iaa011-vc-trabalho

October 23, 2025

1 Trabalho IAA011 - Visão Computacional

1.1 Equipe 03

- Gustavo Costa de Souza
- Marcos Vinicius de Melo
- Marcus Eneas Silveira Galvao do Rio Apa II
- Patrícia Verdugo Pascoal
- Rodrigo de Araujo
- William de Souza Alencar

2 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 o resultado e a métrica utilizada

[1]:
```
import os
import shutil
from collections import defaultdict
from sklearn.model_selection import train_test_split
```

```
import cv2
import numpy as np
from skimage.feature import local_binary_pattern
from tqdm import tqdm
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix
```

2.0.1 1. Carregue a base de dados de Treino.

[2]:
```
!tar -xf Train_Warwick.zip -C train
```

[3]:
```
BASE_DIR_TRAIN = 'train/Train_4cls_amostra'
OUTPUT_DIR_TRAIN = 'train_split'

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

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

[4]:
```
%%time
for split in ['train', 'val']:
    for cls in classes:
        os.makedirs(os.path.join(OUTPUT_DIR_TRAIN, split, cls), exist_ok=True)

def copy_images(image_paths, split_name, cls):
    dest_dir = os.path.join(OUTPUT_DIR_TRAIN, split_name, cls)
    for img_path in image_paths:
        shutil.copy(img_path, dest_dir)

for cls in classes:
    print(f"\n Processando classe: {cls}")

    class_dir = os.path.join(BASE_DIR_TRAIN, cls)
    patient_images = defaultdict(list)

    for filename in os.listdir(class_dir):
        if filename.endswith('.png'):
            patient_id = filename.split('_')[0]
            img_path = os.path.join(class_dir, filename)
            patient_images[patient_id].append(img_path)

    patients = list(patient_images.keys())

    train_patients, val_patients = train_test_split(
        patients, test_size=0.2, random_state=42
```

```
    )

    train_images = [img for p in train_patients for img in patient_images[p]]
    val_images = [img for p in val_patients for img in patient_images[p]]

    copy_images(train_images, 'train', cls)
    copy_images(val_images, 'val', cls)

    print(f" - Pacientes de treino: {len(train_patients)}")
    print(f" - Pacientes de validação: {len(val_patients)}")
    print(f" - Total de imagens: {len(train_images)} treino, {len(val_images)}␣
↪validação")

print("\n Separação concluída com sucesso!")
```

```
 Processando classe: 0
 - Pacientes de treino: 4
 - Pacientes de validação: 1
 - Total de imagens: 116 treino, 30 validação

 Processando classe: 1
 - Pacientes de treino: 4
 - Pacientes de validação: 1
 - Total de imagens: 117 treino, 30 validação

 Processando classe: 2
 - Pacientes de treino: 4
 - Pacientes de validação: 1
 - Total de imagens: 120 treino, 30 validação

 Processando classe: 3
 - Pacientes de treino: 4
 - Pacientes de validação: 1
 - Total de imagens: 120 treino, 30 validação

 Separação concluída com sucesso!
CPU times: total: 656 ms
Wall time: 740 ms
```

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

Extrator LBP

[5]:
```
%%time
RADIUS = 1
N_POINTS = 8 * RADIUS
METHOD = 'uniform'
```

```
def extract_lbp_hist_features(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    lbp = local_binary_pattern(image, N_POINTS, RADIUS, METHOD)

    n_bins = int(lbp.max() + 1)
    hist, _ = np.histogram(lbp.ravel(), bins=n_bins, range=(0, n_bins),␣
↪density=True)
    return hist


def extract_lbp(dir_path):
    features = []
    labels = []
    for cls in classes:
        class_dir = os.path.join(dir_path, cls)
        for filename in tqdm(os.listdir(class_dir), desc=f"Extraindo LBP da␣
↪classe ({cls})"):
            img_path = os.path.join(class_dir, filename)
            hist = extract_lbp_hist_features(img_path)
            features.append(hist)
            labels.append(cls)
    return features, labels

features, labels = extract_lbp(BASE_DIR_TRAIN)

features = np.array(features)
labels = np.array(labels)

print("Formato do vetor de características LBP:", features.shape)
print("Exemplo de histograma LBP:", features[0])

# Exporta os histogramas LBP para o CSV
df_lbp = pd.DataFrame(features)
df_lbp['label'] = labels

output_csv_lbp_train = 'lbp_features_train.csv'
df_lbp.to_csv(output_csv_lbp_train, index=False)

print(f"Arquivo CSV gerado com sucesso: {output_csv_lbp_train}")
print(f"Dimensões: {df_lbp.shape[0]} amostras x {df_lbp.shape[1]} colunas")
```

```
Extraindo LBP da classe (0):
100%|                            | 146/146 [00:02<00:00,
64.62it/s]
Extraindo LBP da classe (1):
```

```
100%|                              | 147/147 [00:02<00:00,
63.09it/s]
Extraindo LBP da classe (2):
100%|                              | 150/150 [00:02<00:00,
57.39it/s]
Extraindo LBP da classe (3):
100%|                              | 150/150 [00:02<00:00,
64.36it/s]
```

```
Formato do vetor de características LBP: (593, 10)
Exemplo de histograma LBP: [0.013488 0.033344 0.04144  0.159392 0.356192
0.212496 0.071904 0.0356
 0.03176  0.044384]
Arquivo CSV gerado com sucesso: lbp_features_train.csv
Dimensões: 593 amostras x 11 colunas
CPU times: total: 9.3 s
Wall time: 9.61 s
```

Extrator CNN VGG16

[6]:
```
%%time
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model

# Carrega o modelo VGG16 pré-treinado no ImageNet, sem a camada de classificação
vgg16 = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

model_vgg16 = Model(inputs=vgg16.input, outputs=vgg16.output)

print("Modelo VGG16 carregado. Dimensões da última camada: ", model_vgg16.
 ↪output_shape)

def extract_vgg16_features(img_path):
    # Carrega imagem e redimensiona para 224x224 (padrão VGG16)
    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)

    # Extrai características
    features = model_vgg16.predict(img_array, verbose=0)
    return features.flatten()

def extract_vgg(dir_path):
    features_vgg = []
    labels_vgg = []

    for cls in classes:
```

```
        class_dir = os.path.join(dir_path, cls)
        for filename in tqdm(os.listdir(class_dir), desc=f"Extraindo VGG16 da␣
↪classe ({cls})"):
            img_path = os.path.join(class_dir, filename)
            vec = extract_vgg16_features(img_path)
            features_vgg.append(vec)
            labels_vgg.append(cls)
    return features_vgg, labels_vgg

features_vgg, labels_vgg = extract_vgg(BASE_DIR_TRAIN)
print("Número de imagens:", len(features_vgg))

# salva no CSV
features_vgg = np.array(features_vgg)
labels_vgg = np.array(labels_vgg)
df_vgg = pd.DataFrame(features_vgg)
df_vgg['label'] = labels_vgg

output_csv = 'vgg16_features_train.csv'
df_vgg.to_csv(output_csv, index=False)

print(f"\nExtração concluída. Arquivo salvo em: {output_csv}")
print(f"Dimensões: {df_vgg.shape[0]} amostras x {df_vgg.shape[1]} colunas")
```

```
Modelo VGG16 carregado. Dimensões da última camada:  (None, 7, 7, 512)

Extraindo VGG16 da classe (0):
100%|                          | 146/146 [00:40<00:00,
3.62it/s]
Extraindo VGG16 da classe (1):
100%|                          | 147/147 [00:39<00:00,
3.70it/s]
Extraindo VGG16 da classe (2):
100%|                          | 150/150 [00:39<00:00,
3.81it/s]
Extraindo VGG16 da classe (3):
100%|                          | 150/150 [00:39<00:00,
3.80it/s]

Número de imagens: 593

Extração concluída. Arquivo salvo em: vgg16_features_train.csv
Dimensões: 593 amostras x 25089 colunas
CPU times: total: 12min 34s
Wall time: 4min 4s
```

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

Treinando os modelos utilizando as características LBP

[7]:
```
%%time
df_lbp = pd.read_csv('lbp_features_train.csv')

print("Formato do DataFrame LBP:", df_lbp.shape)
#print(df_lbp.head(2))

X = df_lbp.drop(columns=['label']).to_numpy(dtype=np.float32)
y = df_lbp['label'].to_numpy(dtype=np.int32)

print("X shape:", X.shape)
print("y shape:", y.shape)
```

```
Formato do DataFrame LBP: (593, 11)
X shape: (593, 10)
y shape: (593,)
CPU times: total: 0 ns
Wall time: 43.6 ms
```

SVM

[8]:
```
%%time

from sklearn.svm import SVC

# Cria e treina o classificador SVM
def svm(X, y):
    svm = SVC(kernel='rbf', gamma='scale', C=1, verbose=True, random_state=42,
  ↪class_weight='balanced')
    svm.fit(X, y)
    print("Modelo SVM treinado")
    return svm

svm_lbp = svm(X, y)

print("Número de vetores de suporte por classe:", svm_lbp.n_support_)
```

```
[LibSVM]Modelo SVM treinado
Número de vetores de suporte por classe: [146 142 150  79]
CPU times: total: 78.1 ms
Wall time: 7.17 s
```

Random Forest

[9]:
```
%%time
from sklearn.ensemble import RandomForestClassifier

def rf(X, y):
```

```
    rf = RandomForestClassifier(n_estimators=100, random_state=42, verbose=True␣
 ↪)
    rf.fit(X, y)
    print("Modelo RF treinado")
    return rf

rf_clf_lbp = rf(X, y)
```

```
Modelo RF treinado
CPU times: total: 438 ms
Wall time: 13.9 s
```

```
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.1s
```

RNA

[10]:
```
%%time
from sklearn.neural_network import MLPClassifier

def rna(X, y):
    rna = MLPClassifier(hidden_layer_sizes=(15,), activation='relu', alpha=0.1,␣
 ↪solver='adam',
                        max_iter=3000, random_state=42, verbose=False)
    rna.fit(X, y)
    print("Modelo RNA treinado")
    return rna

rna_lbp = rna(X, y)
```

```
Modelo RNA treinado
CPU times: total: 15.8 s
Wall time: 16 s
```

Treinando os modelos utilizando as características VGG16

[11]:
```
%%time
df_vgg = pd.read_csv('vgg16_features_train.csv')

print("Formato do DataFrame VGG:", df_vgg.shape)

X = df_vgg.drop(columns=['label']).to_numpy(dtype=np.float32)
y = df_vgg['label'].to_numpy(dtype=np.int32)

print("X shape:", X.shape)
print("y shape:", y.shape)
```

```
Formato do DataFrame VGG: (593, 25089)
X shape: (593, 25088)
y shape: (593,)
CPU times: total: 8.11 s
Wall time: 8.34 s
```

[12]:
```
%%time
svm_vgg = svm(X, y)

print("Número de vetores de suporte por classe:", svm_vgg.n_support_)
```

```
[LibSVM]Modelo SVM treinado
Número de vetores de suporte por classe: [139 145 143 136]
CPU times: total: 2h 23min 40s
Wall time: 19min 31s
```

[13]:
```
%%time
rf_clf_vgg = rf(X, y)
```

```
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.3s
```

```
Modelo RF treinado
CPU times: total: 750 ms
Wall time: 735 ms
```

[14]:
```
%%time
rna_vgg = rna(X, y)
```

```
Modelo RNA treinado
CPU times: total: 52.7 s
Wall time: 22.7 s
```

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

[15]:
```
!tar -xf Test_Warwick.zip -C test
```

[16]:
```
%%time
BASE_DIR_TEST = 'test/Test_4cl_amostra'

features = []
labels = []
features, labels = extract_lbp(BASE_DIR_TEST)

features = np.array(features, dtype=np.float32)  # vetor 1D por imagem
labels = np.array(labels)

print("Número de imagens:", len(features))

## exporta para csv
df_lbp = pd.DataFrame(features)
df_lbp['label'] = labels

#print(df_lbp.head(2))

output_csv = 'lbp_features_test.csv'
```

```
df_lbp.to_csv(output_csv, index=False)

print(f"Arquivo CSV gerado com sucesso: {output_csv}")
print(f"Dimensões: {df_lbp.shape[0]} amostras x {df_lbp.shape[1]} colunas")
```

```
Extraindo LBP da classe (0):
100%|                              | 101/101 [00:00<00:00,
148.61it/s]
Extraindo LBP da classe (1):
100%|                               | 90/90 [00:00<00:00,
148.43it/s]
Extraindo LBP da classe (2):
100%|                               | 90/90 [00:00<00:00,
145.87it/s]
Extraindo LBP da classe (3):
100%|                               | 90/90 [00:00<00:00,
150.20it/s]

Número de imagens: 371
Arquivo CSV gerado com sucesso: lbp_features_test.csv
Dimensões: 371 amostras x 11 colunas
CPU times: total: 2.47 s
Wall time: 2.51 s
```

[17]:
```
%%time
df_lbp_test = pd.read_csv('lbp_features_test.csv')

print("Formato do DataFrame de teste:", df_lbp_test.shape)
#print(df_lbp_test.head(2))

X_test = df_lbp.drop(columns=['label']).to_numpy(dtype=np.float64)
y_test = df_lbp['label'].to_numpy(dtype=np.int32)

print("X shape:", X_test.shape)
print("y shape:", y_test.shape)
```

```
Formato do DataFrame de teste: (371, 11)
X shape: (371, 10)
y shape: (371,)
CPU times: total: 15.6 ms
Wall time: 7.89 ms
```

[18]:
```
%%time
y_pred_svm_lbp = svm_lbp.predict(X_test)
y_pred_rf_lbp = rf_clf_lbp.predict(X_test)
y_pred_rna_lbp = rna_lbp.predict(X_test)
```

```
print("SVM LBP- Acurácia:", accuracy_score(y_test, y_pred_svm_lbp))
print("Rando Forest LBP- Acurácia:", accuracy_score(y_test, y_pred_rf_lbp))
print("RNA LBP- Acurácia:", accuracy_score(y_test, y_pred_rna_lbp))
```

```
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
```

```
SVM LBP- Acurácia: 0.555256064690027
Rando Forest LBP- Acurácia: 0.568733153638814
RNA LBP- Acurácia: 0.5876010781671159
CPU times: total: 15.6 ms
Wall time: 14.6 ms
```

VGG

[19]:
```
features = []
labels = []
features, labels = extract_vgg(BASE_DIR_TEST)

features = np.array(features, dtype=np.float32)  # vetor 1D por imagem
labels = np.array(labels)

print("Número de imagens:", len(features))

## exporta para csv
df_vgg = pd.DataFrame(features)
df_vgg['label'] = labels

#print(df_vgg.head(2))

output_csv = 'vgg_features_test.csv'
df_vgg.to_csv(output_csv, index=False)

print(f"Arquivo CSV gerado com sucesso: {output_csv}")
print(f"Dimensões: {df_vgg.shape[0]} amostras x {df_vgg.shape[1]} colunas")
```

```
Extraindo VGG16 da classe (0):
100%|                          | 101/101 [00:10<00:00,
10.00it/s]
Extraindo VGG16 da classe (1):
100%|                          | 90/90 [00:09<00:00,
9.93it/s]
Extraindo VGG16 da classe (2):
100%|                          | 90/90 [00:09<00:00,
9.91it/s]
Extraindo VGG16 da classe (3):
100%|                          | 90/90 [00:09<00:00,
9.90it/s]
```

```
Número de imagens: 371
Arquivo CSV gerado com sucesso: vgg_features_test.csv
```

```
Dimensões: 371 amostras x 25089 colunas
```

[20]:
```python
%%time
df_vgg_test = pd.read_csv('vgg_features_test.csv')

print("Formato do DataFrame VGG de teste:", df_vgg_test.shape)
#print(df_vgg_test.head(2))

X_vgg_test = df_vgg_test.drop(columns=['label']).to_numpy(dtype=np.float32)
y_vgg_test = df_vgg_test['label'].to_numpy(dtype=np.int32)

print("X shape:", X_vgg_test.shape)
print("y shape:", y_vgg_test.shape)
```

```
Formato do DataFrame VGG de teste: (371, 25089)
X shape: (371, 25088)
y shape: (371,)
CPU times: total: 1.45 s
Wall time: 1.47 s
```

[21]:
```python
%%time
y_pred_svm_vgg = svm_vgg.predict(X_vgg_test)
y_pred_rf_vgg = rf_clf_vgg.predict(X_vgg_test)
y_pred_rna_vgg = rna_vgg.predict(X_vgg_test)

print("SVM VGG- Acurácia:", accuracy_score(y_vgg_test, y_pred_svm_vgg))
print("Rando Forest VGG- Acurácia:", accuracy_score(y_vgg_test, y_pred_rf_vgg))
print("RNA VGG- Acurácia:", accuracy_score(y_vgg_test, y_pred_rna_vgg))
```

```
SVM VGG- Acurácia: 0.8140161725067385
Rando Forest VGG- Acurácia: 0.7574123989218329
RNA VGG- Acurácia: 0.5390835579514824
CPU times: total: 4min 23s
Wall time: 21.5 s
```

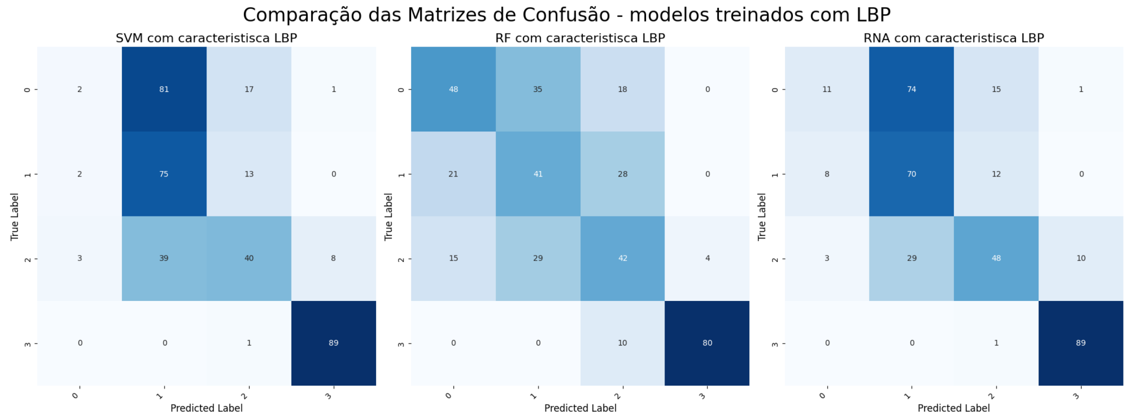
```
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
```

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

[22]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

def plot_heatmap(y_true, y_pred, class_names, ax, title):
    cm = confusion_matrix(y_true, y_pred)
    sns.heatmap(
        cm,
```

```
        annot=True,
        square=True,
        xticklabels=class_names,
        yticklabels=class_names,
        fmt='d',
        cmap=plt.cm.Blues,
        cbar=False,
        ax=ax
    )
    ax.set_title(title, fontsize=16)
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha="right")
    ax.set_ylabel('True Label', fontsize=12)
    ax.set_xlabel('Predicted Label', fontsize=12)
```

[23]:
```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20, 10))

plot_heatmap(y_test, y_pred_svm_lbp, classes, ax1, title="SVM com␣
 ↪caracteristisca LBP")
plot_heatmap(y_test, y_pred_rf_lbp, classes, ax2, title="RF com caracteristisca␣
 ↪LBP")
plot_heatmap(y_test, y_pred_rna_lbp, classes, ax3, title="RNA com␣
 ↪caracteristisca LBP")

fig.suptitle("Comparação das Matrizes de Confusão - modelos treinados com LBP",␣
 ↪fontsize=24)
fig.tight_layout()
fig.subplots_adjust(top=1.2)
plt.show()

print("Métricas SVM LBP")
print(classification_report(y_test, y_pred_svm_lbp, digits=3))
print("Métricas Randon Forest LBP")
print(classification_report(y_test, y_pred_rf_lbp, digits=3))
print("Métricas Randon RNA LBP")
print(classification_report(y_test, y_pred_rna_lbp, digits=3))
```



Métricas SVM LBP

	precision	recall	f1-score	support
0	0.286	0.020	0.037	101
1	0.385	0.833	0.526	90
2	0.563	0.444	0.497	90
3	0.908	0.989	0.947	90
accuracy			0.555	371
macro avg	0.535	0.572	0.502	371
weighted avg	0.528	0.555	0.488	371

Métricas Randon Forest LBP

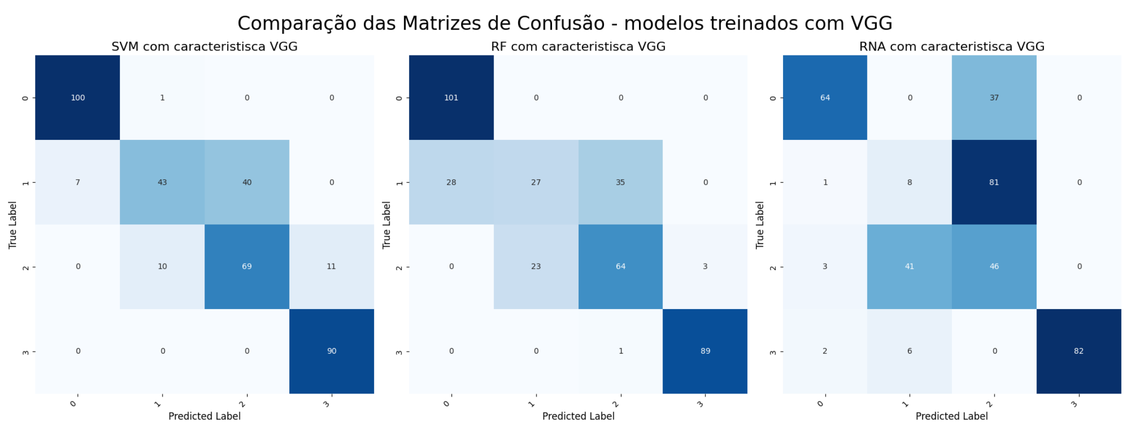
	precision	recall	f1-score	support
0	0.571	0.475	0.519	101
1	0.390	0.456	0.421	90
2	0.429	0.467	0.447	90
3	0.952	0.889	0.920	90
accuracy			0.569	371
macro avg	0.586	0.572	0.576	371
weighted avg	0.585	0.569	0.575	371

Métricas Randon RNA LBP

	precision	recall	f1-score	support
0	0.500	0.109	0.179	101
1	0.405	0.778	0.532	90
2	0.632	0.533	0.578	90
3	0.890	0.989	0.937	90
accuracy			0.588	371

```
   macro avg      0.607     0.602     0.557       371
weighted avg      0.603     0.588     0.545       371
```

[24]:
```python
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20, 10))

plot_heatmap(y_vgg_test, y_pred_svm_vgg, classes, ax1, title="SVM com␣
 ↪caracteristisca VGG")
plot_heatmap(y_vgg_test, y_pred_rf_vgg, classes, ax2, title="RF com␣
 ↪caracteristisca VGG")
plot_heatmap(y_vgg_test, y_pred_rna_vgg, classes, ax3, title="RNA com␣
 ↪caracteristisca VGG")

fig.suptitle("Comparação das Matrizes de Confusão - modelos treinados com VGG",␣
 ↪fontsize=24)
fig.tight_layout()
fig.subplots_adjust(top=1.2)
plt.show()

print("Métricas SVM VGG")
print(classification_report(y_vgg_test, y_pred_svm_vgg, digits=3))
print("Métricas Randon Forest VGG")
print(classification_report(y_vgg_test, y_pred_rf_vgg, digits=3))
print("Métricas Randon RNA VGG")
print(classification_report(y_vgg_test, y_pred_rna_vgg, digits=3))
```




```
Métricas SVM VGG
              precision    recall  f1-score   support

           0      0.935     0.990     0.962       101
           1      0.796     0.478     0.597        90
           2      0.633     0.767     0.693        90
           3      0.891     1.000     0.942        90
```

```
    accuracy                          0.814       371
   macro avg      0.814     0.809     0.799       371
weighted avg      0.817     0.814     0.803       371

Métricas Randon Forest VGG
              precision    recall  f1-score   support

           0      0.783     1.000     0.878       101
           1      0.540     0.300     0.386        90
           2      0.640     0.711     0.674        90
           3      0.967     0.989     0.978        90

    accuracy                          0.757       371
   macro avg      0.733     0.750     0.729       371
weighted avg      0.734     0.757     0.733       371

Métricas Randon RNA VGG
              precision    recall  f1-score   support

           0      0.914     0.634     0.749       101
           1      0.145     0.089     0.110        90
           2      0.280     0.511     0.362        90
           3      1.000     0.911     0.953        90

    accuracy                          0.539       371
   macro avg      0.585     0.536     0.544       371
weighted avg      0.595     0.539     0.550       371
```

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

O melhor modelo é o **SVM** usando **features VGG**, pois apresenta o maior **F1-score ponderado (0.803)** e também a maior **acurácia (0.814)**entre todos os modelos testados.

3 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 o resultado e a métrica utilizada

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

[107]:
```
%%time

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input
from keras.layers import Dense, Dropout, Flatten
from keras.models import Model

# Data augmentation
IMAGE_BASE_DIR = 'train_split'
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,
channel_shift_range=25.0,
zoom_range=0.1,
shear_range=0.15,
                                     preprocessing_function=preprocess_input)

test_generator = ImageDataGenerator(preprocessing_function=preprocess_input)

BATCH_SIZE = 32 # quantidade de imagens criadas em cada ciclo

print('Data augmentation - train')
traingen = train_generator.flow_from_directory(IMAGE_BASE_DIR + '/train',
                                               target_size=(224, 224),
                                               batch_size=BATCH_SIZE,
                                               class_mode='categorical',
                                               classes=classes,
                                               #subset='training',
                                               shuffle=False,
                                               seed=42)

print('Data augmentation - validation')
validgen = train_generator.flow_from_directory(IMAGE_BASE_DIR + '/val',
                                               target_size=(224, 224),
                                               batch_size=BATCH_SIZE,
                                               class_mode='categorical',
```

```
                                                  classes=classes,
                                                  #subset='validation',
                                                  shuffle=False,
                                                  seed=42)

print('Data augmentation - test')
testgen = test_generator.flow_from_directory('test/Test_4cl_amostra',
                                                target_size=(224, 224),
                                                batch_size=BATCH_SIZE,
                                                class_mode=None,
                                                classes=classes,
                                                shuffle=False,
                                                seed=42)
```

```
Data augmentation - train
Found 473 images belonging to 4 classes.
Data augmentation - validation
Found 120 images belonging to 4 classes.
Data augmentation - test
Found 371 images belonging to 4 classes.
CPU times: total: 93.8 ms
Wall time: 95.9 ms
```

[108]:
```
# Sem data augmentation

train_generator_noda = ImageDataGenerator(
                                    #validation_split=0.2,
                                    preprocessing_function=preprocess_input)

print('No data augmentation - train')
traingen_noda = train_generator_noda.flow_from_directory(IMAGE_BASE_DIR + '/
 ↪train',
                                                  target_size=(224, 224),
                                                  batch_size=BATCH_SIZE,
                                                  class_mode='categorical',
                                                  classes=classes,
                                                  #subset='training',
                                                  shuffle=False,
                                                  seed=42)

print('No Data augmentation - validation')
validgen_noda = train_generator_noda.flow_from_directory(IMAGE_BASE_DIR + '/
 ↪val',
                                                  target_size=(224, 224),
                                                  batch_size=BATCH_SIZE,
                                                  class_mode='categorical',
                                                  classes=classes,
```

```
                                                    #subset='validation',
                                                    shuffle=False,
                                                    seed=42)

print('No Data augmentation - test')
testgen_noda = train_generator_noda.flow_from_directory('test/Test_4cl_amostra',
                                                    target_size=(224, 224),
                                                    batch_size=BATCH_SIZE,
                                                    class_mode=None,
                                                    classes=classes,
                                                    shuffle=False,
                                                    seed=42)
```

```
No data augmentation - train
Found 473 images belonging to 4 classes.
No Data augmentation - validation
Found 120 images belonging to 4 classes.
No Data augmentation - test
Found 371 images belonging to 4 classes.
```

3.1.1 Resnet50 com Transfer Learning - sem data augmentation

[109]:
```
# A opção include_top=False não inclui as camadas de aprendizado da rede␣
 ↪original
# Utiliza os pesos treinados na base imagenet
resnet_tl = ResNet50(input_shape=(224,224,3), weights='imagenet',␣
 ↪include_top=False)

# não treinar os pesos existentes
for layer in resnet_tl.layers:
  layer.trainable = False
```

[110]:
```
# A saída da resnet será a entrada da camada criada
x_tl = Flatten()(resnet_tl.output)

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

# Criação do Objeto Modelo (a parte da resnet + as camadas Fully connected␣
 ↪criadas)
model_resnet_tl_no_da = Model(inputs=resnet_tl.input, outputs=prediction)
model_resnet_tl_da = Model(inputs=resnet_tl.input, outputs=prediction)
```

[111]:
```
model_resnet_tl_no_da.summary()
```

Model: "functional_21"

```
Layer (type)                  Output Shape                  Param # ␣
↪Connected to

input_layer_11 (InputLayer)   (None, 224, 224, 3)                 0   - ␣
↪

conv1_pad (ZeroPadding2D)     (None, 230, 230, 3)                 0 ␣
↪input_layer_11[0][0]

conv1_conv (Conv2D)           (None, 112, 112, 64)            9,472 ␣
↪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)              4,160 ␣
↪pool1_pool[0][0]

conv2_block1_1_bn             (None, 56, 56, 64)                256 ␣
↪conv2_block1_1_conv[0][0]
(BatchNormalization)                                                  ␣
↪

conv2_block1_1_relu           (None, 56, 56, 64)                  0 ␣
↪conv2_block1_1_bn[0][0]
(Activation)                                                          ␣
↪

conv2_block1_2_conv (Conv2D)  (None, 56, 56, 64)             36,928 ␣
↪conv2_block1_1_relu[0][0]

conv2_block1_2_bn             (None, 56, 56, 64)                256 ␣
↪conv2_block1_2_conv[0][0]
(BatchNormalization)                                                  ␣
↪
```

```
conv2_block1_2_relu           (None, 56, 56, 64)                 0 ␣
↪conv2_block1_2_bn[0][0]
(Activation)                                                        ␣
↪

conv2_block1_0_conv (Conv2D)  (None, 56, 56, 256)           16,640 ␣
↪pool1_pool[0][0]

conv2_block1_3_conv (Conv2D)  (None, 56, 56, 256)           16,640 ␣
↪conv2_block1_2_relu[0][0]

conv2_block1_0_bn             (None, 56, 56, 256)            1,024 ␣
↪conv2_block1_0_conv[0][0]
(BatchNormalization)                                                ␣
↪

conv2_block1_3_bn             (None, 56, 56, 256)            1,024 ␣
↪conv2_block1_3_conv[0][0]
(BatchNormalization)                                                ␣
↪

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)            16,448 ␣
↪conv2_block1_out[0][0]

conv2_block2_1_bn             (None, 56, 56, 64)               256 ␣
↪conv2_block2_1_conv[0][0]
(BatchNormalization)                                                ␣
↪

conv2_block2_1_relu           (None, 56, 56, 64)                 0 ␣
↪conv2_block2_1_bn[0][0]
(Activation)                                                        ␣
↪

conv2_block2_2_conv (Conv2D)  (None, 56, 56, 64)            36,928 ␣
↪conv2_block2_1_relu[0][0]
```

```
conv2_block2_2_bn              (None, 56, 56, 64)                 256 ␣
↪conv2_block2_2_conv[0][0]
(BatchNormalization)                                                    ␣
↪

conv2_block2_2_relu            (None, 56, 56, 64)                   0 ␣
↪conv2_block2_2_bn[0][0]
(Activation)                                                            ␣
↪

conv2_block2_3_conv (Conv2D)   (None, 56, 56, 256)             16,640 ␣
↪conv2_block2_2_relu[0][0]

conv2_block2_3_bn              (None, 56, 56, 256)              1,024 ␣
↪conv2_block2_3_conv[0][0]
(BatchNormalization)                                                    ␣
↪

conv2_block2_add (Add)         (None, 56, 56, 256)                  0 ␣
↪conv2_block1_out[0][0],
                                                                      ␣
↪conv2_block2_3_bn[0][0]

conv2_block2_out (Activation)  (None, 56, 56, 256)                  0 ␣
↪conv2_block2_add[0][0]

conv2_block3_1_conv (Conv2D)   (None, 56, 56, 64)              16,448 ␣
↪conv2_block2_out[0][0]

conv2_block3_1_bn              (None, 56, 56, 64)                 256 ␣
↪conv2_block3_1_conv[0][0]
(BatchNormalization)                                                    ␣
↪

conv2_block3_1_relu            (None, 56, 56, 64)                   0 ␣
↪conv2_block3_1_bn[0][0]
(Activation)                                                            ␣
↪

conv2_block3_2_conv (Conv2D)   (None, 56, 56, 64)              36,928 ␣
↪conv2_block3_1_relu[0][0]

conv2_block3_2_bn              (None, 56, 56, 64)                 256 ␣
↪conv2_block3_2_conv[0][0]
(BatchNormalization)                                                    ␣
↪
```

```
conv2_block3_2_relu          (None, 56, 56, 64)                 0 ␣
↪conv2_block3_2_bn[0][0]
(Activation)                                                         ␣
↪

conv2_block3_3_conv (Conv2D) (None, 56, 56, 256)           16,640 ␣
↪conv2_block3_2_relu[0][0]

conv2_block3_3_bn            (None, 56, 56, 256)            1,024 ␣
↪conv2_block3_3_conv[0][0]
(BatchNormalization)                                                 ␣
↪

conv2_block3_add (Add)       (None, 56, 56, 256)                0 ␣
↪conv2_block2_out[0][0],
                                                                  ␣
↪conv2_block3_3_bn[0][0]

conv2_block3_out (Activation) (None, 56, 56, 256)               0 ␣
↪conv2_block3_add[0][0]

conv3_block1_1_conv (Conv2D) (None, 28, 28, 128)           32,896 ␣
↪conv2_block3_out[0][0]

conv3_block1_1_bn            (None, 28, 28, 128)              512 ␣
↪conv3_block1_1_conv[0][0]
(BatchNormalization)                                                 ␣
↪

conv3_block1_1_relu          (None, 28, 28, 128)                0 ␣
↪conv3_block1_1_bn[0][0]
(Activation)                                                         ␣
↪

conv3_block1_2_conv (Conv2D) (None, 28, 28, 128)          147,584 ␣
↪conv3_block1_1_relu[0][0]

conv3_block1_2_bn            (None, 28, 28, 128)              512 ␣
↪conv3_block1_2_conv[0][0]
(BatchNormalization)                                                 ␣
↪

conv3_block1_2_relu          (None, 28, 28, 128)                0 ␣
↪conv3_block1_2_bn[0][0]
```

```
(Activation)                                                                      ␣
↪

conv3_block1_0_conv (Conv2D)    (None, 28, 28, 512)             131,584  ␣
↪conv2_block3_out[0][0]

conv3_block1_3_conv (Conv2D)    (None, 28, 28, 512)              66,048  ␣
↪conv3_block1_2_relu[0][0]

conv3_block1_0_bn               (None, 28, 28, 512)               2,048  ␣
↪conv3_block1_0_conv[0][0]
(BatchNormalization)                                                              ␣
↪

conv3_block1_3_bn               (None, 28, 28, 512)               2,048  ␣
↪conv3_block1_3_conv[0][0]
(BatchNormalization)                                                              ␣
↪

conv3_block1_add (Add)          (None, 28, 28, 512)                   0  ␣
↪conv3_block1_0_bn[0][0],
                                                                         ␣
↪conv3_block1_3_bn[0][0]

conv3_block1_out (Activation)   (None, 28, 28, 512)                   0  ␣
↪conv3_block1_add[0][0]

conv3_block2_1_conv (Conv2D)    (None, 28, 28, 128)              65,664  ␣
↪conv3_block1_out[0][0]

conv3_block2_1_bn               (None, 28, 28, 128)                 512  ␣
↪conv3_block2_1_conv[0][0]
(BatchNormalization)                                                              ␣
↪

conv3_block2_1_relu             (None, 28, 28, 128)                   0  ␣
↪conv3_block2_1_bn[0][0]
(Activation)                                                                      ␣
↪

conv3_block2_2_conv (Conv2D)    (None, 28, 28, 128)             147,584  ␣
↪conv3_block2_1_relu[0][0]

conv3_block2_2_bn               (None, 28, 28, 128)                 512  ␣
↪conv3_block2_2_conv[0][0]
```

```
(BatchNormalization)                                                    ␣
↪

conv3_block2_2_relu          (None, 28, 28, 128)                  0  ␣
↪conv3_block2_2_bn[0][0]
(Activation)                                                            ␣
↪

conv3_block2_3_conv (Conv2D) (None, 28, 28, 512)             66,048  ␣
↪conv3_block2_2_relu[0][0]

conv3_block2_3_bn            (None, 28, 28, 512)              2,048  ␣
↪conv3_block2_3_conv[0][0]
(BatchNormalization)                                                    ␣
↪

conv3_block2_add (Add)       (None, 28, 28, 512)                  0  ␣
↪conv3_block1_out[0][0],
                                                                     ␣
↪conv3_block2_3_bn[0][0]

conv3_block2_out (Activation) (None, 28, 28, 512)                 0  ␣
↪conv3_block2_add[0][0]

conv3_block3_1_conv (Conv2D) (None, 28, 28, 128)             65,664  ␣
↪conv3_block2_out[0][0]

conv3_block3_1_bn            (None, 28, 28, 128)                512  ␣
↪conv3_block3_1_conv[0][0]
(BatchNormalization)                                                    ␣
↪

conv3_block3_1_relu          (None, 28, 28, 128)                  0  ␣
↪conv3_block3_1_bn[0][0]
(Activation)                                                            ␣
↪

conv3_block3_2_conv (Conv2D) (None, 28, 28, 128)            147,584  ␣
↪conv3_block3_1_relu[0][0]

conv3_block3_2_bn            (None, 28, 28, 128)                512  ␣
↪conv3_block3_2_conv[0][0]
(BatchNormalization)                                                    ␣
↪
```

```
conv3_block3_2_relu          (None, 28, 28, 128)                   0 ␣
↪conv3_block3_2_bn[0][0]
(Activation)                                                            ␣
↪

conv3_block3_3_conv (Conv2D)   (None, 28, 28, 512)              66,048 ␣
↪conv3_block3_2_relu[0][0]

conv3_block3_3_bn            (None, 28, 28, 512)               2,048 ␣
↪conv3_block3_3_conv[0][0]
(BatchNormalization)                                                    ␣
↪

conv3_block3_add (Add)       (None, 28, 28, 512)                   0 ␣
↪conv3_block2_out[0][0],
                                                                     ␣
↪conv3_block3_3_bn[0][0]

conv3_block3_out (Activation)  (None, 28, 28, 512)                 0 ␣
↪conv3_block3_add[0][0]

conv3_block4_1_conv (Conv2D)   (None, 28, 28, 128)              65,664 ␣
↪conv3_block3_out[0][0]

conv3_block4_1_bn            (None, 28, 28, 128)                 512 ␣
↪conv3_block4_1_conv[0][0]
(BatchNormalization)                                                    ␣
↪

conv3_block4_1_relu          (None, 28, 28, 128)                   0 ␣
↪conv3_block4_1_bn[0][0]
(Activation)                                                            ␣
↪

conv3_block4_2_conv (Conv2D)   (None, 28, 28, 128)             147,584 ␣
↪conv3_block4_1_relu[0][0]

conv3_block4_2_bn            (None, 28, 28, 128)                 512 ␣
↪conv3_block4_2_conv[0][0]
(BatchNormalization)                                                    ␣
↪

conv3_block4_2_relu          (None, 28, 28, 128)                   0 ␣
↪conv3_block4_2_bn[0][0]
(Activation)                                                            ␣
↪
```

```
conv3_block4_3_conv (Conv2D)    (None, 28, 28, 512)               66,048 ␣
↪conv3_block4_2_relu[0][0]

conv3_block4_3_bn               (None, 28, 28, 512)                2,048 ␣
↪conv3_block4_3_conv[0][0]
(BatchNormalization)                                                      ␣
↪

conv3_block4_add (Add)          (None, 28, 28, 512)                    0 ␣
↪conv3_block3_out[0][0],
                                                                         ␣
↪conv3_block4_3_bn[0][0]

conv3_block4_out (Activation)   (None, 28, 28, 512)                    0 ␣
↪conv3_block4_add[0][0]

conv4_block1_1_conv (Conv2D)    (None, 14, 14, 256)              131,328 ␣
↪conv3_block4_out[0][0]

conv4_block1_1_bn               (None, 14, 14, 256)                1,024 ␣
↪conv4_block1_1_conv[0][0]
(BatchNormalization)                                                      ␣
↪

conv4_block1_1_relu             (None, 14, 14, 256)                    0 ␣
↪conv4_block1_1_bn[0][0]
(Activation)                                                              ␣
↪

conv4_block1_2_conv (Conv2D)    (None, 14, 14, 256)              590,080 ␣
↪conv4_block1_1_relu[0][0]

conv4_block1_2_bn               (None, 14, 14, 256)                1,024 ␣
↪conv4_block1_2_conv[0][0]
(BatchNormalization)                                                      ␣
↪

conv4_block1_2_relu             (None, 14, 14, 256)                    0 ␣
↪conv4_block1_2_bn[0][0]
(Activation)                                                              ␣
↪

conv4_block1_0_conv (Conv2D)    (None, 14, 14, 1024)             525,312 ␣
↪conv3_block4_out[0][0]
```

```
conv4_block1_3_conv (Conv2D)    (None, 14, 14, 1024)            263,168 ␣
↪conv4_block1_2_relu[0][0]

conv4_block1_0_bn               (None, 14, 14, 1024)              4,096 ␣
↪conv4_block1_0_conv[0][0]
(BatchNormalization)                                                     ␣
↪

conv4_block1_3_bn               (None, 14, 14, 1024)              4,096 ␣
↪conv4_block1_3_conv[0][0]
(BatchNormalization)                                                     ␣
↪

conv4_block1_add (Add)          (None, 14, 14, 1024)                  0 ␣
↪conv4_block1_0_bn[0][0],
                                                                        ␣
↪conv4_block1_3_bn[0][0]

conv4_block1_out (Activation)   (None, 14, 14, 1024)                  0 ␣
↪conv4_block1_add[0][0]

conv4_block2_1_conv (Conv2D)    (None, 14, 14, 256)             262,400 ␣
↪conv4_block1_out[0][0]

conv4_block2_1_bn               (None, 14, 14, 256)               1,024 ␣
↪conv4_block2_1_conv[0][0]
(BatchNormalization)                                                     ␣
↪

conv4_block2_1_relu             (None, 14, 14, 256)                   0 ␣
↪conv4_block2_1_bn[0][0]
(Activation)                                                             ␣
↪

conv4_block2_2_conv (Conv2D)    (None, 14, 14, 256)             590,080 ␣
↪conv4_block2_1_relu[0][0]

conv4_block2_2_bn               (None, 14, 14, 256)               1,024 ␣
↪conv4_block2_2_conv[0][0]
(BatchNormalization)                                                     ␣
↪

conv4_block2_2_relu             (None, 14, 14, 256)                   0 ␣
↪conv4_block2_2_bn[0][0]
(Activation)                                                             ␣
↪
```

```
conv4_block2_3_conv (Conv2D)    (None, 14, 14, 1024)          263,168 ␣
↪conv4_block2_2_relu[0][0]

conv4_block2_3_bn               (None, 14, 14, 1024)            4,096 ␣
↪conv4_block2_3_conv[0][0]
(BatchNormalization)                                                    ␣
↪

conv4_block2_add (Add)          (None, 14, 14, 1024)                0 ␣
↪conv4_block1_out[0][0],
                                                                      ␣
↪conv4_block2_3_bn[0][0]

conv4_block2_out (Activation)   (None, 14, 14, 1024)                0 ␣
↪conv4_block2_add[0][0]

conv4_block3_1_conv (Conv2D)    (None, 14, 14, 256)           262,400 ␣
↪conv4_block2_out[0][0]

conv4_block3_1_bn               (None, 14, 14, 256)             1,024 ␣
↪conv4_block3_1_conv[0][0]
(BatchNormalization)                                                    ␣
↪

conv4_block3_1_relu             (None, 14, 14, 256)                 0 ␣
↪conv4_block3_1_bn[0][0]
(Activation)                                                            ␣
↪

conv4_block3_2_conv (Conv2D)    (None, 14, 14, 256)           590,080 ␣
↪conv4_block3_1_relu[0][0]

conv4_block3_2_bn               (None, 14, 14, 256)             1,024 ␣
↪conv4_block3_2_conv[0][0]
(BatchNormalization)                                                    ␣
↪

conv4_block3_2_relu             (None, 14, 14, 256)                 0 ␣
↪conv4_block3_2_bn[0][0]
(Activation)                                                            ␣
↪

conv4_block3_3_conv (Conv2D)    (None, 14, 14, 1024)          263,168 ␣
↪conv4_block3_2_relu[0][0]
```

```
conv4_block3_3_bn              (None, 14, 14, 1024)         4,096  ␣
↪conv4_block3_3_conv[0][0]
(BatchNormalization)                                                  ␣
↪

conv4_block3_add (Add)         (None, 14, 14, 1024)             0  ␣
↪conv4_block2_out[0][0],
                                                                   ␣
↪conv4_block3_3_bn[0][0]

conv4_block3_out (Activation)  (None, 14, 14, 1024)             0  ␣
↪conv4_block3_add[0][0]

conv4_block4_1_conv (Conv2D)   (None, 14, 14, 256)        262,400  ␣
↪conv4_block3_out[0][0]

conv4_block4_1_bn              (None, 14, 14, 256)          1,024  ␣
↪conv4_block4_1_conv[0][0]
(BatchNormalization)                                                  ␣
↪

conv4_block4_1_relu            (None, 14, 14, 256)              0  ␣
↪conv4_block4_1_bn[0][0]
(Activation)                                                          ␣
↪

conv4_block4_2_conv (Conv2D)   (None, 14, 14, 256)        590,080  ␣
↪conv4_block4_1_relu[0][0]

conv4_block4_2_bn              (None, 14, 14, 256)          1,024  ␣
↪conv4_block4_2_conv[0][0]
(BatchNormalization)                                                  ␣
↪

conv4_block4_2_relu            (None, 14, 14, 256)              0  ␣
↪conv4_block4_2_bn[0][0]
(Activation)                                                          ␣
↪

conv4_block4_3_conv (Conv2D)   (None, 14, 14, 1024)       263,168  ␣
↪conv4_block4_2_relu[0][0]

conv4_block4_3_bn              (None, 14, 14, 1024)         4,096  ␣
↪conv4_block4_3_conv[0][0]
(BatchNormalization)                                                  ␣
↪
```

```
conv4_block4_add (Add)        (None, 14, 14, 1024)              0 ␣
↪conv4_block3_out[0][0],
                                                                   ␣
↪conv4_block4_3_bn[0][0]

conv4_block4_out (Activation)  (None, 14, 14, 1024)             0 ␣
↪conv4_block4_add[0][0]

conv4_block5_1_conv (Conv2D)   (None, 14, 14, 256)         262,400 ␣
↪conv4_block4_out[0][0]

conv4_block5_1_bn              (None, 14, 14, 256)           1,024 ␣
↪conv4_block5_1_conv[0][0]
(BatchNormalization)                                                  ␣
↪

conv4_block5_1_relu            (None, 14, 14, 256)               0 ␣
↪conv4_block5_1_bn[0][0]
(Activation)                                                          ␣
↪

conv4_block5_2_conv (Conv2D)   (None, 14, 14, 256)         590,080 ␣
↪conv4_block5_1_relu[0][0]

conv4_block5_2_bn              (None, 14, 14, 256)           1,024 ␣
↪conv4_block5_2_conv[0][0]
(BatchNormalization)                                                  ␣
↪

conv4_block5_2_relu            (None, 14, 14, 256)               0 ␣
↪conv4_block5_2_bn[0][0]
(Activation)                                                          ␣
↪

conv4_block5_3_conv (Conv2D)   (None, 14, 14, 1024)        263,168 ␣
↪conv4_block5_2_relu[0][0]

conv4_block5_3_bn              (None, 14, 14, 1024)          4,096 ␣
↪conv4_block5_3_conv[0][0]
(BatchNormalization)                                                  ␣
↪

conv4_block5_add (Add)         (None, 14, 14, 1024)              0 ␣
↪conv4_block4_out[0][0],
```

```
↪conv4_block5_3_bn[0][0]

conv4_block5_out (Activation)   (None, 14, 14, 1024)              0 ␣
↪conv4_block5_add[0][0]

conv4_block6_1_conv (Conv2D)    (None, 14, 14, 256)         262,400 ␣
↪conv4_block5_out[0][0]

conv4_block6_1_bn               (None, 14, 14, 256)           1,024 ␣
↪conv4_block6_1_conv[0][0]
(BatchNormalization)                                                   ␣
↪

conv4_block6_1_relu             (None, 14, 14, 256)               0 ␣
↪conv4_block6_1_bn[0][0]
(Activation)                                                           ␣
↪

conv4_block6_2_conv (Conv2D)    (None, 14, 14, 256)         590,080 ␣
↪conv4_block6_1_relu[0][0]

conv4_block6_2_bn               (None, 14, 14, 256)           1,024 ␣
↪conv4_block6_2_conv[0][0]
(BatchNormalization)                                                   ␣
↪

conv4_block6_2_relu             (None, 14, 14, 256)               0 ␣
↪conv4_block6_2_bn[0][0]
(Activation)                                                           ␣
↪

conv4_block6_3_conv (Conv2D)    (None, 14, 14, 1024)        263,168 ␣
↪conv4_block6_2_relu[0][0]

conv4_block6_3_bn               (None, 14, 14, 1024)          4,096 ␣
↪conv4_block6_3_conv[0][0]
(BatchNormalization)                                                   ␣
↪

conv4_block6_add (Add)          (None, 14, 14, 1024)              0 ␣
↪conv4_block5_out[0][0],
                                                                    ␣
↪conv4_block6_3_bn[0][0]
```

```
conv4_block6_out (Activation)  (None, 14, 14, 1024)               0 ␣
↪conv4_block6_add[0][0]

conv5_block1_1_conv (Conv2D)   (None, 7, 7, 512)            524,800 ␣
↪conv4_block6_out[0][0]

conv5_block1_1_bn              (None, 7, 7, 512)              2,048 ␣
↪conv5_block1_1_conv[0][0]
(BatchNormalization)                                                   ␣
↪

conv5_block1_1_relu            (None, 7, 7, 512)                  0 ␣
↪conv5_block1_1_bn[0][0]
(Activation)                                                           ␣
↪

conv5_block1_2_conv (Conv2D)   (None, 7, 7, 512)          2,359,808 ␣
↪conv5_block1_1_relu[0][0]

conv5_block1_2_bn              (None, 7, 7, 512)              2,048 ␣
↪conv5_block1_2_conv[0][0]
(BatchNormalization)                                                   ␣
↪

conv5_block1_2_relu            (None, 7, 7, 512)                  0 ␣
↪conv5_block1_2_bn[0][0]
(Activation)                                                           ␣
↪

conv5_block1_0_conv (Conv2D)   (None, 7, 7, 2048)         2,099,200 ␣
↪conv4_block6_out[0][0]

conv5_block1_3_conv (Conv2D)   (None, 7, 7, 2048)         1,050,624 ␣
↪conv5_block1_2_relu[0][0]

conv5_block1_0_bn              (None, 7, 7, 2048)             8,192 ␣
↪conv5_block1_0_conv[0][0]
(BatchNormalization)                                                   ␣
↪

conv5_block1_3_bn              (None, 7, 7, 2048)             8,192 ␣
↪conv5_block1_3_conv[0][0]
(BatchNormalization)                                                   ␣
↪
```

```
conv5_block1_add (Add)          (None, 7, 7, 2048)                   0 ␣
↪conv5_block1_0_bn[0][0],                                              ␣
↪conv5_block1_3_bn[0][0]

conv5_block1_out (Activation)   (None, 7, 7, 2048)                   0 ␣
↪conv5_block1_add[0][0]

conv5_block2_1_conv (Conv2D)    (None, 7, 7, 512)            1,049,088 ␣
↪conv5_block1_out[0][0]

conv5_block2_1_bn               (None, 7, 7, 512)                2,048 ␣
↪conv5_block2_1_conv[0][0]
(BatchNormalization)                                                     ␣
↪

conv5_block2_1_relu             (None, 7, 7, 512)                    0 ␣
↪conv5_block2_1_bn[0][0]
(Activation)                                                             ␣
↪

conv5_block2_2_conv (Conv2D)    (None, 7, 7, 512)            2,359,808 ␣
↪conv5_block2_1_relu[0][0]

conv5_block2_2_bn               (None, 7, 7, 512)                2,048 ␣
↪conv5_block2_2_conv[0][0]
(BatchNormalization)                                                     ␣
↪

conv5_block2_2_relu             (None, 7, 7, 512)                    0 ␣
↪conv5_block2_2_bn[0][0]
(Activation)                                                             ␣
↪

conv5_block2_3_conv (Conv2D)    (None, 7, 7, 2048)           1,050,624 ␣
↪conv5_block2_2_relu[0][0]

conv5_block2_3_bn               (None, 7, 7, 2048)               8,192 ␣
↪conv5_block2_3_conv[0][0]
(BatchNormalization)                                                     ␣
↪

conv5_block2_add (Add)          (None, 7, 7, 2048)                   0 ␣
↪conv5_block1_out[0][0],                                               ␣
↪conv5_block2_3_bn[0][0]
```

```
conv5_block2_out (Activation)   (None, 7, 7, 2048)                0 ␣
↪conv5_block2_add[0][0]

conv5_block3_1_conv (Conv2D)    (None, 7, 7, 512)         1,049,088 ␣
↪conv5_block2_out[0][0]

conv5_block3_1_bn               (None, 7, 7, 512)             2,048 ␣
↪conv5_block3_1_conv[0][0]
(BatchNormalization)                                                  ␣
↪

conv5_block3_1_relu             (None, 7, 7, 512)                 0 ␣
↪conv5_block3_1_bn[0][0]
(Activation)                                                          ␣
↪

conv5_block3_2_conv (Conv2D)    (None, 7, 7, 512)         2,359,808 ␣
↪conv5_block3_1_relu[0][0]

conv5_block3_2_bn               (None, 7, 7, 512)             2,048 ␣
↪conv5_block3_2_conv[0][0]
(BatchNormalization)                                                  ␣
↪

conv5_block3_2_relu             (None, 7, 7, 512)                 0 ␣
↪conv5_block3_2_bn[0][0]
(Activation)                                                          ␣
↪

conv5_block3_3_conv (Conv2D)    (None, 7, 7, 2048)        1,050,624 ␣
↪conv5_block3_2_relu[0][0]

conv5_block3_3_bn               (None, 7, 7, 2048)            8,192 ␣
↪conv5_block3_3_conv[0][0]
(BatchNormalization)                                                  ␣
↪

conv5_block3_add (Add)          (None, 7, 7, 2048)                0 ␣
↪conv5_block2_out[0][0],
                                                                    ␣
↪conv5_block3_3_bn[0][0]

conv5_block3_out (Activation)   (None, 7, 7, 2048)                0 ␣
↪conv5_block3_add[0][0]
```

```
flatten_10 (Flatten)          (None, 100352)                    0 ␣
↪conv5_block3_out[0][0]

dense_10 (Dense)              (None, 4)                   401,412 ␣
↪flatten_10[0][0]
```

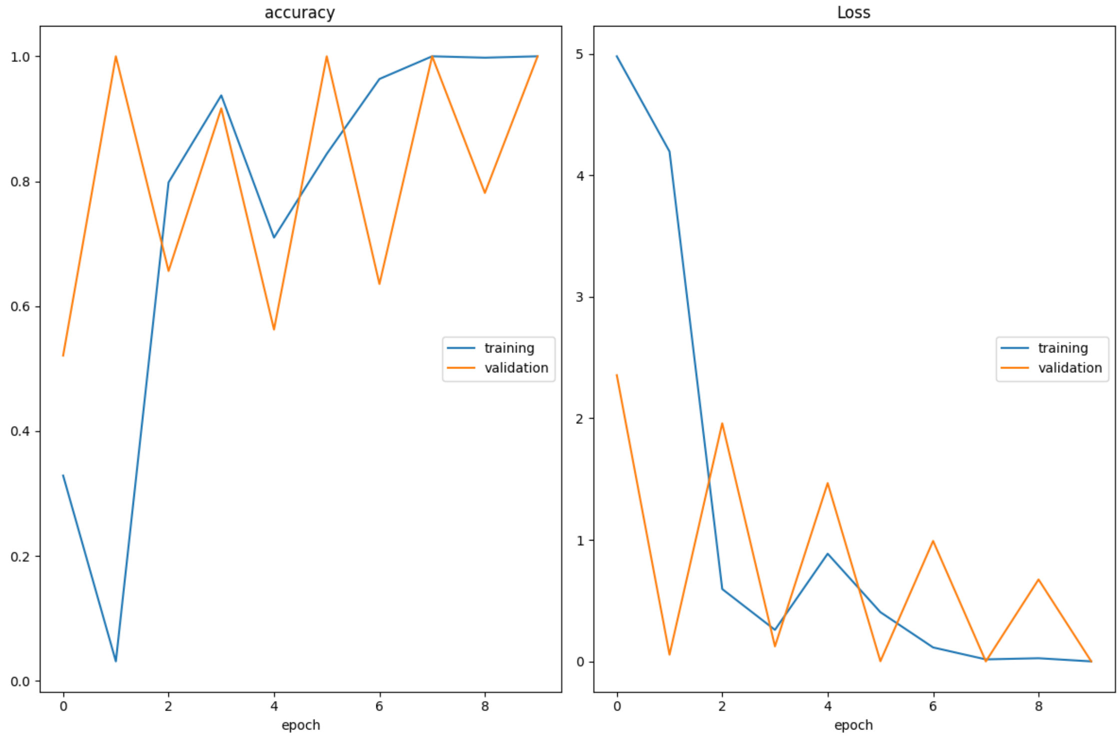
Total params: 23,989,124 (91.51 MB)

Trainable params: 401,412 (1.53 MB)

Non-trainable params: 23,587,712 (89.98 MB)

[112]:

```python
%%time

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

steps_per_epoch = traingen_noda.samples // BATCH_SIZE
val_steps = validgen_noda.samples // BATCH_SIZE

n_epochs = 10

optimizer = RMSprop(learning_rate=0.0001)

model_resnet_tl_no_da.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_resnet_tl_no_da.weights.best.
 ↪keras',
                               verbose=1,
                               save_best_only=True)

print('Resnet50 sem data augmentation')
# Treinamento do Modelo
history_resnet_tl_no_da = model_resnet_tl_no_da.fit(traingen_noda,
                    epochs=n_epochs,
                    steps_per_epoch=steps_per_epoch,
                    validation_data=validgen_noda,
                    validation_steps=val_steps,
                    callbacks=[checkpointer, PlotLossesKeras()],
                    verbose=True)
```

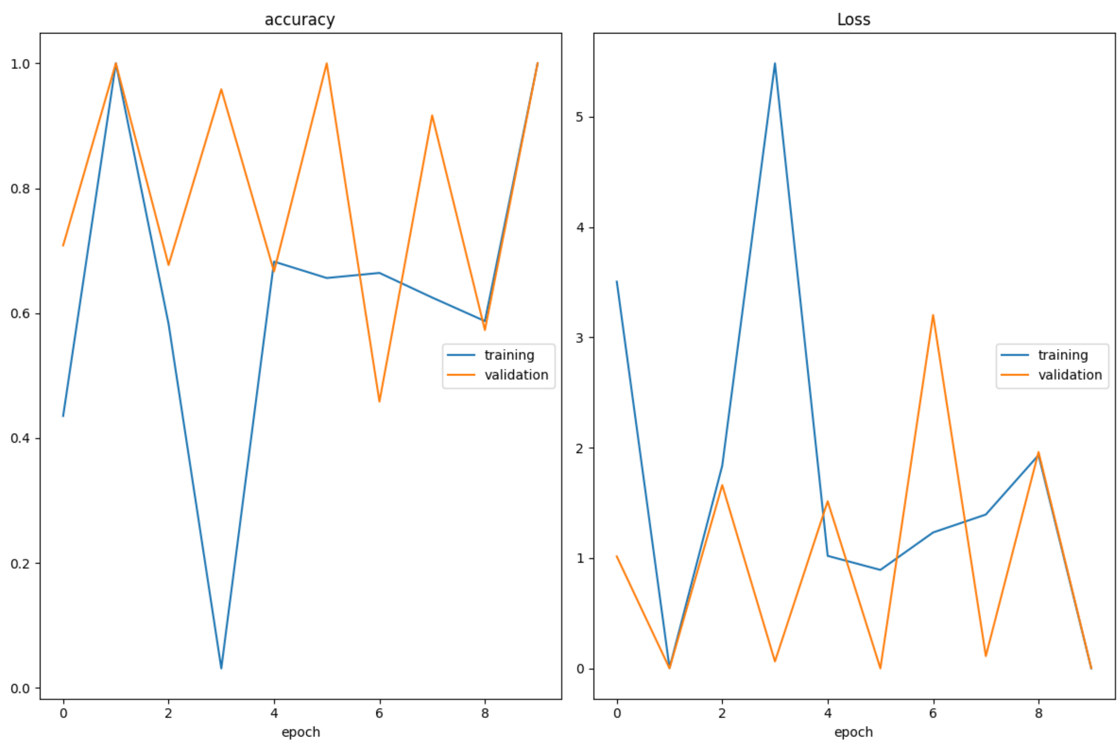



```
accuracy
        training                 (min:    0.031, max:    1.000, cur:    1.000)
        validation               (min:    0.521, max:    1.000, cur:    1.000)
Loss
        training                 (min:    0.000, max:    4.978, cur:    0.000)
        validation               (min:    0.000, max:    2.355, cur:    0.000)
14/14                 3s 114ms/step -
accuracy: 1.0000 - loss: 1.3132e-05 - val_accuracy: 1.0000 - val_loss:
3.3627e-05
CPU times: total: 11min 12s
Wall time: 2min 21s
```

3.1.2 Resnet50 com Transfer Learning - com data augmentation

[113]:
```
%%time
steps_per_epoch = traingen.samples // BATCH_SIZE
val_steps = validgen.samples // BATCH_SIZE

model_resnet_tl_da.compile(loss='categorical_crossentropy',␣
 ↪optimizer=optimizer, metrics=['accuracy'])

checkpointer = ModelCheckpoint(filepath='img_model_resnet_tl_da.weights.best.
 ↪keras',
```

```
                               verbose=1,
                               save_best_only=True)

history_resnet_tl_da = model_resnet_tl_da.fit(traingen,
                    epochs=n_epochs,
                    steps_per_epoch=steps_per_epoch,
                    validation_data=validgen,
                    validation_steps=val_steps,
                    callbacks=[checkpointer, PlotLossesKeras()],
                    verbose=True)
```




```
accuracy
        training             (min:    0.031, max:    1.000, cur:    1.000)
        validation           (min:    0.458, max:    1.000, cur:    1.000)
Loss
        training             (min:    0.000, max:    5.483, cur:    0.000)
        validation           (min:    0.000, max:    3.202, cur:    0.003)
14/14                  3s 121ms/step -
accuracy: 1.0000 - loss: 1.9688e-04 - val_accuracy: 1.0000 - val_loss: 0.0030
CPU times: total: 12min
Wall time: 3min 16s
```

3.1.3 VGG16 com Transfer Learning - sem data augmentation

[114]:
```
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input as␣
↪vgg16_preprocess
```

[115]:
```
# Sem data augmentation

print('Transformador de imagens sem data augmentation')
train_generator_vgg_noda =␣
↪ImageDataGenerator(preprocessing_function=vgg16_preprocess)

test_generator_vgg_noda =␣
↪ImageDataGenerator(preprocessing_function=vgg16_preprocess)

print('No data augmentation - train')
traingen_vgg_noda = train_generator_vgg_noda.flow_from_directory(IMAGE_BASE_DIR␣
↪+ '/train',
                                               target_size=(224, 224),
                                               batch_size=BATCH_SIZE,
                                               class_mode='categorical',
                                               classes=classes,
                                               shuffle=False,
                                               seed=42)

print('No Data augmentation - validation')
validgen_vgg_noda = train_generator_vgg_noda.flow_from_directory(IMAGE_BASE_DIR␣
↪+ '/val',
                                               target_size=(224, 224),
                                               batch_size=BATCH_SIZE,
                                               class_mode='categorical',
                                               classes=classes,
                                               shuffle=False,
                                               seed=42)

print('No Data augmentation - test')
testgen_vgg_noda = test_generator_vgg_noda.flow_from_directory('test/
↪Test_4cl_amostra',
                                             target_size=(224, 224),
                                             batch_size=BATCH_SIZE,
                                             class_mode=None,
                                             classes=classes,
                                             shuffle=False,
                                             seed=42)
```

```
Transformador de imagens sem data augmentation
No data augmentation - train
Found 473 images belonging to 4 classes.
No Data augmentation - validation
```

```
Found 120 images belonging to 4 classes.
No Data augmentation - test
Found 371 images belonging to 4 classes.
```

[116]:

```
print('Transformador de imagens sem data augmentation')

train_generator_vgg = 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,
                                     channel_shift_range=25.0,
                                     zoom_range=0.1,
                                     shear_range=0.15,
                                     preprocessing_function=vgg16_preprocess)

test_vgg_generator = ImageDataGenerator(preprocessing_function=vgg16_preprocess)

BATCH_SIZE = 32 # quantidade de imagens criadas em cada ciclo

print('Data augmentation - train')
traingen_vgg = train_generator_vgg.flow_from_directory(IMAGE_BASE_DIR + '/
 ↪train',
                                               target_size=(224, 224),
                                               batch_size=BATCH_SIZE,
                                               class_mode='categorical',
                                               classes=classes,
                                               shuffle=False,
                                               seed=42)

print('Data augmentation - validation')
validgen_vgg = train_generator_vgg.flow_from_directory(IMAGE_BASE_DIR + '/val',
                                               target_size=(224, 224),
                                               batch_size=BATCH_SIZE,
                                               class_mode='categorical',
                                               classes=classes,
                                               shuffle=False,
                                               seed=42)

print('Data augmentation - test')
testgen_vgg = test_vgg_generator.flow_from_directory('test/Test_4cl_amostra',
                                             target_size=(224, 224),
                                             batch_size=BATCH_SIZE,
                                             class_mode=None,
                                             classes=classes,
```

```
                                        shuffle=False,
                                        seed=42)
```

```
Transformador de imagens sem data augmentation
Data augmentation - train
Found 473 images belonging to 4 classes.
Data augmentation - validation
Found 120 images belonging to 4 classes.
Data augmentation - test
Found 371 images belonging to 4 classes.
```

[117]:
```
# A opção include_top=False não inclui as camadas de aprendizado da rede␣
 ↪original
# Utiliza os pesos treinados na base imagenet
vgg16_tl = VGG16(input_shape=(224,224,3), weights='imagenet', include_top=False)

# não treinar os pesos existentes
for layer in vgg16_tl.layers:
  layer.trainable = False

# A saída da VGG será a entrada da camada criada
x_tl = Flatten()(vgg16_tl.output)

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

# Criação do Objeto Modelo (a parte da vgg + as camadas Fully connected criadas)
model_vgg_tl_no_da = Model(inputs=vgg16_tl.input, outputs=prediction)
model_vgg_tl_da = Model(inputs=vgg16_tl.input, outputs=prediction)
```

[118]:
```
model_vgg_tl_no_da.summary()
```

```
Model: "functional_23"


 Layer (type)                          Output Shape                              ␣
↪Param #

 input_layer_12 (InputLayer)           (None, 224, 224, 3)                           ␣
↪  0

 block1_conv1 (Conv2D)                 (None, 224, 224, 64)                        ␣
↪1,792

 block1_conv2 (Conv2D)                 (None, 224, 224, 64)                        ␣
↪36,928
```

```
block1_pool (MaxPooling2D)          (None, 112, 112, 64)              ␣
↪  0

block2_conv1 (Conv2D)               (None, 112, 112, 128)          ␣
↪73,856

block2_conv2 (Conv2D)               (None, 112, 112, 128)         ␣
↪147,584

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

block3_conv1 (Conv2D)               (None, 56, 56, 256)           ␣
↪295,168

block3_conv2 (Conv2D)               (None, 56, 56, 256)           ␣
↪590,080

block3_conv3 (Conv2D)               (None, 56, 56, 256)           ␣
↪590,080

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

block4_conv1 (Conv2D)               (None, 28, 28, 512)         ␣
↪1,180,160

block4_conv2 (Conv2D)               (None, 28, 28, 512)         ␣
↪2,359,808

block4_conv3 (Conv2D)               (None, 28, 28, 512)         ␣
↪2,359,808

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

block5_conv1 (Conv2D)               (None, 14, 14, 512)         ␣
↪2,359,808

block5_conv2 (Conv2D)               (None, 14, 14, 512)         ␣
↪2,359,808

block5_conv3 (Conv2D)               (None, 14, 14, 512)         ␣
↪2,359,808

block5_pool (MaxPooling2D)          (None, 7, 7, 512)                 ␣
↪  0
```

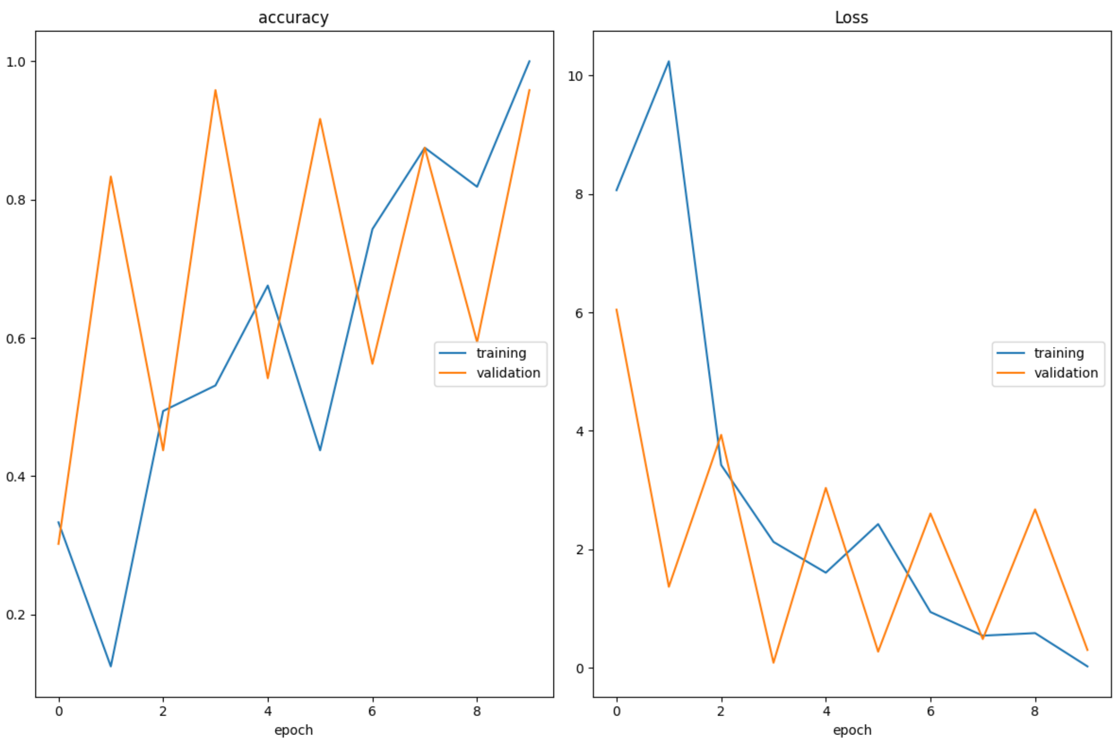
```
 flatten_11 (Flatten)                 (None, 25088)                          ␣
↪  0

 dense_11 (Dense)                     (None, 4)                            ␣
↪100,356
```

Total params: 14,815,044 (56.51 MB)

Trainable params: 100,356 (392.02 KB)

Non-trainable params: 14,714,688 (56.13 MB)

[119]:
```
%%time

steps_per_epoch = traingen_vgg_noda.samples // BATCH_SIZE
val_steps = validgen_vgg_noda.samples // BATCH_SIZE

n_epochs = 10

optimizer = RMSprop(learning_rate=0.0001)

model_vgg_tl_no_da.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_vgg_tl_no_da.weights.best.
 ↪keras',
                               verbose=1,
                               save_best_only=True)

print('VGG16 sem data augmentation')
# Treinamento do Modelo
history_vgg_tl_no_da = model_vgg_tl_no_da.fit(traingen_vgg_noda,
                    epochs=n_epochs,
                    steps_per_epoch=steps_per_epoch,
                    validation_data=validgen_vgg_noda,
                    validation_steps=val_steps,
                    callbacks=[checkpointer, PlotLossesKeras()],
                    verbose=True)
```



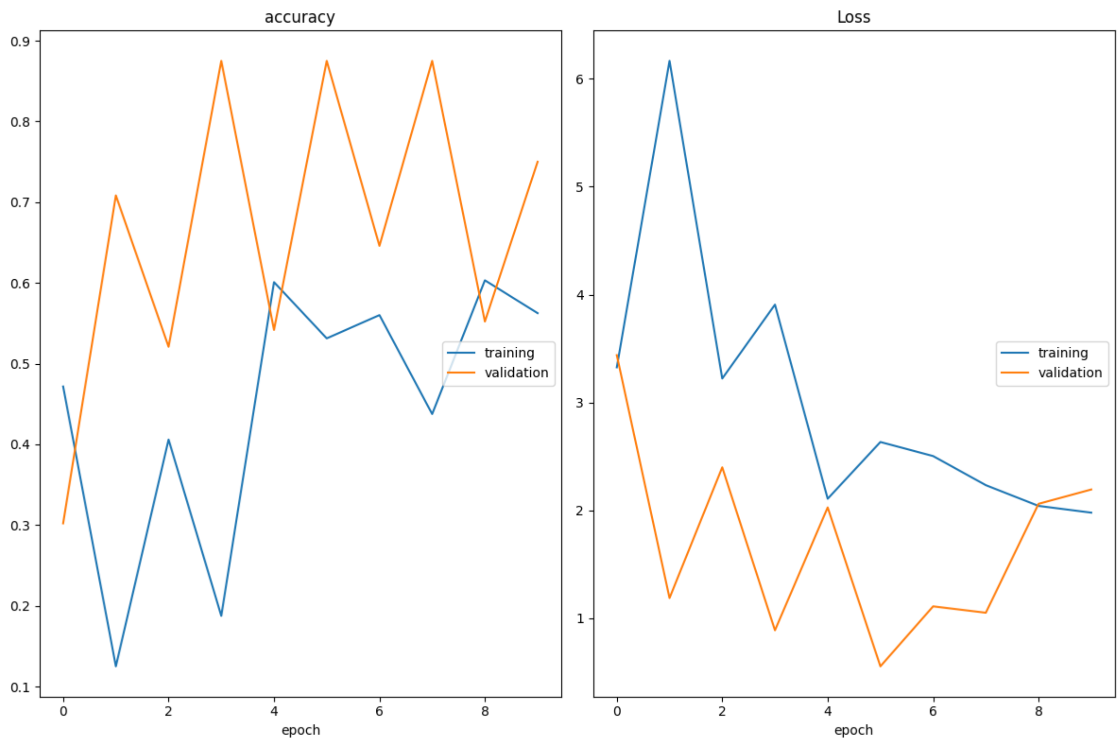

```
accuracy
	training             (min:    0.125, max:    1.000, cur:    1.000)
	validation           (min:    0.302, max:    0.958, cur:    0.958)
Loss
	training             (min:    0.023, max:   10.241, cur:    0.023)
	validation           (min:    0.085, max:    6.047, cur:    0.301)
14/14                  6s 205ms/step -
accuracy: 1.0000 - loss: 0.0231 - val_accuracy: 0.9583 - val_loss: 0.3007
CPU times: total: 33min 19s
Wall time: 4min 53s
```

3.1.4 VGG16 com Transfer Learning - com data augmentation

[120]:

```
%%time
steps_per_epoch = traingen_vgg.samples // BATCH_SIZE
val_steps = validgen_vgg.samples // BATCH_SIZE

model_vgg_tl_da.compile(loss='categorical_crossentropy', optimizer=optimizer,␣
↪metrics=['accuracy'])

checkpointer = ModelCheckpoint(filepath='img_model_vgg_tl_da.weights.best.
↪keras',
                               verbose=1,
```

```
                                             save_best_only=True)

history_vgg_tl_da = model_vgg_tl_da.fit(traingen_vgg,
                            epochs=n_epochs,
                            steps_per_epoch=steps_per_epoch,
                            validation_data=validgen_vgg,
                            validation_steps=val_steps,
                            callbacks=[checkpointer, PlotLossesKeras()],
                            verbose=True)
```




```
accuracy
        training                 (min:    0.125, max:    0.603, cur:    0.562)
        validation               (min:    0.302, max:    0.875, cur:    0.750)
Loss
        training                 (min:    1.979, max:    6.166, cur:    1.979)
        validation               (min:    0.554, max:    3.439, cur:    2.193)
14/14            6s 201ms/step -
accuracy: 0.5625 - loss: 1.9787 - val_accuracy: 0.7500 - val_loss: 2.1934
CPU times: total: 33min 55s
Wall time: 6min
```

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

[121]:
```
%%time
from sklearn.metrics import accuracy_score

# Generate predictions
print('Carregando os modelos')
model_resnet_tl_no_da.load_weights('img_model_resnet_tl_no_da.weights.best.
↪keras')
model_resnet_tl_da.load_weights('img_model_resnet_tl_da.weights.best.keras')

model_vgg_tl_no_da.load_weights('img_model_vgg_tl_no_da.weights.best.keras')
model_vgg_tl_da.load_weights('img_model_vgg_tl_da.weights.best.keras')

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

true_classes_resnet_no_da = testgen_noda.classes
class_indices_resnet_no_da = traingen_noda.class_indices
class_indices_resnet_no_da = dict((v,k) for k,v in class_indices_resnet_no_da.
↪items())

true_classes_vgg = testgen_vgg.classes
class_indices_vgg = traingen_vgg.class_indices
class_indices_vgg = dict((v,k) for k,v in class_indices_vgg.items())

true_classes_vgg_no_da = testgen_vgg_noda.classes
class_indices_vgg_no_da = traingen_vgg_noda.class_indices
class_indices_vgg_no_da = dict((v,k) for k,v in class_indices_vgg_no_da.items())

print('Aplicando os modelos nas imagens de teste')
print('Resnet50 sem data augmentation')
preds_resnet_no_da = model_resnet_tl_no_da.predict(testgen_noda)
pred_classes_resnet_no_da = np.argmax(preds_resnet_no_da, axis=1)

print('Resnet50 com data augmentation')
preds_resnet_da = model_resnet_tl_da.predict(testgen)
pred_classes_resnet_da = np.argmax(preds_resnet_da, axis=1)

print('VGG16 sem data augmentation')
preds_vgg_no_da = model_vgg_tl_no_da.predict(testgen_vgg_noda)
pred_classes_vgg_no_da = np.argmax(preds_vgg_no_da, axis=1)

print('VGG16 com data augmentation')
preds_vgg_da = model_vgg_tl_da.predict(testgen_vgg)
pred_classes_vgg_da = np.argmax(preds_vgg_da, axis=1)
```

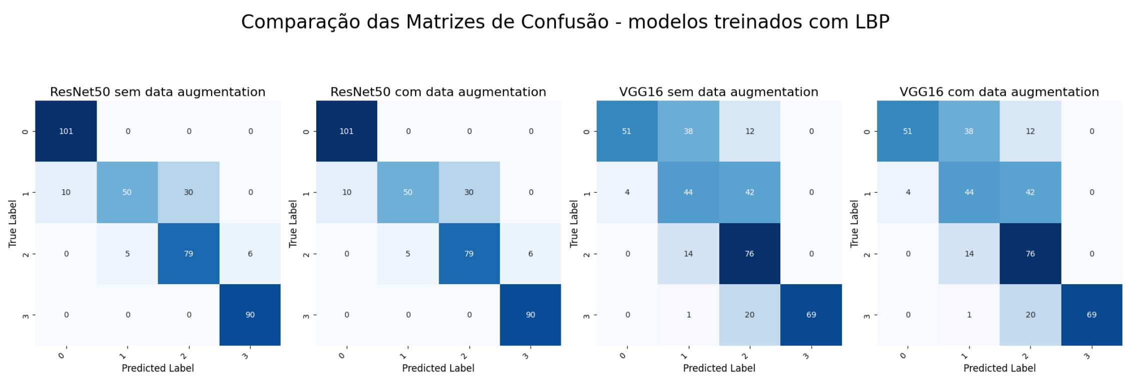
```
Carregando os modelos
Aplicando os modelos nas imagens de teste
Resnet50 sem data augmentation
12/12              20s 1s/step
Resnet50 com data augmentation
12/12              19s 1s/step
VGG16 sem data augmentation
12/12              36s 3s/step
VGG16 com data augmentation
12/12              36s 3s/step
CPU times: total: 11min 55s
Wall time: 2min 39s
```

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

[122]:
```python
print('Calculando as métricas')

acc_resnet_no_da = accuracy_score(true_classes_resnet_no_da, pred_classes_resnet_no_da)
print("Acurácia Modelo ResNet50 sem data augmentation: {:.2f}%".format(acc_resnet_no_da * 100))

acc_resnet_da = accuracy_score(true_classes_resnet, pred_classes_resnet_da)
print("Acurácia Modelo ResNet50 com data augmentation: {:.2f}%".format(acc_resnet_da * 100))

acc_vgg_no_da = accuracy_score(true_classes_vgg_no_da, pred_classes_vgg_no_da)
print("Acurácia Modelo VGG16 sem data augmentation: {:.2f}%".format(acc_vgg_no_da * 100))

acc_vgg_da = accuracy_score(true_classes_vgg, pred_classes_vgg_da)
print("Acurácia Modelo VGG16 com data augmentation: {:.2f}%".format(acc_vgg_da * 100))

fig, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, figsize=(20, 10))

plot_heatmap(true_classes_resnet_no_da, pred_classes_resnet_no_da, classes, ax1, title="ResNet50 sem data augmentation")
plot_heatmap(true_classes_resnet, pred_classes_resnet_da, classes, ax2, title="ResNet50 com data augmentation")
plot_heatmap(true_classes_vgg_no_da, pred_classes_vgg_no_da, classes, ax3, title="VGG16 sem data augmentation")
plot_heatmap(true_classes_vgg, pred_classes_vgg_da, classes, ax4, title="VGG16 com data augmentation")
```

```
fig.suptitle("Comparação das Matrizes de Confusão - modelos treinados com LBP",␣
 ↪fontsize=24)
fig.tight_layout()
fig.subplots_adjust(top=1.2)
plt.show()


print("Métricas ResNet50 sem Data Augmention")
print(classification_report(true_classes_resnet_no_da,␣
 ↪pred_classes_resnet_no_da, digits=3))
print("Métricas ResNet50 com Data Augmention")
print(classification_report(true_classes_resnet, pred_classes_resnet_da,␣
 ↪digits=3))
print("Métricas VGG16 sem Data Augmention")
print(classification_report(true_classes_vgg_no_da, pred_classes_vgg_no_da,␣
 ↪digits=3))
print("Métricas VGG16 com Data Augmention")
print(classification_report(true_classes_vgg, pred_classes_vgg_da, digits=3))
```

```
Calculando as métricas
Acurácia Modelo ResNet50 sem data augmentation: 86.25%
Acurácia Modelo ResNet50 com data augmentation: 86.25%
Acurácia Modelo VGG16 sem data augmentation: 64.69%
Acurácia Modelo VGG16 com data augmentation: 64.69%
```




```
Métricas ResNet50 sem Data Augmention
              precision    recall  f1-score   support

           0      0.910     1.000     0.953       101
           1      0.909     0.556     0.690        90
           2      0.725     0.878     0.794        90
           3      0.938     1.000     0.968        90

    accuracy                          0.863       371
```

```
   macro avg      0.870     0.858     0.851       371
weighted avg      0.871     0.863     0.854       371

Métricas ResNet50 com Data Augmention
              precision    recall  f1-score   support

           0      0.910     1.000     0.953       101
           1      0.909     0.556     0.690        90
           2      0.725     0.878     0.794        90
           3      0.938     1.000     0.968        90

    accuracy                          0.863       371
   macro avg      0.870     0.858     0.851       371
weighted avg      0.871     0.863     0.854       371

Métricas VGG16 sem Data Augmention
              precision    recall  f1-score   support

           0      0.927     0.505     0.654       101
           1      0.454     0.489     0.471        90
           2      0.507     0.844     0.633        90
           3      1.000     0.767     0.868        90

    accuracy                          0.647       371
   macro avg      0.722     0.651     0.656       371
weighted avg      0.728     0.647     0.656       371

Métricas VGG16 com Data Augmention
              precision    recall  f1-score   support

           0      0.927     0.505     0.654       101
           1      0.454     0.489     0.471        90
           2      0.507     0.844     0.633        90
           3      1.000     0.767     0.868        90

    accuracy                          0.647       371
   macro avg      0.722     0.651     0.656       371
weighted avg      0.728     0.647     0.656       371
```

3.2.2 5. Indique qual modelo dá o melhor o resultado e a métrica utilizada

O ResNet50 se mostra melhor que a VGG16 (tanto com ou sem Data Augmentation), pois apresenta o maior **F1-score ponderado (0.854)** e também a maior **acurácia (0.863)** comparado ao VGG16. Isso ocorre devido à sua arquitetura mais profunda e eficiente, que utiliza blocos residuais para facilitar o treinamento de redes muito profundas sem o problema de vanishing gradients permitindo que o modelo aprenda representações mais complexas e discriminativas das imagens, capturando padrões sutis que a VGG16 tende a perder.

```
[ ]:
```