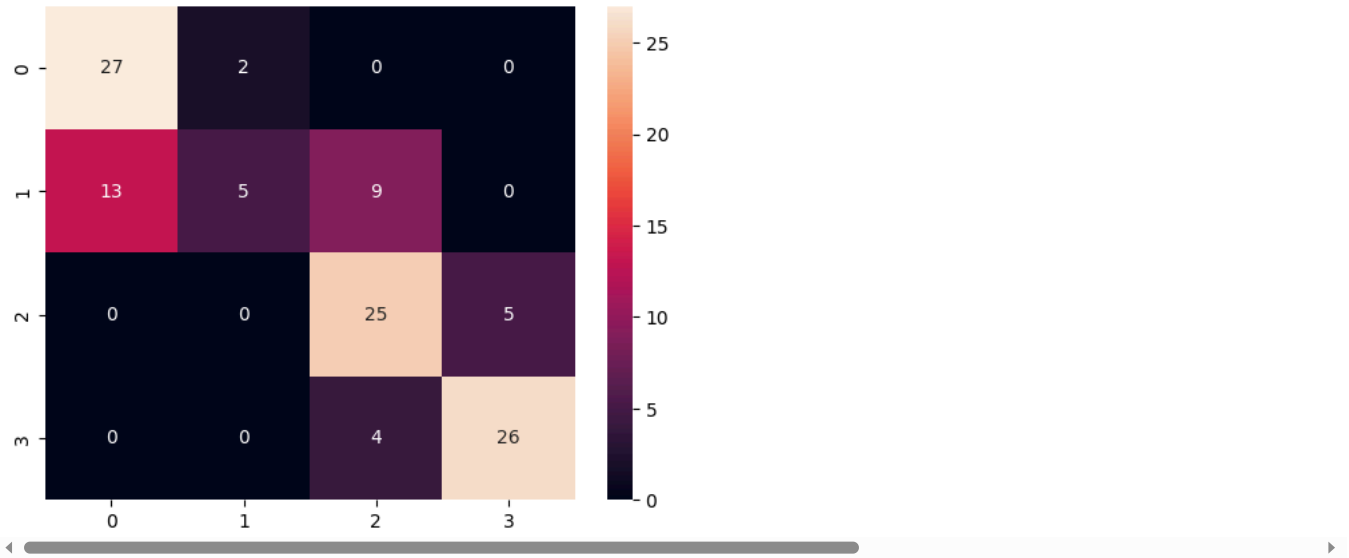

```
accuracy
        training                (min:    0.377, max:    0.852, cur:    0.852)
        validation              (min:    0.531, max:    0.875, cur:    0.859)
Loss
        training                (min:    0.447, max:    4.040, cur:    0.447)
        validation              (min:    0.340, max:    1.556, cur:    0.423)
```

Aplicar Modelo treinado na base de teste

Calcular Métrica de Sensibilidade,, Especificidade e F1-Score com base na matriz de confusão

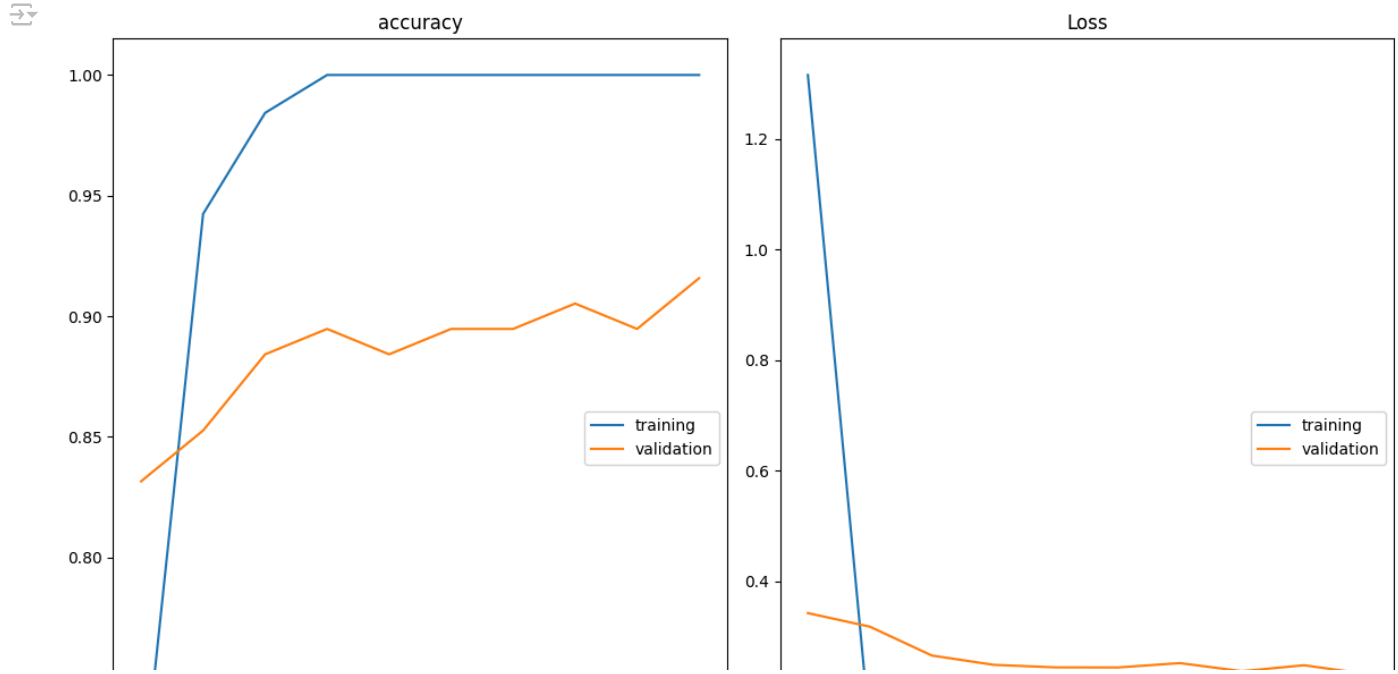
```
##Aplicar Modelo treinado na base de teste
### Calcular Métrica de Sensibilidade,, Especificidade e F1-Score com base na matriz de confusão
from sklearn.metrics import accuracy_score
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report, confusion_matrix

model_resnet50.load_weights('img_model_resnet50.weights.best.keras') ## Inicializa o peso com o melhor treino

true_classes = testgen.classes
class_indices = traingen.class_indices
class_indices = dict((v,k) for k,v in class_indices.items())

predictions = model_resnet50.predict(testgen)
predicted_classes = np.argmax(predictions, axis=1)


accuracy = accuracy_score(true_classes, predicted_classes)
precision = precision_score(true_classes, predicted_classes, average='weighted')
f1 = f1_score(true_classes, predicted_classes, average='weighted')
print("Acurácia Modelo Resnet50 com Data Augmentation treinado {:.2f}%".format(accuracy * 100))
print("Precisão Modelo Resnet50 com  Data Augmentation treinado {:.2f}%".format(precision * 100))
print("F1_Score Modelo Resnet50 com  Data Augmentation treinado {:.2f}%".format(f1 * 100))

# Get the names of the ten classes
class_names = testgen.class_indices.keys()

#TODO: Imprimir Matriz de confusão
plot  = confusion_matrix(true_classes, predicted_classes)
sns.heatmap(plot, annot=True, fmt='d', xticklabels=class_names, yticklabels=class_names)
```

```
2/2 [==============================] - 4s 3s/step
Acurácia Modelo Resnet50 com Data Augmentation treinado 71.55%
Precisão Modelo Resnet50 com  Data Augmentation treinado 72.21%
F1_Score Modelo Resnet50 com  Data Augmentation treinado 67.47%
<Axes: >
```



Treino do Modelo sem Data Augmentation

```
%%time
from keras.optimizers import RMSprop
from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard
from livelossplot import PlotLossesKeras

#steps_per_epoch = traingen.samples // BATCH_SIZE
#val_steps = validgen.samples // BATCH_SIZE

n_epochs = 10

optimizer = RMSprop(learning_rate=0.0001)

model_resnet50.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])

# Salva o modelo Keras após cada época, porém só o de melhor resultado
checkpointer = ModelCheckpoint(filepath='img_model_resnet50_noaug.weights.best.keras',
                               verbose=1,
                               save_best_only=True)

# Para o treinamento para prevenir o overfitting
# Não utilizei aqui, pois queria que rodasse todas as 30 épocas
early_stop = EarlyStopping(monitor='val_loss',
                           patience=10,
                           restore_best_weights=True,
                           mode='min')

# Treinamento do Modelo
history_tl = model_resnet50.fit(train_noaug,
                    epochs=n_epochs,
                    #steps_per_epoch=steps_per_epoch,
                    validation_data=validation_noaug,
                    #validation_steps=val_steps,
                    callbacks=[checkpointer, PlotLossesKeras()],
                    #callbacks=[early_stop, checkpointer, PlotLossesKeras()],
                    verbose=False)
```



Aplicar Modelo treinado na base de teste

Calcular Métrica de Sensibilidade,, Especificidade e F1-Score com base na matriz de confusão

```
##Aplicar Modelo treinado na base de teste
### Calcular Métrica de Sensibilidade,, Especificidade e F1-Score com base na matriz de confusão
from sklearn.metrics import accuracy_score
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report, confusion_matrix

model_resnet50.load_weights('img_model_resnet50_noaug.weights.best.keras') ## Inicializa o peso com o melhor treino

true_classes = testgen.classes
class_indices = traingen.class_indices
class_indices = dict((v,k) for k,v in class_indices.items())

predictions = model_resnet50.predict(testgen)
predicted_classes = np.argmax(predictions, axis=1)


accuracy = accuracy_score(true_classes, predicted_classes)
precision = precision_score(true_classes, predicted_classes, average='weighted')
f1 = f1_score(true_classes, predicted_classes, average='weighted')
print("Acurácia Modelo Resnet50 sem Data Augmentation treinado {:.2f}%".format(accuracy * 100))
print("Precisão Modelo Resnet50 sem Data Augmentation treinado {:.2f}%".format(precision * 100))
print("F1_Score Modelo Resnet50 sem Data Augmentation treinado {:.2f}%".format(f1 * 100))

# Get the names of the ten classes
class_names = testgen.class_indices.keys()

```