FEDERAL UNIVERSITY OF PARANÁ

PROFESSIONAL AND TECHNOLOGICAL EDUCATION SECTOR

SPECIALIZATION IN APPLIED ARTIFICIAL INTELLIGENCE

Computer Vision

Luís Sérgio da Silva Joppert

Leukemia detection based on leukocyte images

This project was developed for the final work of the Computer Vision course, taught by Prof. Dr. Lucas Ferrari de Oliveira for the Specialization Course in Applied Artificial Intelligence at the Federal University of Paraná.

Project description

“The trabalho.zip file has leukocyte images in the central part. The images are named as "ImXXX_Y_Z.jpg". Where ImXXX is the number of the image, Y is its number of the sequence of alteration (data augmentation) and Z its class (0 or 1). Where, 0 indicates a normal patient and 1 indicates leukemia. Using Computer Vision and/or CNNS techniques, extract characteristics from the images and make their correct classification (0 or 1). Remember to separate the training and testing groups. You can use the k-fold technique to divide the images and avoid overfitting.”

Steps to run the code

You need to have a Python environment set up on your machine to run the code. It was developed on Python 3.8, so this is the most recommended version to run. In addition to the Python environment, give preference to running this Notebook in Jupyter Labs. Running on Jupyter Notebooks the console output will only appear on your Python terminal and not on the Notebook.

Beside the environment, some libraries will be needed for the code:

- *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)

If necessary, I can pass my PyCharm venv environment with these packages already installed.

The folder structure of the images must be in the following hierarchy:

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

In addition, the code must be at the root folder for it to run correctly.

Installing libraries on the Python terminal (PIP)

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

Installation of libraries in Conda environment (not tested)

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

Code description

Initially it is necessary to import all packages that will be used within the code.

```
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
```

Then a class is created to store the image data, such as the image in question, its file name and its type, with 1 representing the leukocyte with leukemia and 0 for the normal leukocyte.

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

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

With the created class it is now possible to create a function that reads a directory, takes images, their file names and within the file name extract the right type of each image.

```
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
```

Having a list of images it is possible to process them and save them in another directory to use them in the future for the training execution. For this process a light Gaussian blur was applied to decrease the image noise, and then a simple inverted binary threshold was applied. This threshold is applied to the entire image, so if the pixel is smaller than the threshold, it is passed to the maximum value, otherwise it is passed to zero. Then this binary image is saved as a mask and applied on top of the original image, aiming to isolate only the leukocyte in the central part. Once it is done, the images are separated into two different folders, one for normal cases and another for leukemia cases. It is separated in this way so that it is possible to separate it further into 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))
```

In addition to the treated images, a function was prepared to generate more images using data augmentation, since the number of data to work with is small and within it there has also been a data augmentation process. The purpose of this function is to generate images that are more different than those that have already been generated, using a random 30 degrees rotation, allowing horizontal flip and varying the brightness and zoom in the image. Besides to the change to generate new images, the same process as the previous function was also applied to standardize the images. After processing, the images are saved in another folder for future validation.

```
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))
```

Finally, the previous functions are compacted in a specific function to prepare the data so that the algorithm itself classifies them.

```
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!')
```

Having made the initial implementations, the code declares global folder variables to facilitate the configurations if necessary and starts with the preparation of the data presented previously. If the project has already been downloaded with the processed images, this function does not need to be executed, but if it is, it will only do all the process again and replace the data. After that, the constants are defined for the size of the images to be trained, the size of the batch to be processed, the seed for the random data and the size for the separation of the validation data.

```
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
```

Then the training data is partitioned using the configuration variables declared earlier. The names of the classes that were found are also saved, so that it is possible to compare with the results when validating the model. The classes are equivalent to the folders in which the data are located, so they were separated into two different folders in the preparation of the data.

```
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)
```

In the same way that the training partition was made, so is the data validation partition.

```
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)
```

Then an optimization is configured for processing, caching data to speed up the query and enabling future data to be prepared while the current data is being executed.

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

With the separate partitions, the layers they will have in the classification model are assembled. In this case having 3 layers of 2D convolution using the RELU activation function, 3 pooling layers, 1 dropout layer to help prevent overfitting, 1 flatten layer and 2 density layers.

```
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)
])
```

Then the model is compiled using the SGD optimizer, which uses accuracy as a parameter to define the best results.

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

Finally, the model runs with 20 epochs, since the number of data is small and many epochs tend to overfit the model.

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

Then validate the model with the training partition and save it.

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

model.save('LeukemiaModel')
```

Finally, it tries to make the predictions using the treated images generated by the data augmentation function, and counts the results in comparison with what was obtained in the previous model validation.

```
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))
```

At the end obtaining 93.02% accuracy in the test validation and 91.21% accuracy with the images generated by data increase.

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