UNIVERSIDADE FEDERAL DA PARAÍBA - UFPB
CENTRO DE INFORMÁTICA - CI
PROCESSAMENTO DIGITAL DE IMAGENS

Trabalho Final
CLASSIFICADOR DE DOENÇAS BUCAIS

CAIO ASSUNÇÃO ALBUQUERQUE - 20160144689
EPITÁCIO PESSOA DE BRITO NETO - 11506856
KAIO MOURA DOS SANTOS- 11506860
JORDAN ELIAS RODRIGUES - 11516379
GUILHERME MOREIRA RODRIGUES -20160105205

JOÃO PESSOA - PB
DEZEMBRO 2020

1. Introdução

Nesse projeto final foi desenvolvido uma rede neural para classificação de doenças bucais: cárie, hiperplasia fibrosa e carcinoma, com intuito de auxiliar o estudante/profissional da área.

2. Materiais e Métodos

Todo o projeto foi desenvolvido na linguagem Python com o Jupyter Notebook que é um interface gráfica que permite a edição de notebooks em um navegador web, como Google Chrome ou Mozilla Firefox. Assim, todos os integrantes do grupo poderiam participar e compartilhar as atualizações entre si, através do github, uma plataforma de hospedagem de código-fonte e arquivos com controle de versão usando o Git.

O processo em que se desenvolveu o trabalho começou com a extração dos dados através de uma extensão do Google Chrome: “Download All Images”, as imagens foram separadas em pastas e subpastas de Treino e Teste onde na pasta de treino temos subpastas com as classes e seus dados pertencentes.

Boca saudável	Cárie	Hiperplasia Fibrosa	Carcinoma
70	425	190	362

Tabela 1: Número de amostras por classe

3. Desenvolvimento

3.1 Pré-Processamento

Para esta etapa de pré-processamento utilizamos o OpenCV como a biblioteca principal.

```
import numpy as np
from imutils import paths
import cv2 as cv
import os
from PIL import Image
import argparse
```

Figura 1: Importando bibliotecas

3.2 Tratamento de imagens duplicadas

Etapa onde removemos manualmente do data frame as imagens duplicadas (imagens com o mesmo hash são consideradas duplicadas).

```
def hash(image, hashSize = 8):

    cinza = cv.cvtColor(image, cv.COLOR_BGR2GRAY) #rgb para cinza
    resize = cv.resize(cinza, (hashSize + 1,  hashSize)) #redimensiona com hashSize = 8

    diff = resize[:, 1:] > resize[:, :-1] #diferença de imagem pelo gradiente horizontal relativo entre os pixels

    return sum([2 ** i for (i, v) in enumerate(diff.flatten()) if v]) #retorna o calculo do hash

#sistema de linha de comando
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True, help="Path to input dataset")
ap.add_argument("-r", "--remove", type=int, default=-1, help="deseja remover duplicas?")
args= vars(ap.parse_args())
```

Figura 2: Tratamento de imagens duplicadas 1

```
#input diretorio do dataset
print("[INFO] computing image hashes...")
imagePaths = list(paths.list_images(args["dataset"]))
hashes = {}

for imagePath in imagePaths:
    image = cv.imread(imagePath)
    h = hash(image)

    p=hashes.get(h, [])
    p.append(imagePath)
    hashes[h] = p
```

Figura 3: Tratamento de imagens duplicadas 2

```
for (h, hashedPaths) in hashes.items():

    if len(hashedPaths) > 1:
        if args["remove"] <= 0:
            montage = None

            for p in hashedPaths:
                image = cv.imread(p)
                image = cv.resize(image, (150, 150))

                if montage is None:
                    montage = image
                else:
                    montage = np.hstack([montage, image])

            print("[INFO] hash: {}".format(h))
            cv.imshow("Montage", montage)
            cv.waitKey(0)
    else:
        for p in hashedPaths[1:]:
            os.remove(p)
```

Figura 4: Tratamento de imagens duplicadas 3

3.3 Conversão do formato da imagem

Aqui convertemos imagens com extensão PNG para JPG a fim de padronizar os nossos dados :

```
png = glob('Teste/fibrosa/*.png')
for i in png:
    im = cv.imread(i)
    cv.imwrite(i[:-3] + 'jpg', im)
```

Figura 5: Conversão png para jpg

3.4 Removendo arquivos de extensões não correspondentes

Nessa etapa removemos arquivos que não possuem a extensão .JPEG da base de dados.

```
#path valida
folder_path = 'Valida' ##Valida e Teste
extensions = []
for filee in os.listdir(folder_path):
    file_path = os.path.join(folder_path, filee)
    print('** Path: {}  **'.format(file_path), end="\r", flush=True)
    im = Image.open(file_path)
    rgb_im = im.convert('RGB')
    if filee.split('.')[1] not in extensions:
        extensions.append(filee.split('.')[1])
```

Figura 6: Removendo arquivos indesejados 1

```
#path treino
folder_path = 'Treino'
extensions = []
for fldr in os.listdir(folder_path):
    sub_folder_path = os.path.join(folder_path, fldr)
    for filee in os.listdir(sub_folder_path):
        file_path = os.path.join(sub_folder_path, filee)
        print('** Path: {}  **'.format(file_path), end="\r", flush=True)
        im = Image.open(file_path)
        rgb_im = im.convert('RGB')
        if filee.split('.')[1] not in extensions:
            extensions.append(filee.split('.')[1])
```

Figura 7: Removendo arquivos indesejados 2

4. Treinamento e teste

Para etapa de treinamento e testes é necessário a importação de algumas bibliotecas especiais como Tensorflow e Keras para a criação e treinamento da rede neural

```
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
from PIL import Image
import io
import tensorflow as tf
import pathlib
import glob as glob
import cv2 as cv
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

Figura 8: Importando Tensorflow e Keras

Tabela 2: Classes

Classe 1	Classe 2	Classe 3	Classe 4
Carcinoma	Hiperplasia Fibrosa	Cáries	Boca Saudável

4.1 Melhorando desempenho

Para melhor desempenho do processamento dos dados utilizamos dois métodos, Dataset.cache() que mantém as imagens na memória depois que são carregadas no disco e Dataset.prefetch() que sobrepõe o pré-processamento de dados e a execução do modelo durante o treinamento

4.2 Padronização dos dados

Para a padronização dos dados temos os valores do canal RGB na faixa [0, 255]. Diminuímos esse valor de entrada para um melhor desempenho da rede neural padronizando os valores de todas as imagens na faixa [0, 1] utilizando uma camada de reescalonamento.

4.3 Criação e compilação dos modelos

Criamos um modelo de rede neural sequencial que foi dividida em três blocos de convolução com uma camada máxima de pool em cada uma delas. Todas as camadas são ativadas por uma função de ativação relu. Além disso, utilizamos a acurácia como métrica de avaliação.

4.4 Treinamento e Teste

Para o treinamento dividimos um número de épocas e obtivemos resultado que nos fizeram decidir se o modelo estava correto. Além disso, obtivemos resultados gráficos do treinamento onde mostram o comportamento da acurácia com a precisão da validação. Como teste utilizamos as imagens disponíveis no path de teste onde obtivemos resultados satisfatórios que serão apresentados logo em seguida.

5. Discussão

Uma das dificuldades observadas para esse projeto está diretamente relacionada a quantidade de imagens disponíveis na base de dados, quanto maior o número de imagens disponíveis para o treinamento, maiores as chances do aumento da métrica de avaliação.

6. Resultados

Os resultados foram totalmente satisfatórios para as classes Normal, Carcinoma e Cárie. A classe da doença hiperplasia fibrosa se mostrou com resultado ruim sendo classificada com cárie com taxa de confiança abaixo dos 50% pelo modelo, isso ocorreu pois os dados desta classe são escassos.

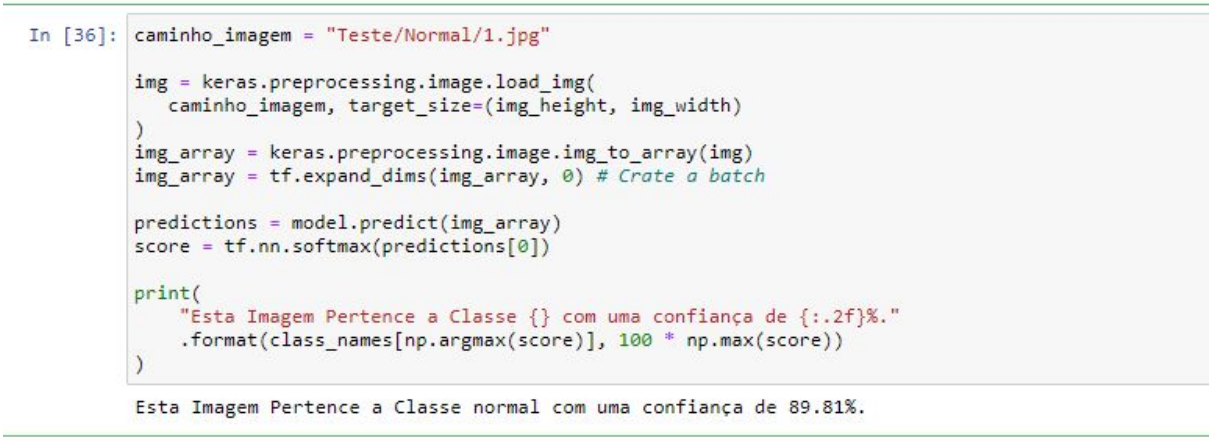

Figura 9: RESULTADO - Label Normal

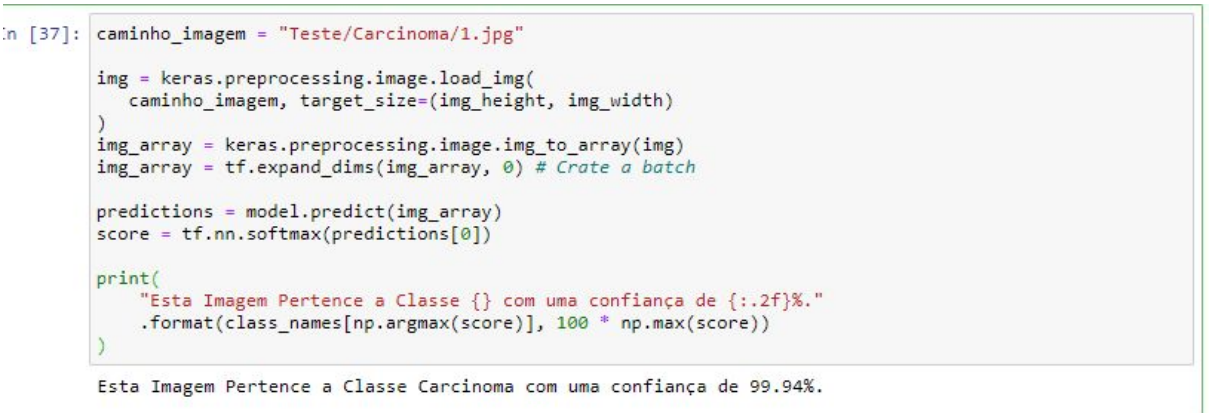

Figura 10: RESULTADO - Label Carcinoma

```
[38]: caminho_imagem = "Teste/Carie/1.jpg"

img = keras.preprocessing.image.load_img(
   caminho_imagem, target_size=(img_height, img_width)
)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Crate a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "Esta Imagem Pertence a Classe {} com uma confiança de {:.2f}%."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

```
Esta Imagem Pertence a Classe Carie com uma confiança de 98.46%.
```

Figura 11: RESULTADO - Label Cárie

```
In [39]: caminho_imagem = "Teste/Fibrosa/1.jpg"

img = keras.preprocessing.image.load_img(
   caminho_imagem, target_size=(img_height, img_width)
)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Crate a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "Esta Imagem Pertence a Classe {} com uma confiança de {:.2f}%."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

```
Esta Imagem Pertence a Classe Carie com uma confiança de 47.93%.
```

Figura 12: RESULTADO - Label Hiperplasia Fibrosa

7. Conclusão

Elaborado com o intuito de auxiliar os estudantes de odontologia e até mesmo os profissionais da área, o classificador conseguiu se sair bem, com resultados de até 98% de eficácia em algumas classes. Pelo motivo do data frame ter sido montado do zero com imagens do google, os resultados não se mostraram melhores tendo algumas classes sua classificação errada pelo modelo. A fim de melhorar o projeto, seguimos em busca de melhorar os nossos dados mesmo com a entrega deste relatório.

Referências

[1]
https://financial-engineering.medium.com/tensorflow-2-0-load-images-to-tensorflow-897b8b067fc2

[2]
https://www.tensorflow.org/tutorials/images/classification

[3]
Bonan, Perez e Melo. Diagnóstico Diferencial de Lesões Bucais na Clínica Odontológica