

IAA011 – VISÃO COMPUTACIONAL

Equipe 16

Ana Beatriz Kindinger

Daniel Victor Andrade

Marlon Mateus Prudente de Oliveira

De acordo com os treinamentos e testes executados nesse trabalho, obtivemos os seguintes resultados:

1- Extração de Características:

Base de treino	Sensibilidade	Especificidade	F1-Score
RF - LBP	0.766060	0.789186	0.757797
RF - CNN	0.841730	0.850291	0.842091
SVM - LBP	0.786877	0.826087	0.790327
SVM - CNN	0.831258	0.846108	0.83201
RNA - LBP	0.74122	0.779888	0.743243
RNA - CNN	0.825351	0.830564	0.824516

Base de Teste	Sensibilidade	Especificidade	F1-Score
RF - LBP	0.689026	0.697747	0.691186
RF - CNN	0.794087	0.795062	0.786184
SVM - LBP	0.683718	0.721638	0.691717
SVM - CNN	0.74934	0.74087	0.736752
RNA - LBP	0.631903	0.646659	0.634835
RNA - CNN	0.7555	0.753941	0.751216

O modelo escolhido na extração de características foi o Random Forest CNN que foi melhor em todas as métricas tanto na base de treino como da de testes.

2- Redes Neurais:

Redes Neurais - Base de Teste	Sensibilidade	Especificidade	F1-Score
VGG16 com D.A.	60.86%	62.07%	59.20%
VGG16 sem D.A.	48.97%	44.83%	44.28%
ResNet50 com D.A.	80.74%	75.86%	74.07%
ResNet50 sem D.A.	80.38%	78.45%	76.36%

O modelo escolhido dentre as redes neurais foi o Resnet50 sem data augmentation, pois performou melhor em Especificidade e em F1-Score

✓ 1 - Extração de Características

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import zipfile
4 from google.colab.patches import cv2_imshow
5 import os
6 import cv2 as cv
7 from skimage.feature import local_binary_pattern
8 import pandas as pd
```

```
1 !wget /content/Train_Warwick.zip
2 !wget /content/Test_Warwick.zip
3
4 # Extraí o conteúdo do arquivo zip
5 !unzip Train_Warwick.zip
6 !unzip Test_Warwick.zip
```



Mostrar saída oculta

✓ Carregar base de dados

```
1 # prompt: carregue a pasta Test_4cl_amostra em uma base de dados de imagens
2 # ## Carregar base de dados
3
4
5 def load_images_from_folder(folder):
6     images = []
7     for filename in os.listdir(folder):
8         img = cv.imread(os.path.join(folder,filename))
9         if img is not None:
10             images.append((img, filename, i))
11     return images
12
13 train_images = []
14
15 for i in range(4):
16     train_images += load_images_from_folder('Train_4cls_amostra/' + str(i) + '/')
17
18 #train_images
19
```

```
1 ''' separando em treino de validação, separando as imagens de 1 paciente por
2 categoria (aprox. 30 imagens por paciente) temos ~20% da base de treino
3 Pacientes escolhidos: 01, 14, 04 e 06'''
4
```

```
5 dados_treino = []
6 dados_val = []
7
8 for image, filename, cat in train_images:
9     if filename.startswith('01'):
10         dados_val.append((image, cat))
11     elif filename.startswith('14'):
12         dados_val.append((image, cat))
13     elif filename.startswith('04'):
14         dados_val.append((image, cat))
15     elif filename.startswith('06'):
16         dados_val.append((image, cat))
17     else:
18         dados_treino.append((image, cat))
19
20 len(dados_treino), len(dados_val)
```



(477, 116)

```
1 # exibindo as imagens com o nome do arquivo e categoria
2
3 import matplotlib.pyplot as plt
4
5 # Assuming your tuple structure is (image, filename, label)
6 for image, filename, cat in train_images:
7     # Display the image
8     plt.imshow(cv.cvtColor(image, cv.COLOR_BGR2RGB)) # Convert to RGB for Matplotlib
9     plt.title(f"Filename: {filename}, Label: {cat}")
```

```

10 plt.show()
11
12 # Print other information
13 print(f"Image shape: {image.shape}")
14 print(f"Filename: {filename}")
15 print(f"Label: {cat}")
16 print("-" * 20) # Separator

```



Mostrar saída oculta

```

1 #Extrai características utilizando LBP
2
3 def extract_lbp_features(image):
4     gray_image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
5     radius = 3 # Adjust as needed
6     n_points = 8 * radius # Adjust as needed
7     lbp = local_binary_pattern(gray_image, n_points, radius, method='uniform')
8     # Calculate histogram to represent the LBP features
9     n_bins = int(lbp.max() + 1)
10    hist, _ = np.histogram(lbp, density=True, bins=n_bins, range=(0, n_bins))
11    return hist
12
13 # Extraí característica e salva em lista junto com a categoria
14 data_treino = []
15 data_val = []
16
17 #Base treino
18 for image, cat in dados_treino:
19     features_lbp = extract_lbp_features(image)
20     data_treino.append(np.concatenate([cat, features_lbp]))
21
22 #Base validação
23 for image, cat in dados_val:
24     features_lbp_val = extract_lbp_features(image)
25     data_val.append(np.concatenate([cat, features_lbp_val]))
26
27
28 #cria um dataframe pandas e salva em csv
29 #base treino
30 colunas = ['cat'] + [f'lbp_{i}' for i in range(len(features_lbp))]
31 df_treino_lbp = pd.DataFrame(data_treino, columns=colunas)
32
33 df_treino_lbp.to_csv('lbp_features.csv', index=False)
34
35 #base validação
36 colunas = ['cat'] + [f'lbp_{i}' for i in range(len(features_lbp_val))]
37 df_val_lbp = pd.DataFrame(data_val, columns=colunas)
38
39 df_val_lbp.to_csv('lbp_features_val.csv', index=False)


1 #Extrai características utilizando CNN VGG16
2
3 from tensorflow.keras.applications.vgg16 import VGG16
4 from tensorflow.keras.preprocessing import image
5 from tensorflow.keras.applications.vgg16 import preprocess_input
6 from tensorflow.keras.models import Model
7
8
9 # Carrega o modelo VGG16 pré-treinado (sem a camada de classificação)
10 base_model = VGG16(weights='imagenet', include_top=True)
11
12 # Cria um novo modelo que gera as características da camada 'fc2'
13 model = Model(inputs=base_model.input, outputs=base_model.get_layer('fc2').output)
14
15 def extract_features(img):
16     # Redimensiona a imagem para o tamanho de entrada do VGG16 (224x224)
17     img = cv.resize(img, (224, 224))
18     # Converte a imagem para um array NumPy
19     #x = image.img_to_array(img)
20     # Expande as dimensões para criar um batch de uma única imagem
21     x = np.expand_dims(img, axis=0)
22     # Pré-processa a imagem de acordo com o VGG16
23     x = preprocess_input(x)
24     # Extraí as características usando o modelo
25     features = model.predict(x)
26     # Retorna as características como um array NumPy
27     return features[0]
28
29 features_list = []
30 features_list_val = []
31

```

```

32 # BASE DE TREINO
33 for image, cat in dados_treino:
34     features_cnn = extract_features(image)
35     features_list.append(np.concatenate([cat, features_cnn]))
36
37 # BASE DE VALIDAÇÃO
38 for image, cat in dados_val:
39     features_cnn_val = extract_features(image)
40     features_list_val.append(np.concatenate([cat, features_cnn_val]))
41

```



Mostrar saída oculta

```

1 # Create DataFrame and save to CSV
2 #BASE TREINO
3 num_features = features_cnn.shape[0] # Get the number of features from features_cnn
4 colunas_cnn = ['cat'] + [f'cnn_{i}' for i in range(num_features)]
5 df_treino_cnn = pd.DataFrame(features_list, columns=colunas_cnn)
6
7 df_treino_cnn.to_csv('cnn_features.csv', index=False)
8
9 # BASE VALIDAÇÃO
10 num_features = features_cnn_val.shape[0] # Get the number of features from features_cnn
11 colunas_cnn = ['cat'] + [f'cnn_{i}' for i in range(num_features)]
12 df_val_cnn = pd.DataFrame(features_list_val, columns=colunas_cnn)
13
14 df_val_cnn.to_csv('cnn_features_val.csv', index=False)

```

✓ TREINAMENTO DOS MODELOS

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

```

1 #função para avaliação de resultados
2
3 def avaliar_resultados(y_real, y_pred):
4     # Avaliando a acurácia
5     accuracy = accuracy_score(y_real, y_pred)
6     print(f"Acurácia do modelo: {accuracy}")
7
8     # Exibindo a matriz de confusão
9     cm = confusion_matrix(y_real, y_pred)
10    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
11    disp.plot(cmap=plt.cm.Blues)
12    plt.show()
13
14    # Calculando as métricas por classe e a média macro
15    report = classification_report(y_real, y_pred, output_dict=True)
16    sensibilidade_macro = report['macro avg']['recall'] # Sensibilidade (Recall) macro
17    especificidade_macro = report['macro avg']['precision'] # Especificidade (Precision) macro
18    f1_score_macro = report['macro avg']['f1-score'] # F1-Score macro
19
20    print(f"Sensibilidade (Macro): {sensibilidade_macro}")
21    print(f"Especificidade (Macro): {especificidade_macro}")
22    print(f"F1-Score (Macro): {f1_score_macro}")
23
24    return {
25        "acuracia": accuracy,
26        "sensibilidade_macro": sensibilidade_macro,
27        "especificidade_macro": especificidade_macro,
28        "f1_score_macro": f1_score_macro
29    }

```

✓ RANDOM FOREST

✓ LBP

```

1 #LBP
2
3 from sklearn.ensemble import RandomForestClassifier
4 from sklearn.metrics import accuracy_score
5 from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report
6
7 #df_treino_lbp = pd.read_csv('lbp_features.csv')
8

```

```

9 # Separate features (X) and target (y)
10 X_train = df_treino_lbp.drop('cat', axis=1) # Caracteristicas
11 y_train = df_treino_lbp['cat'] # alvo
12
13 # criando e treinando o modelo
14 rf_model_lbp = RandomForestClassifier(n_estimators=300,
15                                     max_depth=15,
16                                     min_samples_split=8,
17                                     min_samples_leaf=5,
18                                     max_features='sqrt')
19 #rf_model = RandomForestClassifier()
20
21 rf_model_lbp.fit(X_train, y_train)
22
23

```



```

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

```

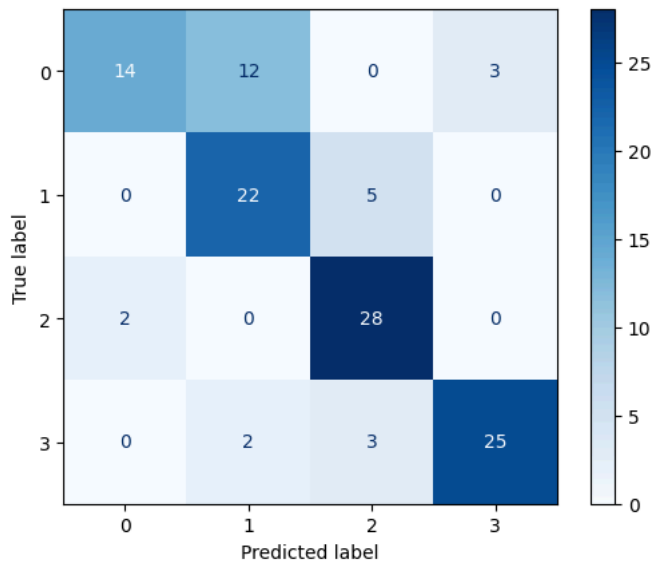
```

1 # Avaliando o modelo para o tuning dos hiperparametros
2 y_pred = rf_model_lbp.predict(df_val_lbp.drop('cat', axis=1))
3 resultado_tuning_rf_lbp = avaliar_resultados(df_val_lbp['cat'], y_pred)
4

```



Acurácia do modelo: 0.7672413793103449



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

✓ CNN

```

1 # CNN
2 from sklearn.ensemble import RandomForestClassifier
3 from sklearn.metrics import accuracy_score
4 from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
5
6 #df_treino_cnn = pd.read_csv('cnn_features.csv')
7
8 # Separate features (X) and target (y)
9 X_train = df_treino_cnn.drop('cat', axis=1) # Caracteristicas
10 y_train = df_treino_cnn['cat'] # alvo
11
12 # instanciando com hiperparametros
13 rf_model_cnn = RandomForestClassifier(n_estimators=250,
14                                     max_depth=12,
15                                     min_samples_split=10,
16                                     min_samples_leaf=5,
17                                     max_features='sqrt')
18 #rf_model = RandomForestClassifier()
19
20 #trainando o modelo
21 rf_model_cnn.fit(X_train, y_train)
22

```

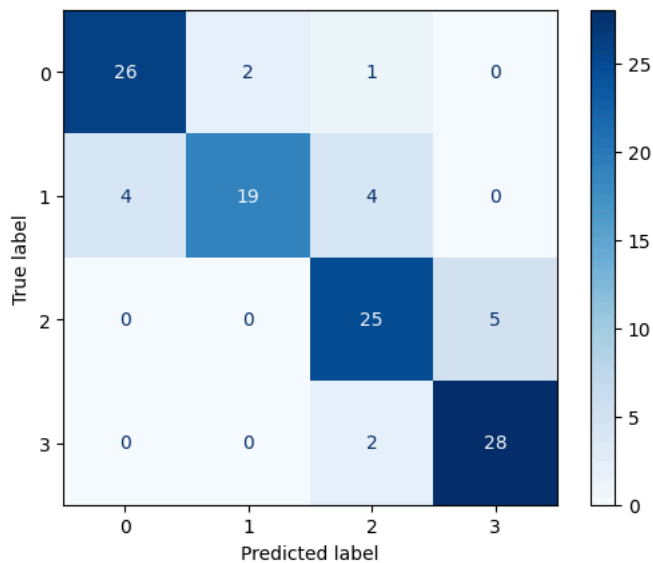


```
RandomForestClassifier
RandomForestClassifier(max_depth=12, min_samples_leaf=5, min_samples_split=10,
n_estimators=250)
```

```
1 # Avaliando o modelo para o tuning dos hiperparametros
2 y_pred = rf_model_cnn.predict(df_val_cnn.drop('cat', axis=1))
3 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

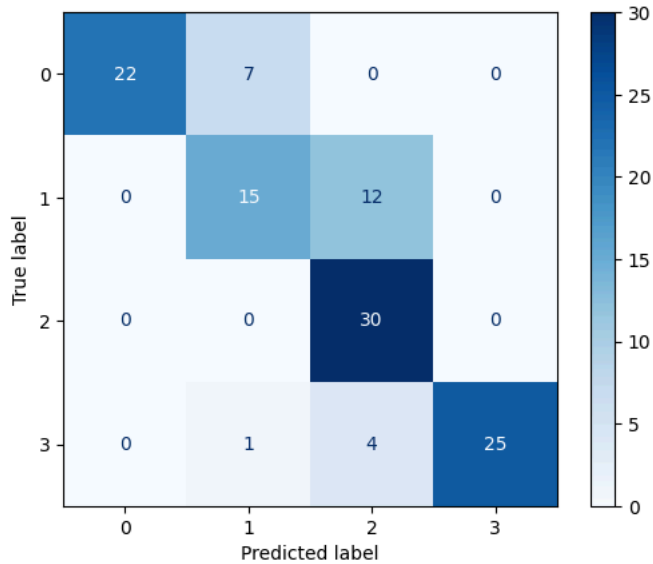
```
1 # LBP
2 from sklearn.svm import SVC
3
4 # Separate features (X) and target (y)
5 X_train = df_treino_lbp.drop('cat', axis=1) # Caracteristicas
6 y_train = df_treino_lbp['cat'] # alvo
7
8 #instanciando com hiperparametros
9 svm_model_lbp = SVC(kernel='rbf', C=100, gamma = 5)
10
11
12 #treinando o modelo
13 svm_model_lbp.fit(X_train, y_train)
14
15
```



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

```
1 # Avaliando o modelo para o tuning dos hiperparametros
2 y_pred = svm_model_lbp.predict(df_val_lbp.drop('cat', axis=1))
3 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

```
1 #CNN
2 # Separate features (X) and target (y)
3 X_train = df_treino_cnn.drop('cat', axis=1) # Caracteristicas
4 y_train = df_treino_cnn['cat'] # alvo
5
6 #instanciando com hiperparametros
7 svm_model_cnn = SVC(kernel='linear', C=1)
8
9 #treinando o modelo
10 svm_model_cnn.fit(X_train, y_train)
```

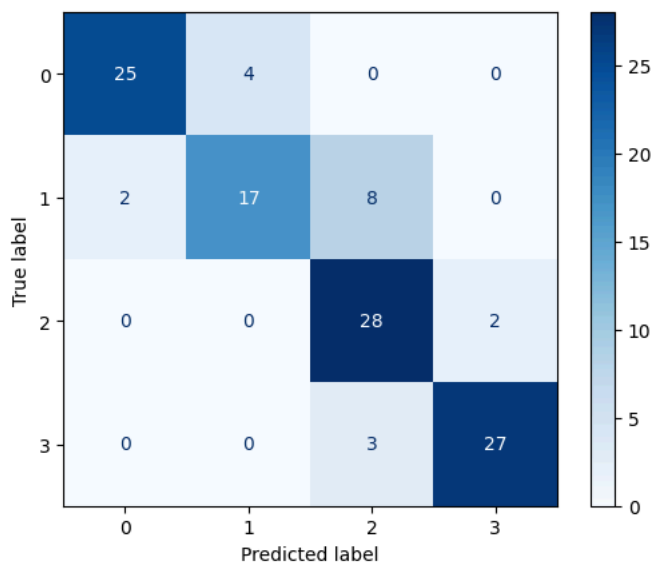
↔

▼ SVC ⓘ ?

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

```
1 # Avaliando o modelo para o tuning dos hiperparametros
2 y_pred = svm_model_cnn.predict(df_val_cnn.drop('cat', axis=1))
3 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

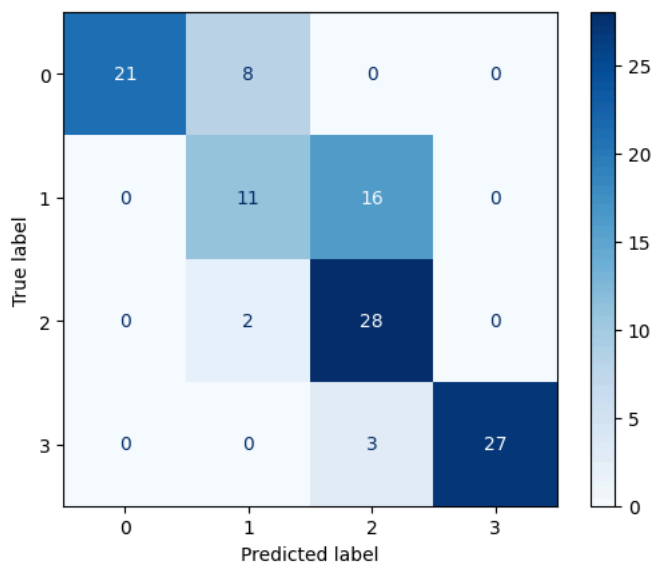
```
1 #LBP
2 from sklearn.neural_network import MLPClassifier
3
4 # Separate features (X) and target (y) - treinamento
5 X_train = df_treino_lbp.drop('cat', axis=1) # Caracteristicas
6 y_train = df_treino_lbp['cat'] # alvo
7
8 #instanciando com hiperparametros
9 rna_model_lbp = MLPClassifier(hidden_layer_sizes=(100,),
10                               activation='relu',
11                               solver='adam',
12                               alpha=0.0001,
13                               batch_size='auto',
14                               max_iter=1000)
15
16 #treinando modelo
17 rna_model_lbp.fit(X_train, y_train)
18
```

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

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

```
1 # Avaliando o modelo para o tuning dos hiperparametros
2 y_pred = rna_model_lbp.predict(df_val_lbp.drop('cat', axis=1))
3 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

```
1 # CNN
2 # Separate features (X) and target (y) - treinamento
3 X_train = df_treino_cnn.drop('cat', axis=1) # Caracteristicas
4 y_train = df_treino_cnn['cat'] # alvo
5
6 #instanciando com hiperparametros
7 rna_model_cnn = MLPClassifier(hidden_layer_sizes=(100,),
8                               activation='relu',
9                               solver='adam',
10                               alpha=0.0001,
11                               batch_size='auto',
12                               max_iter=1000)
```



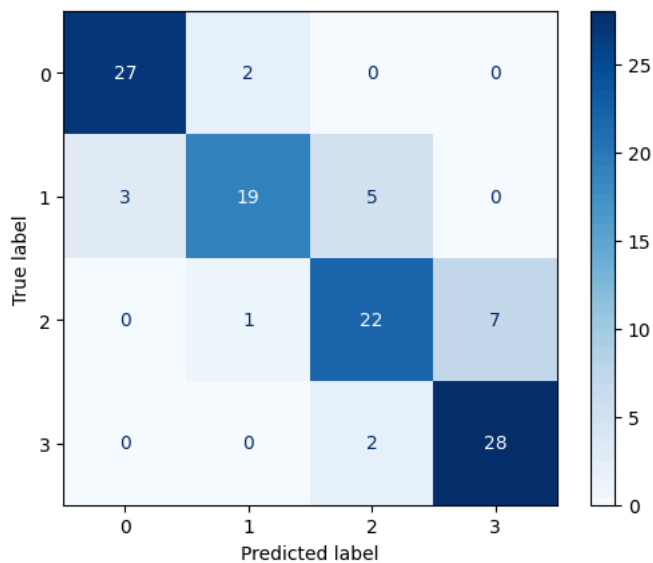
```

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

1 # Avaliando o modelo para o tuning 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))

```

```
4 y_pred_rf_cnn = rf_model_cnn.predict(df_test_cnn.drop('cat', axis=1))
5
6 y_pred_svm_lbp = svm_model_lbp.predict(df_test_lbp.drop('cat', axis=1))
7 y_pred_svm_cnn = svm_model_cnn.predict(df_test_cnn.drop('cat', axis=1))
8
9 y_pred_rna_lbp = rna_model_lbp.predict(df_test_lbp.drop('cat', axis=1))
10 y_pred_rna_cnn = rna_model_cnn.predict(df_test_cnn.drop('cat', axis=1))
11
12 # avaliação dos resultados
13 print("Random Forest - LBP")
14 resultados_rf_lbp = avaliar_resultados(df_test_lbp['cat'], y_pred_rf_lbp)
15 print("\nRandom Forest - CNN")
16 resultados_rf_cnn = avaliar_resultados(df_test_cnn['cat'], y_pred_rf_cnn)
17
18 print("\nSVM - LBP")
19 resultados_svm_lbp = avaliar_resultados(df_test_lbp['cat'], y_pred_svm_lbp)
20 print("\nSVM - CNN")
21 resultados_svm_cnn = avaliar_resultados(df_test_cnn['cat'], y_pred_svm_cnn)
22
23 print("\nRNA - LBP")
24 resultados_rna_lbp = avaliar_resultados(df_test_lbp['cat'], y_pred_rna_lbp)
25 print("\nRNA - CNN")
26 resultados_rna_cnn = avaliar_resultados(df_test_cnn['cat'], y_pred_rna_cnn)
```

✓ 2 - Redes Neurais

```
1 ##Mudar para essa versão do Tensorflow
2 %pip install tensorflow==2.15
```

 **Mostrar saída oculta**

```
1 from tensorflow.keras.applications import VGG16, ResNet50
2 from tensorflow.keras import layers, models
3 from keras.layers import Dense, Dropout, Flatten
4 from keras.models import Model
5 import cv2 as cv
6 import zipfile
7 import os
8
9
10
11 # Módulo para imprimir os gráficos de treinamento de forma dinâmica.
12 !pip install livelossplot
```

 **Mostrar saída oculta**

```
1 #from google.colab import drive
2 #drive.mount('/content/drive/')
3
4 ##Tentativa de usar o zip da pasta do meu google drive
```

 Mounted at /content/drive/

```
1 !wget /content/Train_Warwick.zip
2 !wget /content/Test_Warwick.zip
3
4 # Extraí o conteúdo do arquivo zip
5 !unzip Train_Warwick.zip
6 !unzip Test_Warwick.zip
```

 **Mostrar saída oculta**

```
1 def load_images_from_folder(folder):
2     '''Carrega as imagens dos diretórios em uma lista com o nome do arquivo e classe'''
3     images = []
4     for filename in os.listdir(folder):
5         img = cv.imread(os.path.join(folder,filename))
6         if img is not None:
7             images.append((img, filename, i))
8     return images
9
10 def resize_images(images):
11     '''Redimensiona as imagens para 224x224 como esperado por VGG16, ResNet50'''
12     resized_images = []
13     for image, filename, i in images:
14         resized_image = cv.resize(image, (224, 224))
15         resized_images.append((resized_image, filename, i))
16     return resized_images
17
18 # carregando as imagens em 2 listas
19 # Move os dados do diretório images para images/train ou images/test:
20 import shutil
21 from collections import defaultdict
22 import json
23 from pathlib import Path
24 import os
25
26 os.makedirs('images/train', exist_ok=True)
27 os.makedirs('images/test', exist_ok=True)
28
29 temp_train_images = [[],[],[],[]]
30 test_images = [[],[],[],[]]
31
32 for i in range(4):
33     temp_train_images[i].append(load_images_from_folder('Train_4cls_amostra/' + str(i) + '/'))
34
35 for i in range(4):
36     test_images[i].append(load_images_from_folder('Test_4cl_amostra/' + str(i) + '/'))
37
38
```

```

21 ''' separando em treino de validação, separando as imagens de 1 paciente por
22 categoria (aprox. 30 imagens por paciente) temos ~20% da base de treino
23 Pacientes escolhidos: 01, 14, 04 e 06'''
24
25
26 for i in range(4):
27     for image in temp_train_images[i]:
28         image = resize_images(image)
29         for img, filename, cat in image:
30             os.makedirs('images/test/' + str(i) + '/', exist_ok=True)
31             os.makedirs('images/train/' + str(i) + '/', exist_ok=True)
32             if filename.startswith('01'):
33                 cv.imwrite(os.path.join('images/test/' + str(i) + '/', filename), img)
34             elif filename.startswith('14'):
35                 cv.imwrite(os.path.join('images/test/' + str(i) + '/', filename), img)
36             elif filename.startswith('04'):
37                 cv.imwrite(os.path.join('images/test/' + str(i) + '/', filename), img)
38             elif filename.startswith('06'):
39                 cv.imwrite(os.path.join('images/test/' + str(i) + '/', filename), img)
40             else:
41                 cv.imwrite(os.path.join('images/train/' + str(i) + '/', filename), img)
42

```

```

1 import os
2
3 class_subset = sorted(os.listdir('Train_4cls_amostra/'))
4 print(class_subset)

```

```

→ ['0', '1', '2', '3']

```

```

1 # carregando as redes com transfer learning sem as camadas fully connected
2
3 base_model_vgg16 = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
4 base_model_vgg16_noaug = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
5 base_model_resnet50 = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
6 base_model_resnet50_noaug = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
7
8 for layer in base_model_vgg16.layers:
9     layer.trainable = False
10
11 for layer in base_model_resnet50.layers:
12     layer.trainable = False
13
14 for layer in base_model_vgg16_noaug.layers:
15     layer.trainable = False
16
17 for layer in base_model_resnet50_noaug.layers:
18     layer.trainable = False

```

```

1 # Adicionar camadas fully connected no VGG16
2
3 # camadas próprias - você pode colocar mais se quiser
4 # A saída da resnet será a entrada da camada criada
5 x = Flatten()(base_model_vgg16.output)
6
7 # camada de classificação com as 4 classes utilizadas
8 prediction = Dense(4, activation='softmax')(x)
9
10 # Criação do Objeto Modelo (a parte da resnet + as camadas Fully connected criadas)
11 model_vgg16 = Model(inputs=base_model_vgg16.input, outputs=prediction)
12

```

```

1 # Adicionar camadas fully connected no VGG16 sem Data Augmentation
2
3 # camadas próprias - você pode colocar mais se quiser
4 # A saída da resnet será a entrada da camada criada
5 x_noaug = Flatten()(base_model_vgg16_noaug.output)
6
7 # camada de classificação com as 4 classes utilizadas
8 prediction_noaug = Dense(4, activation='softmax')(x_noaug)
9
10 # Criação do Objeto Modelo (a parte da resnet + as camadas Fully connected criadas)
11 model_vgg16_noaug = Model(inputs=base_model_vgg16_noaug.input, outputs=prediction_noaug)

```

```

1 # Adicionar camadas fully connected no resnet50
2
3 # camadas próprias - você pode colocar mais se quiser
4 # A saída da resnet será a entrada da camada criada
5 x_res = Flatten()(base_model_resnet50.output)
6

```

```

7 # camada de classificação com as 4 classes utilizadas
8 prediction_res = Dense(len(class_subset), activation='softmax')(x_res)
9
10 # Criação do Objeto Modelo (a parte da resnet + as camadas Fully connected criadas)
11 model_resnet50 = Model(inputs=base_model_resnet50.input, outputs=prediction_res)

1 # Adicionar camadas fully connected no resnet50 sem data augmentation
2
3 # camadas próprias - você pode colocar mais se quiser
4 # A saída da resnet será a entrada da camada criada
5 x_res_noaug = Flatten()(base_model_resnet50_noaug.output)
6
7 # camada de classificação com as 4 classes utilizadas
8 prediction_res_noaug = Dense(len(class_subset), activation='softmax')(x_res_noaug)
9
10 # Criação do Objeto Modelo (a parte da resnet + as camadas Fully connected criadas)
11 model_resnet50_noaug = Model(inputs=base_model_resnet50_noaug.input, outputs=prediction_res_noaug)

```

▼ Data Augmentation

```

1 from tensorflow.keras.applications.resnet50 import preprocess_input
2 from tensorflow.keras.preprocessing.image import ImageDataGenerator
3
4 train_generator = ImageDataGenerator(
5     rotation_range=90,
6     brightness_range=[0.1, 0.7],
7     width_shift_range=0.5,
8     height_shift_range=0.5,
9     horizontal_flip=True,
10    vertical_flip=True,
11    validation_split=0.2,
12    preprocessing_function=preprocess_input)
13
14 test_generator = ImageDataGenerator(preprocessing_function=preprocess_input)

```

```

1 BATCH_SIZE = 64 # quantidade de imagens criadas em cada ciclo
2
3 traingen = train_generator.flow_from_directory('images/train',
4     target_size=(224, 224),
5     batch_size=BATCH_SIZE,
6     class_mode='categorical',
7     classes=class_subset,
8     subset='training',
9     shuffle=True,
10    seed=42)
11
12 validgen = train_generator.flow_from_directory('images/train',
13     target_size=(224, 224),
14     batch_size=BATCH_SIZE,
15     class_mode='categorical',
16     classes=class_subset,
17     subset='validation',
18     shuffle=True,
19     seed=42)

```

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

```

1 testgen = test_generator.flow_from_directory('images/test',
2     target_size=(224, 224),
3     batch_size=BATCH_SIZE,
4     class_mode=None,
5     classes=class_subset,
6     shuffle=False,
7     seed=42)

```

➡ Found 116 images belonging to 4 classes.

```

1 import tensorflow as tf
2 train_noaug = tf.keras.utils.image_dataset_from_directory('images/train',
3     labels='inferred',
4     label_mode='categorical',
5     subset="training",
6     class_names=class_subset,
7     validation_split = 0.2,
8     image_size=(224, 224),
9     batch_size=BATCH_SIZE,

```

```

10                                     shuffle=True,
11                                     seed=42)
12
13 validation_noaug = tf.keras.utils.image_dataset_from_directory('images/train',
14                                     labels='inferred',
15                                     label_mode='categorical',
16                                     subset="validation",
17                                     class_names=class_subset,
18                                     validation_split = 0.2,
19                                     image_size=(224, 224),
20                                     batch_size=BATCH_SIZE,
21                                     shuffle=True,
22                                     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.

```

1 test_noaug = tf.keras.utils.image_dataset_from_directory('images/test',
2                                     labels='inferred',
3                                     label_mode='categorical',
4                                     class_names=class_subset,
5                                     image_size=(224, 224),
6                                     batch_size=BATCH_SIZE,
7                                     shuffle=False,
8                                     seed=42)

```

Found 116 files belonging to 4 classes.

✓ Treino da VGG16

```

1 model_vgg16.summary()
2 model_vgg16_noaug.summary()

```

Model: "model_2"

Layer (type)	Output Shape	Param #
input_7 (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_2 (Flatten)	(None, 25088)	0
dense_2 (Dense)	(None, 4)	100356

```

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

```

Model: "model_3"

Layer (type)	Output Shape	Param #
input_8 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792

```
1 model_resnet50.summary()
2 model_resnet50_noaug.summary()
```

Model: "model_4"

Layer (type)	Output Shape	Param #	Connected to
input_9 (InputLayer)	[(None, 224, 224, 3)]	0	[]
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	['input_9[0][0]']
conv1_conv (Conv2D)	(None, 112, 112, 64)	9472	['conv1_pad[0][0]']
conv1_bn (BatchNormalization)	(None, 112, 112, 64)	256	['conv1_conv[0][0]']
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 (Conv2D)	(None, 56, 56, 64)	4160	['pool1_pool[0][0]']
conv2_block1_1_bn (BatchNormalization)	(None, 56, 56, 64)	256	['conv2_block1_1_conv[0][0]']
conv2_block1_1_relu (Activation)	(None, 56, 56, 64)	0	['conv2_block1_1_bn[0][0]']
conv2_block1_2_conv (Conv2D)	(None, 56, 56, 64)	36928	['conv2_block1_1_relu[0][0]']
conv2_block1_2_bn (BatchNormalization)	(None, 56, 56, 64)	256	['conv2_block1_2_conv[0][0]']
conv2_block1_2_relu (Activation)	(None, 56, 56, 64)	0	['conv2_block1_2_bn[0][0]']
conv2_block1_0_conv (Conv2D)	(None, 56, 56, 256)	16640	['pool1_pool[0][0]']
conv2_block1_3_conv (Conv2D)	(None, 56, 56, 256)	16640	['conv2_block1_2_relu[0][0]']
conv2_block1_0_bn (BatchNormalization)	(None, 56, 56, 256)	1024	['conv2_block1_0_conv[0][0]']
conv2_block1_3_bn (BatchNormalization)	(None, 56, 56, 256)	1024	['conv2_block1_3_conv[0][0]']
conv2_block1_add (Add)	(None, 56, 56, 256)	0	['conv2_block1_0_bn[0][0]', 'conv2_block1_3_bn[0][0]']
conv2_block1_out (Activation)	(None, 56, 56, 256)	0	['conv2_block1_add[0][0]']
conv2_block2_1_conv (Conv2D)	(None, 56, 56, 64)	16448	['conv2_block1_out[0][0]']

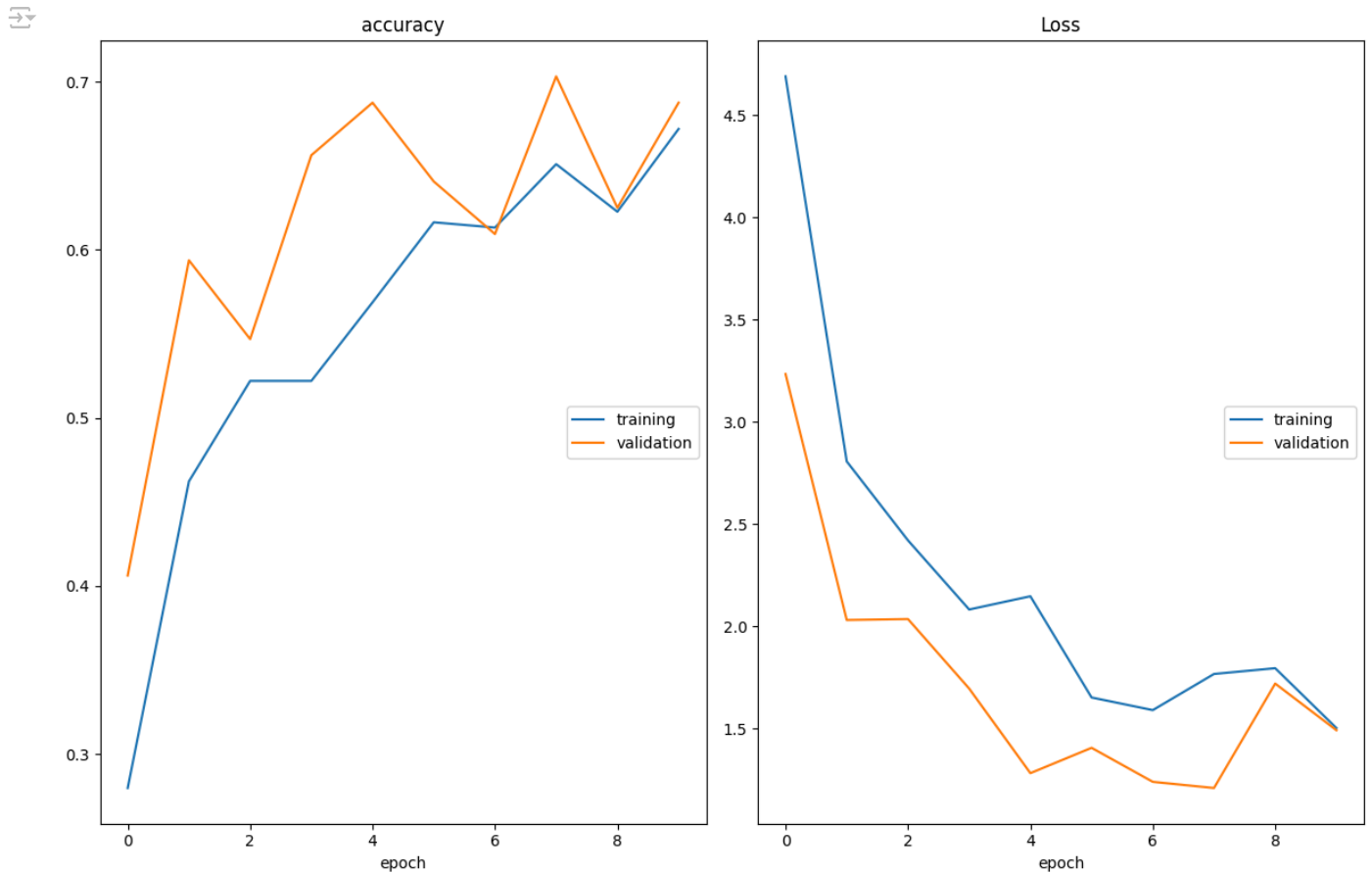
✓ Treinamento VGG16 com Data Augmentation

```
1 %%time
2 from keras.optimizers import RMSprop
3 from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard
4 from livelossplot import PlotLossesKeras
5
6 steps_per_epoch = traingen.samples // BATCH_SIZE
7 val_steps = validgen.samples // BATCH_SIZE
8
9 n_epochs = 10
10
11 optimizer = RMSprop(learning_rate=0.0001)
12
```

```

13 model_vgg16.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
14
15 # Salva o modelo Keras após cada época, porém só o de melhor resultado
16 checkpointer = ModelCheckpoint(filepath='img_model_vgg16.weights.best.keras',
17                                 verbose=1,
18                                 save_best_only=True)
19
20 # Para o treinamento para prevenir o overfitting
21 # Não utilizei aqui, pois queria que rodasse todas as 30 épocas
22 early_stop = EarlyStopping(monitor='val_loss',
23                             patience=10,
24                             restore_best_weights=True,
25                             mode='min')
26
27 # Treinamento do Modelo
28 history_t1 = model_vgg16.fit(traingen,
29                               epochs=n_epochs,
30                               steps_per_epoch=steps_per_epoch,
31                               validation_data=validgen,
32                               validation_steps=val_steps,
33                               callbacks=[checkpointer, PlotLossesKeras()],
34                               #callbacks=[early_stop, checkpointer, PlotLossesKeras()],
35                               verbose=False)

```



```

accuracy
  training      (min:  0.280, max:  0.672, cur:  0.672)
  validation     (min:  0.406, max:  0.703, cur:  0.688)

Loss
  training      (min:  1.504, max:  4.690, cur:  1.504)
  validation     (min:  1.211, max:  3.234, cur:  1.493)

CPU times: user 1min 32s, sys: 6.23 s, total: 1min 38s
Wall time: 1min 57s

```

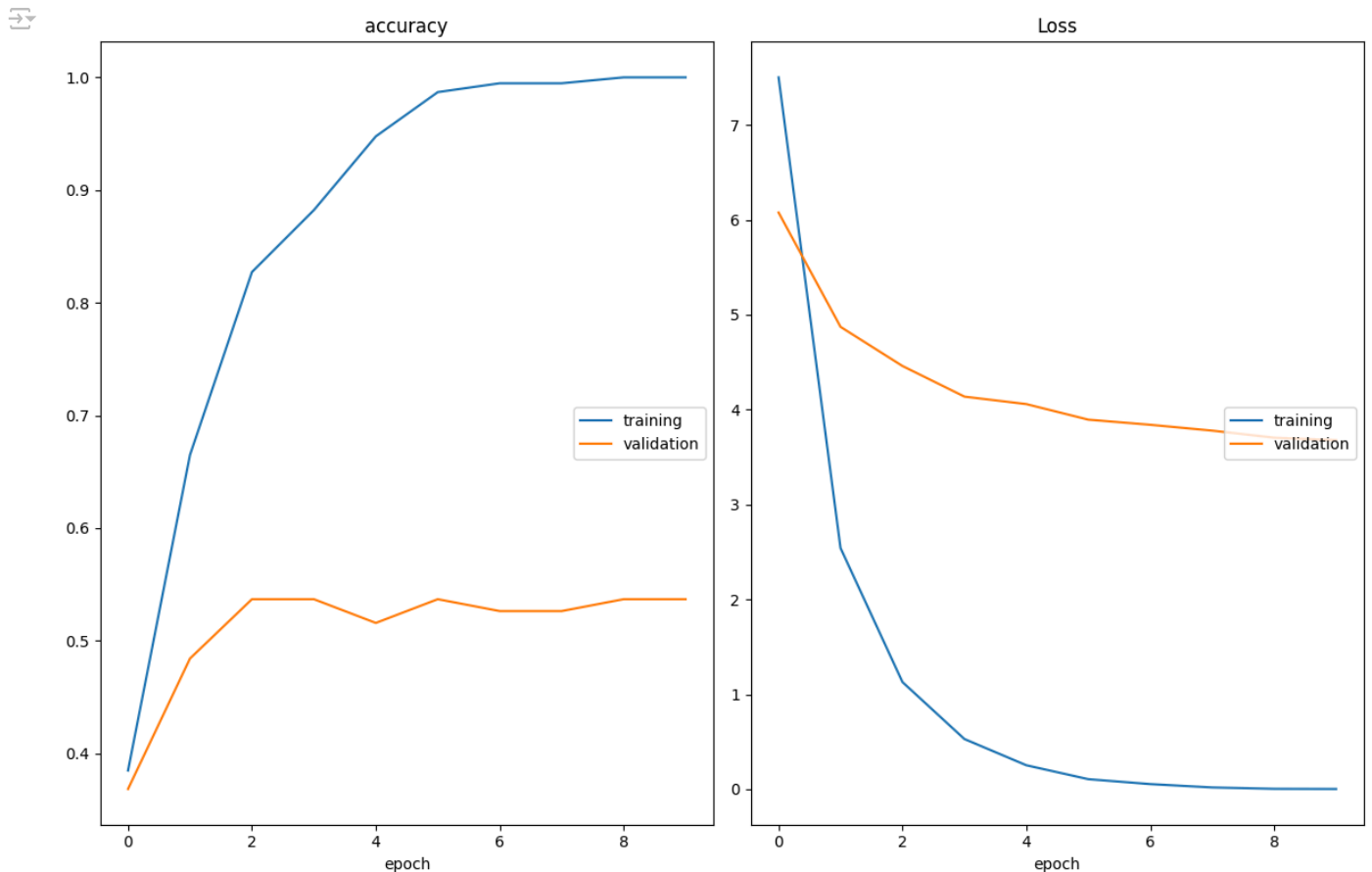
✎ Aplicar Modelo treinado na base de teste

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


```

16 checkpointer = ModelCheckpoint(filepath='img_model_vgg16_noaug.weights.best.keras',
17                                 verbose=1,
18                                 save_best_only=True)
19
20 # Para o treinamento para prevenir o overfitting
21 # Não utilizei aqui, pois queria que rodasse todas as 30 épocas
22 early_stop = EarlyStopping(monitor='val_loss',
23                             patience=10,
24                             restore_best_weights=True,
25                             mode='min')
26
27 # Treinamento do Modelo
28 history_t1 = model_vgg16_noaug.fit(train_noaug,
29                                    epochs=n_epochs,
30                                    #steps_per_epoch=steps_per_epoch,
31                                    validation_data=validation_noaug,
32                                    #validation_steps=val_steps,
33                                    callbacks=[checker, PlotLossesKeras()],
34                                    #callbacks=[early_stop, checker, PlotLossesKeras()],
35                                    verbose=False)

```



```

accuracy
  training      (min:  0.385, max:  1.000, cur:  1.000)
  validation    (min:  0.368, max:  0.537, cur:  0.537)
Loss
  training      (min:  0.003, max:  7.501, cur:  0.003)
  validation    (min:  3.678, max:  6.078, cur:  3.678)
CPU times: user 20.6 s, sys: 6.05 s, total: 26.6 s
Wall time: 39.2 s

```

✓ Aplicar Modelo treinado na base de teste

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

```

1  ##Aplicar Modelo treinado na base de teste
2  ### Calcular Métrica de Sensibilidade,, Especificidade e F1-Score com base na matriz de confusão
3  from sklearn.metrics import accuracy_score
4  import numpy as np

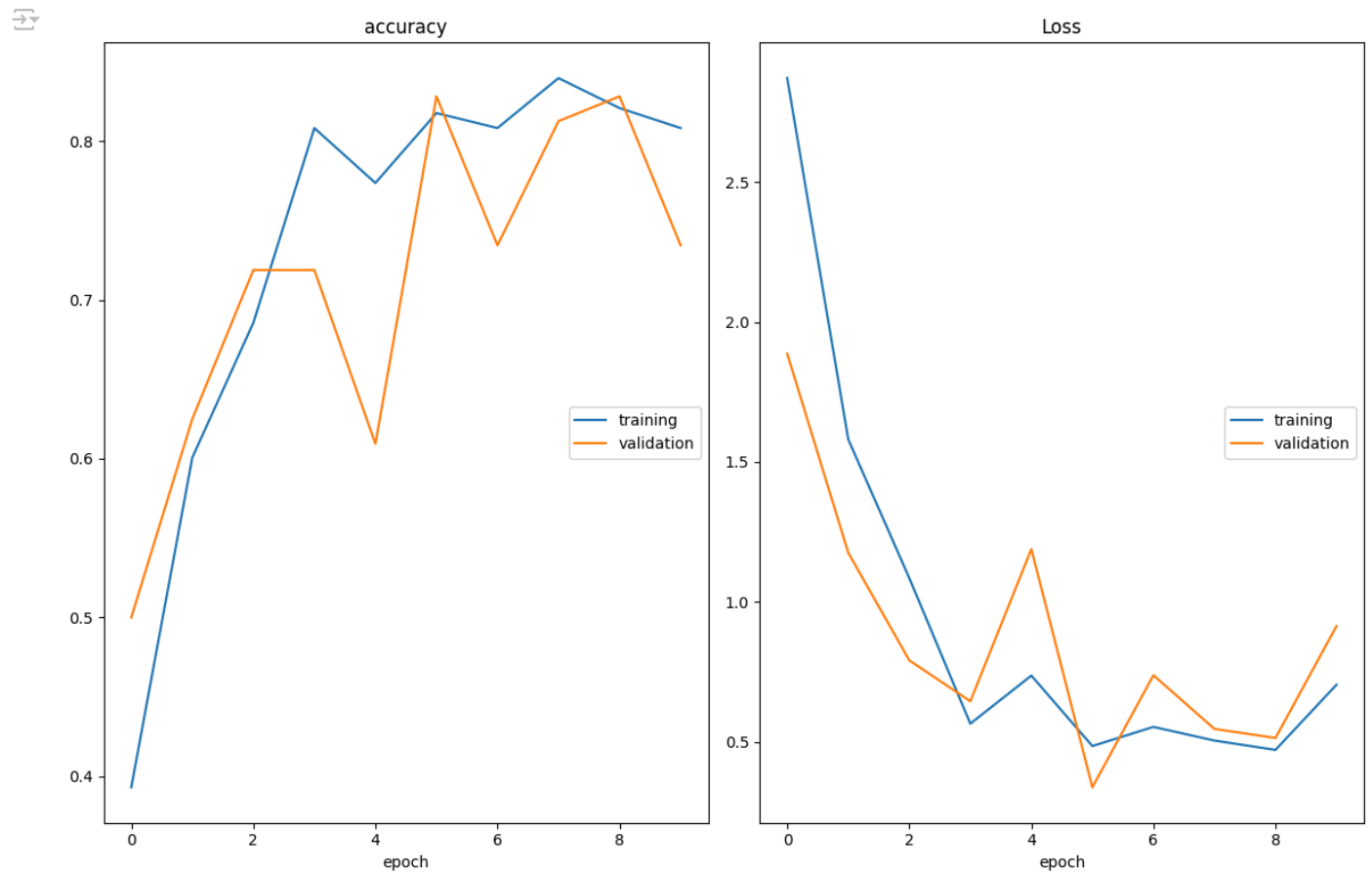
```



```

25         mode='min')
26
27 # Treinamento do Modelo
28 history_tl = model_resnet50.fit(traugen,
29                                 epochs=n_epochs,
30                                 steps_per_epoch=steps_per_epoch,
31                                 validation_data=validgen,
32                                 validation_steps=val_steps,
33                                 callbacks=[checkpointer, PlotLossesKeras()],
34                                 #callbacks=[early_stop, checkpointer, PlotLossesKeras()],
35                                 verbose=False)

```



```

accuracy
  training      (min:  0.393, max:  0.840, cur:  0.808)
 validation     (min:  0.500, max:  0.828, cur:  0.734)

Loss
  training      (min:  0.472, max:  2.870, cur:  0.705)
 validation     (min:  0.339, max:  1.887, cur:  0.914)

CPU times: user 1min 40s, sys: 6.5 s, total: 1min 46s
Wall time: 2min

```

✓ Aplicar Modelo treinado na base de teste

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

```

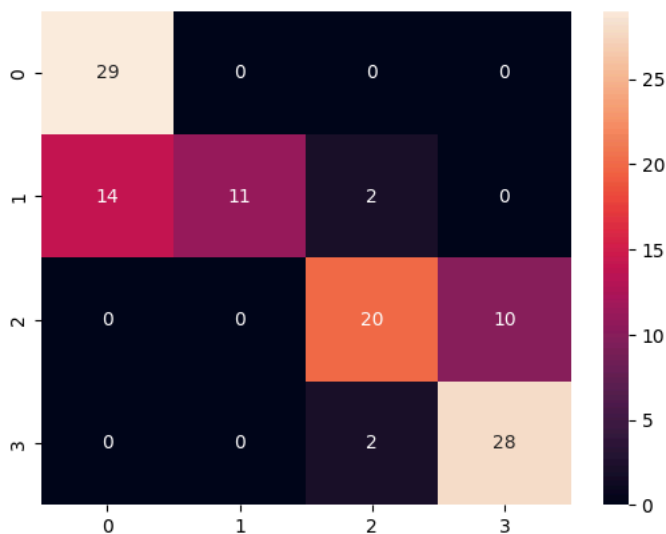
1 ##Aplicar Modelo treinado na base de teste
2 ### Calcular Métrica de Sensibilidade,, Especificidade e F1-Score com base na matriz de confusão
3 from sklearn.metrics import accuracy_score
4 import numpy as np
5 import seaborn as sns
6 import matplotlib.pyplot as plt
7 from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report, confusion_matrix
8
9 model_resnet50.load_weights('img_model_resnet50.weights.best.keras') ## Inicializa o peso com o melhor treino
10
11 true_classes = testgen.classes
12 class_indices = traingen.class_indices
13 class_indices = dict((v,k) for k,v in class_indices.items())

```

```

14
15 predictions = model_resnet50.predict(testgen)
16 predicted_classes = np.argmax(predictions, axis=1)
17
18
19 accuracy = accuracy_score(true_classes, predicted_classes)
20 precision = precision_score(true_classes, predicted_classes, average='weighted')
21 f1 = f1_score(true_classes, predicted_classes, average='weighted')
22 print("Acurácia Modelo Resnet50 com Data Augmentation treinado {:.2f}%".format(accuracy * 100))
23 print("Precisão Modelo Resnet50 com Data Augmentation treinado {:.2f}%".format(precision * 100))
24 print("F1_Score Modelo Resnet50 com Data Augmentation treinado {:.2f}%".format(f1 * 100))
25
26 # Get the names of the ten classes
27 class_names = testgen.class_indices.keys()
28
29 #TODO: Imprimir Matriz de confusão
30 plot = confusion_matrix(true_classes, predicted_classes)
31 sns.heatmap(plot, annot=True, fmt='d', yticklabels=class_names, xticklabels=class_names)
32
33 2/2 [=====] - 1s 242ms/step
Acurácia Modelo Resnet50 com Data Augmentation treinado 75.86%
Precisão Modelo Resnet50 com Data Augmentation treinado 80.74%
F1_Score Modelo Resnet50 com Data Augmentation treinado 74.07%
<Axes: >

```



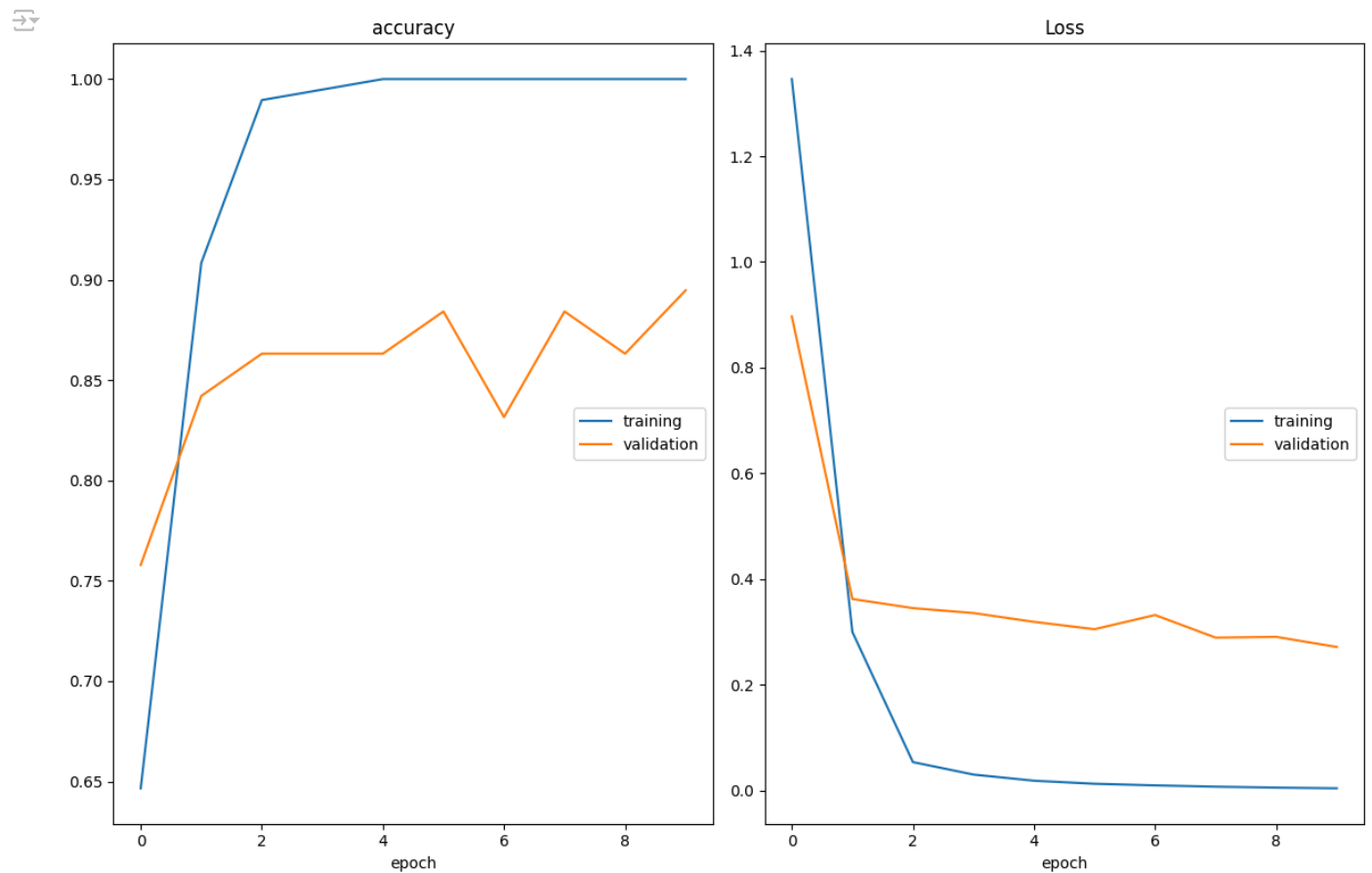
- ✓ Treino do Modelo sem Data Augmentation

[illegible]

```

33     callbacks=[checkpointer, PlotLossesKeras()],
34     #callbacks=[early_stop, checkpointer, PlotLossesKeras()],
35     verbose=False)

```



```

accuracy
  training      (min: 0.647, max: 1.000, cur: 1.000)
  validation    (min: 0.758, max: 0.895, cur: 0.895)
Loss
  training      (min: 0.004, max: 1.346, cur: 0.004)
  validation    (min: 0.271, max: 0.897, cur: 0.271)
CPU times: user 29.3 s, sys: 6.4 s, total: 35.7 s
Wall time: 44.5 s

```

✓ Aplicar Modelo treinado na base de teste

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

```

1 ##Aplicar Modelo treinado na base de teste
2 ### Calcular Métrica de Sensibilidade,, Especificidade e F1-Score com base na matriz de confusão
3 from sklearn.metrics import accuracy_score
4 import numpy as np
5 import seaborn as sns
6 import matplotlib.pyplot as plt
7 from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report, confusion_matrix
8
9 model_resnet50_noaug.load_weights('img_model_resnet50_noaug.weights.best.keras') ## Inicializa o peso com o melhor treino
10
11 true_classes = np.argmax(np.concatenate([y for x, y in test_noaug], axis=0), axis=1)
12 class_indices = {0: '0', 1: '1', 2: '2', 3: '3'}
13
14 predictions = model_resnet50_noaug.predict(test_noaug)
15 predicted_classes = np.argmax(predictions, axis=1)
16
17 accuracy = accuracy_score(true_classes, predicted_classes)
18 precision = precision_score(true_classes, predicted_classes, average='weighted')
19 f1 = f1_score(true_classes, predicted_classes, average='weighted')
20 print("Acurácia Modelo Resnet50 sem Data Augmentation treinado {:.2f}%".format(accuracy * 100))
21 print("Precisão Modelo Resnet50 sem Data Augmentation treinado {:.2f}%".format(precision * 100))

```

```

22 print("F1_Score Modelo Resnet50 sem Data Augmentation treinado {:.2f}%".format(f1 * 100))
23
24 # Get the names of the ten classes
25 class_names = test_noaug.class_names
26
27 #TODO: Imprimir Matriz de confusão
28 plot = confusion_matrix(true_classes, predicted_classes)
29 sns.heatmap(plot, annot=True, fmt='d', xticklabels=class_names, yticklabels=class_names)
30
31 2/2 [=====] - 1s 230ms/step
32 Acurácia Modelo Resnet50 sem Data Augmentation treinado 78.45%
33 Precisão Modelo Resnet50 sem Data Augmentation treinado 80.38%
34 F1_Score Modelo Resnet50 sem Data Augmentation treinado 76.36%
35 <Axes: >

```

