UNIVERSIDADE FEDERAL DO PARANÁ

SETOR DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA

ESPECIALIZAÇÃO EM INTELIGÊNCIA ARTIFICIAL APLICADA

Visão Computacional

Luís Sérgio da Silva Joppert

Detecção de leucemia baseado em imagens de leucócitos

Este projeto foi desenvolvido para o trabalho final da disciplina Visão Computacional, ministrada pelo Prof. Dr. Lucas Ferrari de Oliveira para o curso de Especialização em Inteligência Artificial Aplicada na Universidade Federal do Paraná.

Descrição do trabalho

“O arquivo trabalho.zip possui imagens de leucócitos na parte central. As imagens são nomeadas como "ImXXX_Y_Z.jpg". Onde ImXXX é o número da imagem, Y é o seu número da sequência de alteração (data augmentation) e Z a sua classe (0 ou 1). Onde, 0 indica paciente normal e 1 pacientes com leucemia. Utilizando técnicas de Visão Computacional e/ou CNNS extraia características das imagens e faça a sua correta classificação (0 ou 1). Lembre-se de separar os grupos de treinamento e teste. Você pode utilizar a técnica de k-folds para a divisão das imagens e evitar o overfitting.”

Passos para executar o código

É necessário ter um ambiente *Python* configurado em sua máquina para executar o código. Ele foi desenvolvido no *Python 3.8*, logo essa é a versão mais recomendada para a execução. Além do ambiente *Python* dê preferência por executar os códigos a seguir no *Jupyter Labs* (vide arquivo *leucemia_pt_BR.ipynb)*. Executando no *Jupyter Notebook* a saída do console aparecerá apenas no seu terminal *Python* e não no *Notebook*.

Além do ambiente serão necessárias algumas bibliotecas para o código:

- *Numpy* (https://numpy.org/) (1.19.0)
- *OpenCV* (https://opencv.org/) (4.5.1.48)
- *Tensorflow* (https://www.tensorflow.org/) (2.2)
- *Keras* (https://keras.io/) (2.3.1)
- *ScyPy* (https://www.scipy.org/) (1.6.0)

Caso seja necessário posso passar meu ambiente *venv* do *PyCharm* com estes pacotes já instalados.

A estrutura de pastas das imagens deve estar na seguinte hierarquia:

- Raiz
    - *AugmentedImages*
    - *Images*
    - *ModifiedAugmentedImages*
    - *ModifiedImages*
        - *Leukemia*
        - *Normal*

Além disso o código deve estar na pasta raiz para que seja executado corretamente.

Instalação das bibliotecas no terminal *Python* (PIP)

- *pip install numpy*
- *pip install opencv-python*
- *pip install tensorflow*
- *pip install matplotlib*
- *pip install SciPy*

Instalação das bibliotecas em ambiente *Conda* (Não testado)

- *conda install numpy*
- *conda install -c menpo opencv*
- *conda create -n tf tensorflow*
- *conda activate tf*
- *conda install -c anaconda scipy*

Apresentação do código

Inicialmente é necessário importar todos os pacotes que serão usados dentro do código.

```
import numpy as np
from numpy import expand_dims
import cv2
import os

import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing import image as kerasImage
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Então é criada uma classe para guardar os dados das imagens, como a imagem em questão, seu nome de arquivo e seu tipo, sendo 1 representando o leucócito com leucemia e 0 para o leucócito normal.

```
class Image:
    img = None
    name = ''
    type = ''

    def __init__(self, name, img, type):
        self.name = name
        self.img = img
        self.type = type
```

Com a classe criada agora é possível criar uma função que leia um diretório, pegue as imagens, seus nomes de arquivo e dentro do nome do arquivo extraia o tipo certo de cada imagem.

```
def load(folder):
    print("Loading images...")
    pictures = []
    for filename in enumerate(os.listdir(folder)):
        pictures.append(Image(filename[1],
        cv2.imread(os.path.join(folder, filename[1])), filename[1][-5]))
    return pictures
```

Tendo uma lista de imagens é possível processá-las e guardá-las em outro diretório para usá-las futuramente para a execução do treinamento. Para este processamento foi aplicado um borrão gaussiano leve para diminuir o ruído da imagem, e então foi aplicado um limite simples binário invertido. Este limite é aplicado em toda a imagem, com isso se o pixel for menor que o limite, o mesmo é passado para o valor máximo, do contrário é passado para zero. Então essa imagem binária é guardada como uma máscara e aplicada em cima da imagem original, visando isolar apenas o leucócito na parte central. Feito isso as imagens são separadas em duas pastas diferentes uma para os casos normais e outra para os casos de leucemia. É separado dessa forma para que mais adiante seja possível separar em classes.

```
def process_and_save(folder, pictures):
    print("Processing and saving images...")
    for index, image in enumerate(pictures):
        img = cv2.cvtColor(image.img, cv2.COLOR_BGR2GRAY)
        suave = cv2.GaussianBlur(img, (7, 7), 0)
        (T, binI) = cv2.threshold(suave, 105, 255, cv2.THRESH_BINARY_INV)
        subfolder = folder + 'Leukemia/' if image.type == '1' else folder + 'Normal/'
        cv2.imwrite(subfolder + image.name,
                    cv2.bitwise_and(img, img, mask=binI))
```

Além das imagens tratadas foi preparada uma função para gerar mais imagens usando aumento de dados, visto que o número de dados para trabalhar é pequeno e dentro dele já houve também um processo de aumento de dados. O objetivo dessa função é gerar imagens mais diferentes do que as que já foram geradas, usando uma rotação aleatória de 30 graus, permitindo uma inversão horizontal e variando o brilho e zoom na imagem. Além da mudança para gerar imagens novas, também foi aplicado o mesmo processamento da função anterior para padronizar as imagens. Feito o processamento as imagens são guardadas em outra pasta para validação futuramente.

```
def generate_augmented_images(folder, pictures):
    print("Generate augmented images...")
    datagen = ImageDataGenerator(rotation_range=30, horizontal_flip=True,
                                 brightness_range=[0.5, 1.0],
                                 zoom_range=[0.5, 1.0])
    for index, image in enumerate(pictures):
        samples = expand_dims(img_to_array(image.img), 0)
        augmented_image = datagen.flow(samples, batch_size=1).next()[0].astype('uint8')
        cv2.imwrite(folder + image.name, augmented_image)

        img = cv2.cvtColor(image.img, cv2.COLOR_BGR2GRAY)
        suave = cv2.GaussianBlur(img, (7, 7), 0)
        (T, binI) = cv2.threshold(suave, 105, 255, cv2.THRESH_BINARY_INV)
        cv2.imwrite('Modified' + folder + image.name,
                    cv2.bitwise_and(img, img, mask=binI))
```

Por fim as funções anteriores são compactadas em uma função especifica para preparar os dados para que o algoritmo em si classifique-os.

```
def prepare_data(images_folder, modified_images_folder, augmented_images_folder):
    print('Preparing data...')
    images = load(images_folder)
    process_and_save(modified_images_folder, images)
    generate_augmented_images(augmented_images_folder, images)
    print('Data prepared!')
```

Feita as implementações inicias, o código declara variáveis globais de pastas para facilitar as configurações caso seja necessário e inicia com a preparação dos dados apresentada anteriormente. Caso o projeto já tenha sido baixado com as imagens processadas, esta função não precisa ser executada, mas caso seja, só irá fazer todo o processamento novamente e substituirá os dados. Após isso são definidas as constantes para o tamanho das imagens a serem treinadas, o tamanho do lote a ser processado, a semente para os dados randômicos e o tamanho da separação dos dados de validação.

```
if __name__ == '__main__':

    images_folder = 'Images/'
    modified_images_folder = 'ModifiedImages/'
    augmented_images_folder = 'AugmentedImages/'
    modified_augmented_images_folder = 'ModifiedAugmentedImages/'

    prepare_data(images_folder, modified_images_folder,
                 augmented_images_folder)

    batch_size = 32
    img_height = 180
    img_width = 180
    seed = 123
    validation_split = 0.2
```

Então é feita a partição dos dados de treinamento usando as variáveis de configuração declaradas anteriormente. Também são salvos os nomes das classes que foram encontradas, para que seja possível comparar com os resultados ao validar o modelo. As classes são equivalentes as pastas em que os dados se encontram, por isso eles foram separados em duas pastas diferentes na preparação dos dados.

```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    modified_images_folder,
    validation_split=validation_split,
    subset="training",
    seed=seed,
    image_size=(img_height, img_width),
    batch_size=batch_size)

class_names = train_ds.class_names
print(class_names)
```

Da mesma forma que foi feita a partição de treinamento também é feita a partição de validação dos dados.

```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    modified_images_folder,
    validation_split=validation_split,
    subset="validation",
    seed=seed,
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

Então é configurada uma otimização para o processamento, guardando dados em cache para acelerar a consulta e possibilitando que os dados futuros possam ser preparados enquanto os dados atuais estão sendo executados.

```
train_ds = train_ds.cache().shuffle(1000)
.prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache()
.prefetch(buffer_size=tf.data.AUTOTUNE)
```

Com as partições separadas são montadas as camadas que terão no modelo de classificação. Neste caso tendo 3 camadas de convolução 2D utilizando a função de ativação RELU, 3 camadas de agregação, 1 camada de *dropout* para ajudar a evitar *overfitting*, 1 camada de nivelamento e 2 camadas de densidade.

```
model = Sequential([
    layers.experimental.preprocessing.Rescaling(1. / 255),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(2)
])
```

Em seguida o modelo é compilado utilizando o otimizador SGD, este usando a acurácia como parâmetro para definir os melhores resultados.

```
model.compile(optimizer='SGD',
              loss=tf.keras.losses.
              SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Por fim o modelo é executado com 20 épocas, visto que o número de dados é pequeno e muitas épocas tendem a viciar o modelo.

```
print('Executing model...')
epochs = 20
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)
print('Model done!')
```

Então valida o modelo com a partição de treinamento e o salva.

```
results = model.evaluate(train_ds, batch_size=128)

model.save('LeukemiaModel')
```

Por fim tenta fazer as predições do utilizando as imagens já tratadas geradas pela função de aumento de dados, e contabiliza os resultados comparando com o que foi obtido na validação anterior do modelo.

```
print('Predicting images...')
num_img, num_score = 0, 0
for index, filename in enumerate(
    os.listdir(modified_augmented_images_folder)):
    img = os.path.join(
        modified_augmented_images_folder, filename)
    img = kerasImage.load_img(
        img, target_size=(img_width, img_height))
    img = kerasImage.img_to_array(img)
    img = np.expand_dims(img, axis=0)

    image = Image(filename, img, filename[-5])

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

    if (class_names[np.argmax(score)] == "Leukemia"
        and image.type == '1') or (
            class_names[np.argmax(score)] == "Normal"
        and image.type == '0'):
        num_score += 1

    num_img += 1

print("Test loss: {:.2f}%; Test accuracy: {:.2f}%."
      .format(results[0] * 100, results[1] * 100))
print('This model had {:.2f}% accuracy on augmented images.'
      .format(0 if num_score == 0 else num_score * 100 / num_img))
```

Obtendo ao final 93.02% de acurácia na validação do teste e 91.21% de acurácia com as imagens geradas por aumento de dados.

```
Predicting images...
Test loss: 18.81%; Test accuracy: 93.02%.
This model had 91.12% accuracy on augmented images.
```