MRIMath

Generated by Doxygen 1.8.15

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Namespace Documentation

3.1 DataHandler Namespace Reference

Classes

- class DataHandler

3.1.1 Detailed Description

Class designed to do all data handling and manipulation, ranging from dataloading to network preprocessing. As time goes on, some of this may be refactored, and some of this functionality is contingent on the data beir in a certain structure.

@author Daniel Enrico Cahall

3.2 EmailHandler Namespace Reference

Classes

- class EmailHandler

3.2.1 Detailed Description

Class designed to handle constructing and sending emails, usually in the context of notifying one or more recipients when a process has finished. Currently, the EmailHandler has the capability to send an email to one or more people as long as they are identified in the Address Book, and attach one or more files to the email

@author Daniel Enrico Cahall

3.3 GeneralNetwork Namespace Reference

3.3.1 Detailed Description

Created on Jan 12, 2018

@author: daniel

3.4 HardwareHandler Namespace Reference

Classes

- class HardwareHandler

3.4.1 Detailed Description

```
Class designed to handle all hardware related tasks, such as creating a threadpool or getting the number of co
I anticipate that this class will grow over time, but for the time being it handles all necessary hardware tas

@author Daniel Enrico Cahall
```

3.5 LoadAndTestModel Namespace Reference

Variables

- string **data_dir** = '/media/daniel/ExtraDrive1/Patient_Data_Images';
- **model** = load_model('/home/daniel/eclipse-workspace/MRIMath/Models/2018-01-25_20_55/model.h5');
- **dataHandler** = DataHandler()
- list **segments** = [ ]
- **patient_directory** = os.fsencode(dataHandler.getDirectoryFromIndex(i, data_dir))
- string **orig_data** = patient_directory + b'/Original_Img_Data'
- string **seg_data** = patient_directory + b'/Segmented_Img_Data'
- **img** = dataHandler.getImage(orig_data+b'/'+file)
- **patches** = dataHandler.derivePatches(img, 1)
- **x**
- **y**
- **patch** = patch.reshape(1,25, 25, 1)
- **pred** = model.predict(patch)
- **label** = np.argmax(pred)
- **fig** = plt.figure()
- **a** = fig.add_subplot(1,9,1)
- int **ind** = 2;

3.5.1 Detailed Description

```
Created on Nov 28, 2017

@author: daniel
```

3.6 TimerModule Namespace Reference

Classes

- class TimerModule

3.6.1 Detailed Description

```
Class designed to keep track of time and performance. Pretty simple and small, although more capability can be
if necessary.
@author Daniel Enrico Cahall
```

3.7 TrainModels Namespace Reference

Functions

- def **precision** (y_true, y_pred)

Variables

- **now** = datetime.now()
- **date_string** = now.strftime('%Y-%m-%d_%H_%M')
- **dataHandler** = DataHandler()
- **emailHandler** = EmailHandler()
- **hardwareHandler** = HardwareHandler()
- **timer** = TimerModule()
- tuple **input_img** = (dataHandler.n, dataHandler.n, 1)
- **model** = Sequential()
- string **data_dir** = '/coe_data/MRIMath/MS_Research/Patient_Data_Images'
- **training**
- **training_labels**
- **testing**
- **testing_labels**
- string **model_directory** = "/coe_data/MRIMath/MS_Research/MRIMath/Models/" + date_string
- int **num_epochs** = 15
- int **batchSize** = 64
- float **lrate** = 0.1
- float **momentum** = 0.9
- **sgd** = SGD(lr=lrate, momentum=momentum, nesterov=True)
- string **model_info_filename** = 'model_info.txt'
- **model_info_file** = open(model_directory + '/' + model_info_filename, "w")
- string **log_info_filename** = 'model_loss_log.csv'
- **log_info** = open(model_directory + '/' + log_info_filename, "w")
- **csv_logger** = CSVLogger(model_directory + '/' + log_info_filename, append=True, separator=',')
- **reduce_lr** = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=0.001)
- **G** = hardwareHandler.getAvailableGPUs()
- **parallel_model** = multi_gpu_model(model, G)
- **optimizer**
- **loss**
- **metrics**
- **epochs**
- **batch_size**
- **shuffle**
- **validation_data**
- **callbacks**
- string **message** = "Finished training network at " + str(datetime.now()) + '\n\n'
- **print_fn**

3.7.1 Detailed Description

```
Created on Jan 9, 2018

@author: daniel
```

Chapter 4

Class Documentation

4.1 DataHandler.DataHandler Class Reference

Public Member Functions

- def __init__ (self, tolerance=0.25, numPatches=10, n=25)

  The constructor for the datahandler class.
- def getImage (self, path)

  Reads an image from a filepath.
- def loadDataSequential (self, data_directory, start, finish)

  Loads patient images and segments sequentially, assuming you want to through a range of numbered patients.
- def loadDataParallel (self, data_directory, start, finish)

  Loads patient images and segments in parallel, assuming you want to through a range of numbered patients.
- def loadIndividualPatient (self, index, data_directory)

  Derives and labels patches from an individual patient.
- def getDirectoryFromIndex (self, index, data_directory)

  Constructs the patient directory string based on the index (based on current labeling scheme)
- def preprocessForNetwork (self)

  Preprocesses the data for the network by converting the list of patches and labels to a numpy array, and normalizing the patches.
- def deriveRandomPatch (self, patient_directory, img, file)

  Derives random patches from an image.
- def derivePatches (self, img, stepSize)

  Derives an individual patch from an image.
- def derivePatchFromSegments (self, patient_dir, x, y, img_num)

  Derives and labels the patches in the segment imiage.
- def getData (self)

  Acquires the data from the DataHandler after all data has been loaded and processed.
- def clearVectors (self)

  Clears the data and labels.

Public Attributes

- **X**
- **labels**

Static Public Attributes

- **lock** = threading.Lock()
- **manager** = Manager()
- list **X** = [ ]
- list **labels** = [ ]
- int **W** = 240
- int **H** = 240
- **hardwareHandler** = HardwareHandler()
- **tolerance** = None
- **numPatches** = None
- **n** = None
- list **patches** = [ ]
- **window** = img[y:y + self.n, x:x + self.n]

4.1.1 Constructor & Destructor Documentation

4.1.1.1 __init__()

```
def DataHandler.DataHandler.__init__ (
              self,
              tolerance = 0.25,
              numPatches = 10,
              n = 25 )
```

The constructor for the datahandler class.

Parameters

tolerance	the percentage of pixels in a patch that can be background (default 0.25)
numPatches	the number of patches to extract per image (default 10)
n	the dimensions of the patch to be taken from the image (default 25)

4.1.2 Member Function Documentation

4.1.2.1 derivePatches()

```
def DataHandler.DataHandler.derivePatches (
              self,
              img,
              stepSize )
```

Derives an individual patch from an image.

Parameters

img	the image to derive patches from
stepSize	the amount the sliding window shifts per iteration
file	the patient image number (e.g. img_1)

Returns

patches a list of all patches in the image

4.1.2.2 derivePatchFromSegments()

```
def DataHandler.DataHandler.derivePatchFromSegments (
              self,
              patient_dir,
              x,
              y,
              img_num )
```

Derives and labels the patches in the segment imiage.

Parameters

patient_dir	the specific patient directory (e.g. Patient_001_Data)
x	the starting point for columns (x-direction) for the patch
y	the starting point for rows (y-direction) for the patch
img_num	image number (e.g. img_1)

Returns

a boolean flag which states if a label for the segment was suceesfully found

4.1.2.3 deriveRandomPatch()

```
def DataHandler.DataHandler.deriveRandomPatch (
              self,
              patient_directory,
              img,
              file )
```

Derives random patches from an image.

Parameters

patient_directoy	the directory where the specific patient data is located (e.g. Patient_001_Data)
img	the image to derive patches from
file	the patient image number (e.g. img_1)

4.1.2.4 getData()

```
def DataHandler.DataHandler.getData (
              self )
```

Acquires the data from the DataHandler after all data has been loaded and processed.

Parameters

img_num	image number (e.g. img_1)

Returns

the data and the labels for the loaded and processed data

4.1.2.5 getDirectoryFromIndex()

```
def DataHandler.DataHandler.getDirectoryFromIndex (
              self,
              index,
              data_directory )
```

Constructs the patient directory string based on the index (based on current labeling scheme)

Parameters

index	the index of the patient that you need the specific directory for
data_directory	Directory where all patient data is located

4.1.2.6 getImage()

```
def DataHandler.DataHandler.getImage (
              self,
              path )
```

Reads an image from a filepath.

Parameters

path	the path to an image file

Returns

the image from the filepath (if one existed) as a numpy array

4.1.2.7 loadDataParallel()

```
def DataHandler.DataHandler.loadDataParallel (
              self,
              data_directory,
              start,
              finish )
```

Loads patient images and segments in parallel, assuming you want to through a range of numbered patients.

Parameters

data_directory	the directory where all patient data is located
start	the patient number to start with (inclusive)
finish	the patient number to stop at (exclusive)

4.1.2.8 loadDataSequential()

```
def DataHandler.DataHandler.loadDataSequential (
              self,
              data_directory,
              start,
              finish )
```

Loads patient images and segments sequentially, assuming you want to through a range of numbered patients.

Parameters

data_directory	the directory where all patient data is located
start	the patient number to start with (inclusive)
finish	the patient number to stop at (exclusive)

4.1.2.9 loadIndividualPatient()

```
def DataHandler.DataHandler.loadIndividualPatient (
              self,
              index,
              data_directory )
```

Derives and labels patches from an individual patient.

Parameters

data_directory	the directory where all patient data is located
index	index of the patient in the numbered directory

The documentation for this class was generated from the following file:

- DataHandler.py

4.2 EmailHandler.EmailHandler Class Reference

Public Member Functions

- def __init__ (self)

  The constructor for the emailHandler class.
- def prepareMessage (self, subject, body)

  Prepares the message to be sent by setting up the subject and body.
- def connectToServer (self)

  Connects to the gmail server and logs in using the mrimathnotifier gmail address.
- def sendMessage (self, recipients)

  Sends the email to all desired recipients as long as they are within the address book.
- def finish (self)

  Clears the body of the email, the attached files, and the list of recipients, and disconnects from the gmail server.
- def attachFile (self, file, filename)

  Attaches a file to the email.

Public Attributes

- **msg**
- **body**
- **server**

Static Public Attributes

- string **addr** = "mrimathnotifier@gmail.com"
- string **password** = "mrimathpw"
- dictionary **addressBook**
- string **body** = ""

4.2.1 Constructor & Destructor Documentation

4.2.1.1 __init__()

```
def EmailHandler.EmailHandler.__init__ (
              self )
```

The constructor for the emailHandler class.

This creates the message and sets the from address to the mrimathnotifier@gmail.com

4.2.2 Member Function Documentation

4.2.2.1 attachFile()

```
def EmailHandler.EmailHandler.attachFile (
              self,
              file,
              filename )
```

Attaches a file to the email.

Parameters

file	the file to attach to the email
filename	the name of the file to attach (may not be necessary actually...)

4.2.2.2 prepareMessage()

```
def EmailHandler.EmailHandler.prepareMessage (
              self,
              subject,
              body )
```

Prepares the message to be sent by setting up the subject and body.

Parameters

subject	Subject line of the email to be sent
body	Body of the email to be sent

4.2.2.3 sendMessage()

```
def EmailHandler.EmailHandler.sendMessage (
            self,
            recipients )
```

Sends the email to all desired recipients as long as they are within the address book.

Parameters

recipients	a list of the names of desired recipients which have their emails linked in the address book map

4.2.3 Member Data Documentation

4.2.3.1 addressBook

```
dictionary EmailHandler.EmailHandler.addressBook  [static]
```

Initial value:

```
=  {'Danny': 'danielenricocahall@gmail.com',
        'Dr.Bouaynaya': 'bouaynaya@rowan.edu',
        'Oliver': 'palumb48@students.rowan.edu',
        'Dimah': 'derad6@rowan.edu',
        'Alena': 'alenagusevarus@gmail.com',
        'Dr.Hassan': 'hfshaykh@uabmc.edu',
        'Dr.Rasool': 'rasool@rowan.edu'}
```

The documentation for this class was generated from the following file:

- EmailHandler.py

4.3 HardwareHandler.HardwareHandler Class Reference

Public Member Functions

- def __init__ (self)

  The constructor for the hardwarehandler class.
- def getAvailableGPUs (self)

  Acquires the number of GPUs available to use.
- def getNumberOfCores (self)

  Acquires the number of CPU cores to use.
- def createThreadPool (self, threadCount=None)

  Creates a threadpool, where the number of threads is the number of available cores by default.

Public Attributes

- **numThreads**

Static Public Attributes

- **pool** = None

4.3.1 Constructor & Destructor Documentation

4.3.1.1 __init__()

```
def HardwareHandler.HardwareHandler.__init__ (
              self )
```

The constructor for the hardwarehandler class.

Sets the default number of threads to the number of available cores

4.3.2 Member Function Documentation

4.3.2.1 createThreadPool()

```
def HardwareHandler.HardwareHandler.createThreadPool (
              self,
              threadCount = None )
```

Creates a threadpool, where the number of threads is the number of available cores by default.

Parameters

threadCount	the number of threads to use (default is number of cores)

Returns

pool the threadpool which was created

4.3.2.2 getAvailableGPUs()

```
def HardwareHandler.HardwareHandler.getAvailableGPUs (
              self )
```

Acquires the number of GPUs available to use.

Returns

the available number of GPUs on the device (int)

4.3.2.3 getNumberOfCores()

```
def HardwareHandler.HardwareHandler.getNumberOfCores (
              self )
```

Acquires the number of CPU cores to use.

Returns

the number of cores on the device (int)

The documentation for this class was generated from the following file:

- HardwareHandler.py

4.4 TimerModule.TimerModule Class Reference

Public Member Functions

- def startTimer (self)
  Starts the timer.
- def stopTimer (self)
  Stops the timer.
- def getElapsedTime (self)
  Computes the amount of time elapsed based on when the timer started and stopped.

Public Attributes

- **start_time**
- **stop_time**

Static Public Attributes

- float **start_time** = 0.0
- float **stop_time** = 0.0

4.4.1 Member Function Documentation

4.4.1.1 getElapsedTime()

```
def TimerModule.TimerModule.getElapsedTime (
              self )
```

Computes the amount of time elapsed based on when the timer started and stopped.

Returns

the amount of time between the time being started and stopped

The documentation for this class was generated from the following file:

- TimerModule.py

Index