AutomaticTICI

May 20, 2019

1 Automatic TICI

1.1 Import modules

```
In [1]: # Import modules
        import numpy as np
        import matplotlib.pyplot as plt
        import os
        from scipy.io import loadmat
        from tensorflow import keras
```

1.2 Load data & extract images and TICI scores

```
In [2]: # Get the path of the data directory.
        data_dir = os.path.join(os.getcwd(), 'data')
        # Get a list of full paths of all mat files in the data directory.
        for root, _dirs, files in os.walk(data_dir):
            files = list(filter(lambda fname: fname.lower().endswith('.mat'), sorted(files)))
        nfiles = len(files)
        print('{} files found in the data directory \'{}\'.'.format(nfiles, data_dir))
```

```
201 files found in the data directory '/Users/vincentdong/Documents/College/UCLA/14 Spring 2019/
```

```
In [3]: # Given the data_dir and a file name, extract an image from
        # the set of images and the TICI score.
        # By default, nothing is printed.
        # If verbose=1, print the keys of the mat file content (which is a dictionary).
        # If verbose=2, print the 3 TICI scores for debugging purpose.
        def extract_data_file(data_dir, fname, verbose=False):
            content = loadmat(os.path.join(data_dir, fname))

            if verbose == 1:
                print('{}\tkeys={}'.format(fname, sorted(content.keys())))
            if verbose == 2:
                print('{}\t\tTICI_Dr1={}\tTICI_Dr2={}\tTICI_report={}'.format(fname, content['TI

            raw_image_set, TICI = content['X'], content['TICI_report']
            # Originally, raw_image_set[:, :, k] is the kth image.
            # Reorder the dimensions such that raw_image_set[k, :, :] is the kth image.
            image_set = np.transpose(raw_image_set, (2, 0, 1))
            # Only one image from each image set is selected to be fed into the model.
            # For simplicity, the image in the middle of each image set is selected just for now
            count, _, __ = np.shape(image_set)
            image = image_set[count // 2]
            return image, TICI
```

```
In [4]: # From the first mat file, learn the structure and the image set dimensions.
        # Assume that all mat files have the same structure and that all the images
        # in each image set have the same dimensions, though each image sets may
        # contain various number of images.
        sample_image, sample_TICI = extract_data_file(data_dir, files[0], verbose=1)
        image_shape = sample_image.shape
        print(image_shape)
```

```
fractals_1.mat        keys=['TICI_Dr1', 'TICI_Dr2', 'TICI_report', 'X', '__globals__', '__header
(1024, 1024)
```

```
In [5]: # For debugging purpose, print the 3 TICI scores for the first 10 mat files.
        for n in range(10):
            _, _ = extract_data_file(data_dir, files[n], verbose=2)
```

```
fractals_1.mat                TICI_Dr1=['2b']        TICI_Dr2=['2b']        TICI_report=['2a']
fractals_10.mat                TICI_Dr1=[[nan]]        TICI_Dr2=[[nan]]        TICI_report=['2a'
fractals_100.mat                TICI_Dr1=[[nan]]        TICI_Dr2=[[nan]]        TICI_report=[[3]
fractals_101.mat                TICI_Dr1=[[nan]]        TICI_Dr2=[[nan]]        TICI_report=[[0]
fractals_102.mat                TICI_Dr1=[[nan]]        TICI_Dr2=[[nan]]        TICI_report=['2b
fractals_103.mat                TICI_Dr1=[[nan]]        TICI_Dr2=[[nan]]        TICI_report=[[na
fractals_104.mat                TICI_Dr1=[[nan]]        TICI_Dr2=[[nan]]        TICI_report=['2b
fractals_105.mat                TICI_Dr1=[[nan]]        TICI_Dr2=[[nan]]        TICI_report=[[0]
fractals_106.mat                TICI_Dr1=[[nan]]        TICI_Dr2=[[nan]]        TICI_report=[[0]
fractals_107.mat                TICI_Dr1=[[nan]]        TICI_Dr2=[[nan]]        TICI_report=['2a
```

```
In [6]: # Initialize a list for images and a numpy arrays for
        # corresponding TICI scores. Assume all images have
        # the same dimensions. The TICI score in each
        images = []
        TICI_strings = []

        # Extract image and TICI information for all mat files.
        for n in range(nfiles):
            # Print the extracting progress.
            if n % 10 == 0:
                print('{} / {} done'.format(n, nfiles))
            image, TICI = extract_data_file(data_dir, files[n])
            images.append(image)
            # The TICI scores in the mat files are in the form of
            # nested np.ndarray's of either strings, numbers, of nan.
            # e.g., ['2a'], [[3]], [[nan]]. With assumption of
            # this structure, simplify TICI before append it to TICIs.
            while isinstance(TICI, np.ndarray):
                TICI = TICI[0] if len(TICI) > 0 else ''
            TICI_strings.append(str(TICI))

        print(np.shape(images))
        print(TICI_strings)
```

```
0 / 201 done
10 / 201 done
20 / 201 done
30 / 201 done
40 / 201 done
50 / 201 done
60 / 201 done
70 / 201 done
80 / 201 done
90 / 201 done
100 / 201 done
110 / 201 done
120 / 201 done
130 / 201 done
140 / 201 done
150 / 201 done
160 / 201 done
170 / 201 done
180 / 201 done
190 / 201 done
200 / 201 done
(201, 1024, 1024)
['2a', '2a', '3', '0', '2b', 'nan', '2b', '0', '0', '2a', '2b', '2a', '2a', '2b', '0', '2b', '2b
```

1.3 Reformat TICI scores

```
In [7]: # The number of different TICI scores.
        # Including 0, 1, 2a, 2b, 3, nan.
        num_TICI_classes = 6

        # Convert a TICI string to a number
        def map_TICI_str_to_num(TICI):
            relation = {
                '0': 0,
                '1': 1,
                '2a': 2,
                '2b': 3,
                '3': 4,
                'nan': 5,
                '0 (bilateral MCA)': 0,
                '2a?': 2
            }
            return relation[TICI]

        # Convert a numerical encoded TICI to a string
        def map_TICI_num_to_str(label):
            relation = ['0', '1', '2a', '2b', '3', 'nan']
            return relation[label]

        # Convert TICI scores in the form of strings to numeric labels before fed to the model.
        TICI_nums = list(map(map_TICI_str_to_num, TICI_strings))
        print(TICI_nums)

        # Convert the array of integer labels (0 ~ num_TICI_classes-1) to an array of
        # one-hot (aka one-of-K) encoded labels, for better accuracy.
        TICI_one_hot = keras.utils.to_categorical(TICI_nums, num_TICI_classes)
        print(TICI_one_hot)
```

```
[2, 2, 4, 0, 3, 5, 3, 0, 0, 2, 3, 2, 2, 3, 0, 3, 3, 0, 5, 4, 5, 3, 1, 3, 2, 0, 3, 3, 5, 0, 2, 0,
[[0. 0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0.]
 ...
 [0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0.]]
```

```
In [8]: # Build the model with tensorflow.keras.
        # The general idea is to reduce the size by maxpooling and
        # extract more features with convolutions of an increasing
        # number of filters.
        model = keras.Sequential([
            keras.layers.Conv2D(32, 5, padding='same', activation='relu',
                                input_shape=(image_shape[0], image_shape[1], 1)),  # learn why t
            keras.layers.MaxPooling2D(pool_size=(4, 4), strides=(4, 4), padding='same'),
            keras.layers.BatchNormalization(),
            keras.layers.Conv2D(64, 5, padding='same', activation='relu'),
            keras.layers.MaxPooling2D((4, 4), (4, 4), padding='same'),
            keras.layers.Flatten(),
            # keras.layers.Dense(1024, activation='relu'),
            keras.layers.Dropout(0.4),
            keras.layers.Dense(num_TICI_classes, activation='softmax')
        ])

        model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 1024, 1024, 32)    832
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 256, 256, 32)      0
_________________________________________________________________
batch_normalization_v2 (Batc (None, 256, 256, 32)      128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 256, 256, 64)      51264
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 262144)            0
_________________________________________________________________
dropout (Dropout)            (None, 262144)            0
_________________________________________________________________
dense (Dense)                (None, 6)                 1572870
=================================================================
Total params: 1,625,094
Trainable params: 1,625,030
Non-trainable params: 64
_________________________________________________________________
```

```
In [10]: model.compile(
             loss=keras.losses.categorical_crossentropy,
             optimizer='adam',
             metrics=['accuracy'])
```

```
In [11]: # Add one dimension to the images input for channels.
         images_as_model_input = np.array(images).reshape(nfiles, image_shape[0], image_shape[1]
         print(images_as_model_input.shape)
         model.fit(
             x=images_as_model_input,
             y=TICI_one_hot,
             batch_size=4, #################### TODO
             epochs=10, #################### TODO
             verbose=1, # progress bar
         )
```

```
(201, 1024, 1024, 1)
Epoch 1/10
201/201 [==============================] - 171s 853ms/sample - loss: 1.7601 - accuracy: 0.3781
```

```
Epoch 2/10
201/201 [==============================] - 164s 817ms/sample - loss: 1.5463 - accuracy: 0.3483
Epoch 3/10
201/201 [==============================] - 169s 842ms/sample - loss: 1.3301 - accuracy: 0.5025
Epoch 4/10
201/201 [==============================] - 163s 810ms/sample - loss: 1.0872 - accuracy: 0.5622
Epoch 5/10
201/201 [==============================] - 146s 729ms/sample - loss: 0.8285 - accuracy: 0.6816
Epoch 6/10
201/201 [==============================] - 145s 722ms/sample - loss: 0.6011 - accuracy: 0.7960
Epoch 7/10
201/201 [==============================] - 151s 750ms/sample - loss: 0.2965 - accuracy: 0.8856
Epoch 8/10
201/201 [==============================] - 141s 700ms/sample - loss: 0.3174 - accuracy: 0.8806
Epoch 9/10
201/201 [==============================] - 198s 986ms/sample - loss: 0.1931 - accuracy: 0.9502
Epoch 10/10
201/201 [==============================] - 181s 902ms/sample - loss: 0.0831 - accuracy: 0.9900
```

```
Out[11]: <tensorflow.python.keras.callbacks.History at 0x13f7a0e10>
```