Convolutional Neural Network for Binary Classification for the Diagnosis of Alzheimer's Disease

Jake Billings, Gunnar Enserro

University of Colorado Denver
STEM School Highlands Ranch

ABSTRACT

A form of machine learning known as a convolutional neural network is used to analyze open-access MRI brain scans to diagnose Alzheimer's disease. Open access cross-section gif images from the Open Access Series of Imaging Studies (OASIS) are first preprocessed by cropping to the brain area then scaled to the mean size of each cross section type. The data is then divided into training and test using a pseudorandom 80/20 split. The machine learning library Keras is then used to design and train a convolutional neural network consisting of three convolutional layers and one deeply connected layer. The relu activation function is used at each node. After fifty epochs of training, the network was able to recognize AD brains with approximately 95% accuracy.

https://github.com/jake-billings/research-alzheimers

INTRODUCTION

Machine Learning/Convolutional Neural Networks

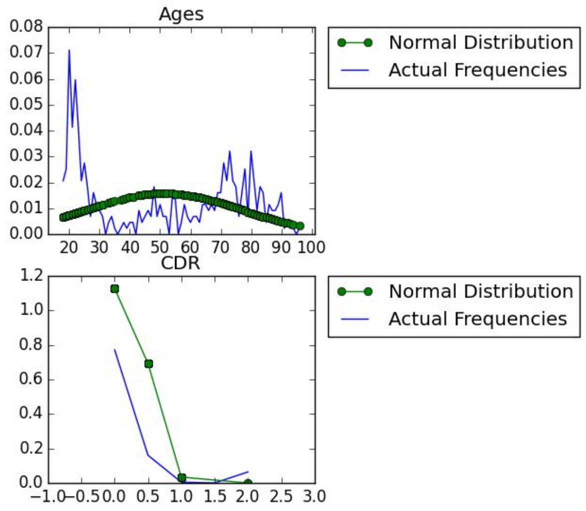
Machine learning is an field of computer science that aims to teach computers to solve very difficult problems in a similar manner to how humans learn in classrooms. Using machine learning, a programmer can provide a set of examples that follow a pattern to produce a set of outputs. Without any knowledge of the underlying pattern, machine learning algorithms seek to extrapolate the pattern from data. Machine learning is often applied in situations where it would be too complex for any human programmer to write all of the logic necessary to solve a problem. Convolutional neural networks (CNNs) have proven useful for the classification of images by the identification of features that are distinct to certain objects. They are inspired by the biological function of neurons found in the brain. Many layers of interconnected neurons work together to produce a result. CNNs have proven effective in many situations where many had previously believed computers would never succeed.

Alzheimer’s Disease

Alzheimer’s Disease is a progressively degenerative neurological disease that severely affects the mental ability of patients suffering from it. It is the highest cause of dementia, a chronic or persistent disorder that is the reason for impaired mental abilities. This includes memory, personality, and reasoning. Alzheimer's is known to affect the age demographic of 40 - 60. It is most heavily seen in ages above 60. Symptoms of Alzheimer's is Lost of memory overtime, this includes misplacing things on a regular bases to forgetting the names of family members. The severity of Alzheimer's can be quantified using the clinical dementia rating (CDR). CDR ranges from 0, which is indicates no impairment, to 5 which indicates very severe decline in mental abilities. The cause for Alzheimer's has been linked to past head trauma, age, and genetics. These different factors cause death of brain cells and tissue lost. There is currently no cure for Alzheimer’s. However, early identification can aid other treatment options.

Data Acquisition

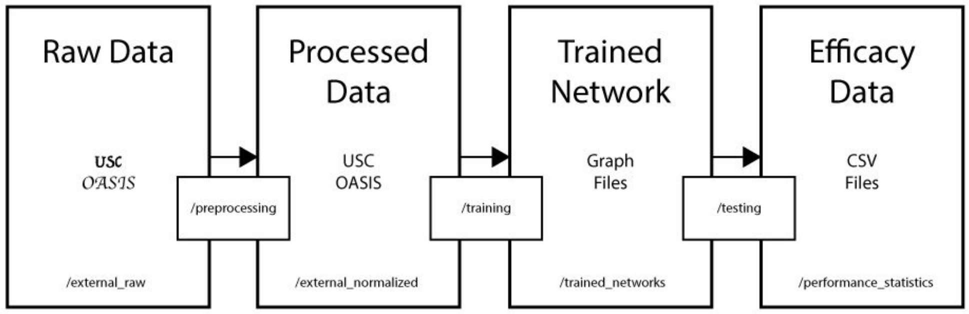
In order to begin training a network to classify images, it was necessary to acquire large sets of MRI scan data that had been pre-classified as suffering from Alzheimer's Disease or healthy. The Open Access Series of Imaging Studies (OASIS) is "a project aimed at making MRI data sets of the brain freely available to the scientific community." In this study of machine learning, the "Cross-sectional MRI Data in Young, Middle Aged, Nondemented and Demented Older Adults" dataset was used. The set provides MRI scans of 416 subjects aged 18 to 96 gendered both male and female. All data has been publically released at http://www.oasis-brains.org/. All subjects from OASIS are right-handed.



The distributions of age and clinical dementia rating (CDR) in the OASIS dataset do not follow a normal distribution. This should be noted when analyzing the results of the neural network. Further study is necessary to ensure that the network is not detecting signs of aging in the brain that correlate with the appearance of Alzheimer's disease as opposed to signs of the disease itself.

METHODS

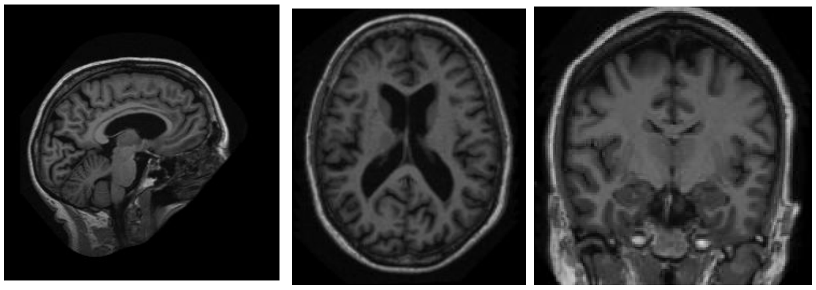
Software Architecture



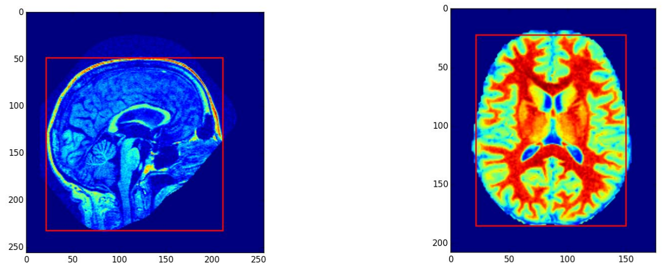
Software for the project was divided into three categories. Each one transformed one of four types of data to the next. A majority of the actual Python code written was to preprocess the raw data from the OASIS data set. The processed data was then used by code written for the Keras library to train a network. This resulted in h5 files that stored the weighted graphs and training data from the network. The efficacy data was then recorded as csv.

Pre-processing

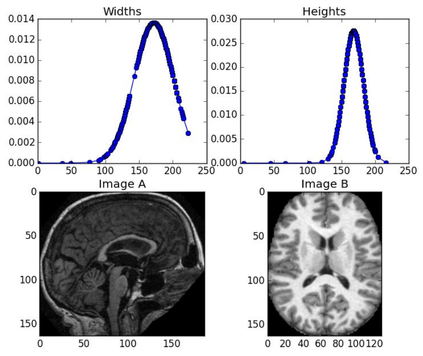
Raw data from the OASIS dataset consisted of grayscale gif images sized ~256px x ~256px. Images were not centered in the box and were not of consistent size, so algorithms were used to normalize images for neural networks. Each set of images had a constant width/height by orientation. Additionally, facially identifying features were removed. The size and positioning of the brain within each image was not constant. A black border surrounded each image.



In an effort to improve the accuracy and consistency of feature detection in the neural network, images from the OASIS database were pre-processed into a constant-width/constant-height format such. This was done to eliminate the scale/position of the brain within the image as a variable for the network to have to consider. Additionally, the image scaling resulted in the removal of scaling differences between individuals. All brains had a constant width, height, and position. Thus, anatomically similar features in each individual remained similar in training and test images regardless of scale and positioning.

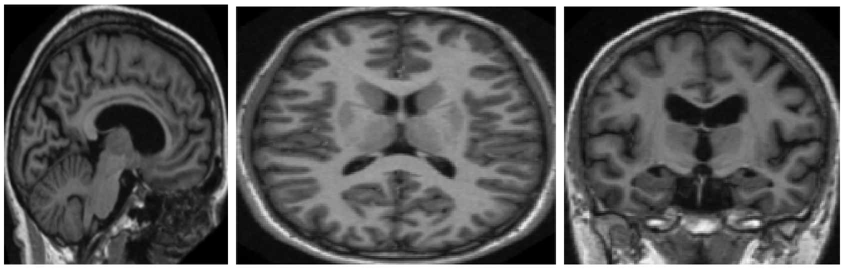


Python scripts were used to detect anatomical boundaries in images. Then, the borders were cropped off of each image. This resulted in a set of images without unused borders. In all images, brain information filled the entire image.

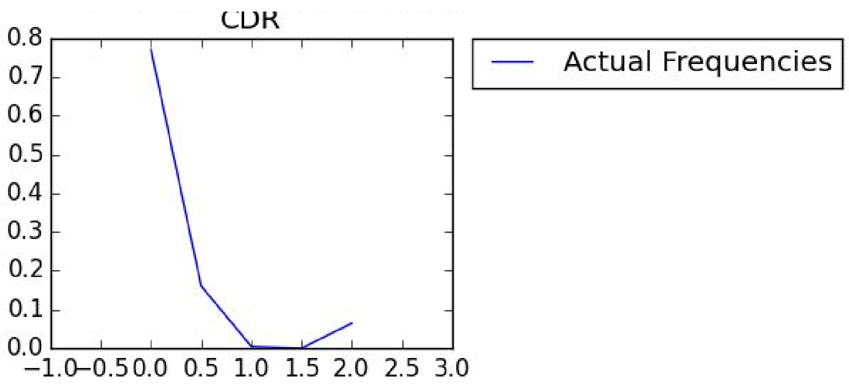


Section Type	Width	Height
COR	Mean: 155.82 px St. Dev.: 6.37 px	Mean: 166.97 px St. Dev.: 14.43 px
SAG	Mean 193.78 px St. Dev.: 8.20 px	Mean: 166.70 px St. Dev.: 10.74 px
TRA	Mean: 154.89 px St. Dev.: 9.14 px	Mean: 191.09 px St. Dev.: 7.06 px

The widths and heights of each image varied. In order to make images more consistent for ingestion into neural networks, the images were resized to the mean of dimensions of their respective image category within their datasets.



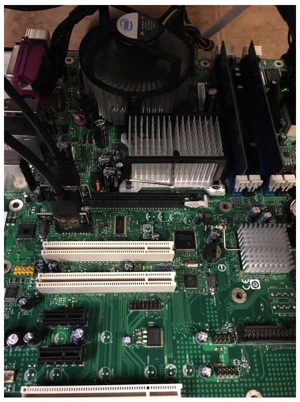
Results were then sorted by cross-section type and scaled to the mean dimensions of that type. Resulting images were all of the same sizes for each cross-section type and possessed no unused border space.



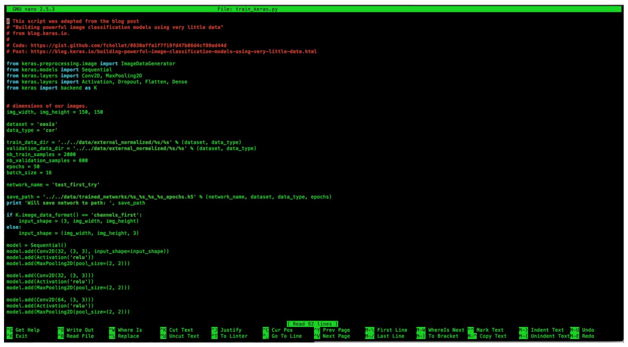
At this point, data from the index csv file was used to sort images into AD positive and AD negative categories. The clinical dementia rating (CDR) Patients with a blank CDR were assumed to have CDR 0. Patients with a CDR or 0 were considered AD negative. Patients with a positive CDR were considered to be AD positive. The positive/negative results for each category of cross-sectional image were then further divided into train/test data randomly using and 80% probability that images would be used as training data and a 20% probability that images would be used as test/validation data.

Hardware

Due its processing-intensive nature, a repurposed GPU-heavy Ethereum mining rig was used for the training of the neural network. Two motherboards, two CPUs, and two power supplies were used to manage four AMD rx580 GPUs.



One of the computers used to train the networks used a recycled motherboard that was connected to its two AMD rx580 GPUs with GPU riser cables. This enabled more efficient cooling of the system via the use of a large fan.



```
  GNU nano 2.5.3                    File: train_keras.py

# This script was adapted from the blog post
# "Building powerful image classification models using very little data"
# from blog.keras.io.
#
# Code: https://gist.github.com/fchollet/0830affa1f7f19fd47b06d4cf89ed44d
# Post: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras import backend as K


# dimensions of our images.
img_width, img_height = 150, 150

dataset = 'oasis'
data_type = 'cor'

train_data_dir = '../../data/external_normalized/%s/%s' % (dataset, data_type)
validation_data_dir = '../../data/external_normalized/%s/%s' % (dataset, data_type)
nb_train_samples = 2000
nb_validation_samples = 800
epochs = 50
batch_size = 16

network_name = 'test_first_try'

save_path = '../../data/trained_networks/%s_%s_%s_%s_epochs.h5' % (network_name, dataset, data_type, epochs)
print 'Will save network to path: ', save_path

if K.image_data_format() == 'channels_first':
    input_shape = (3, img_width, img_height)
else:
    input_shape = (img_width, img_height, 3)

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

                                   [ Read 92 lines ]
```

Ubuntu 16.04 was used to to host the Python configurations while training the neural network, and git was used to manage the synchronization of code and trained weight files.

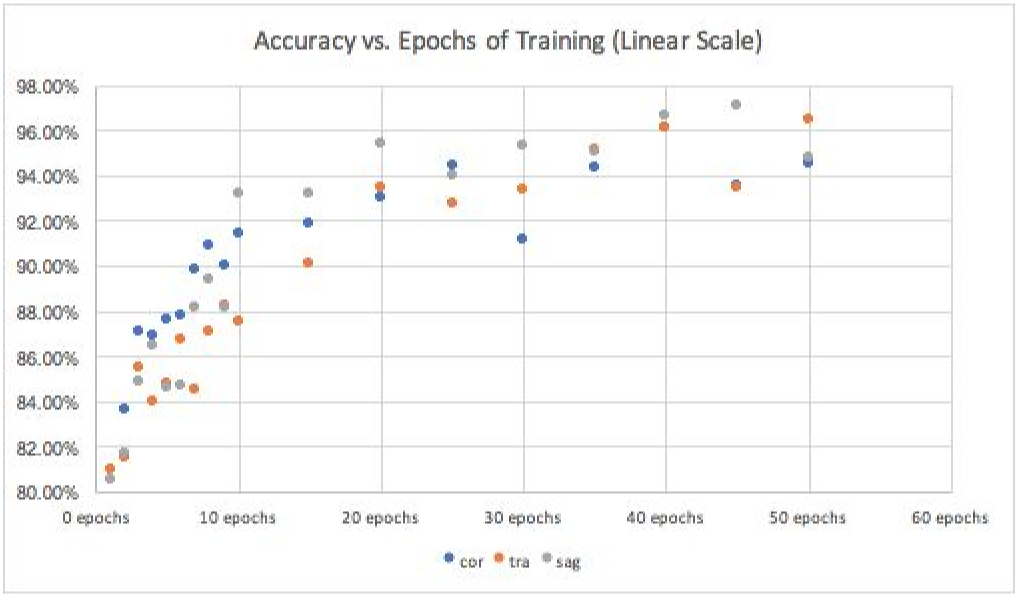
Training

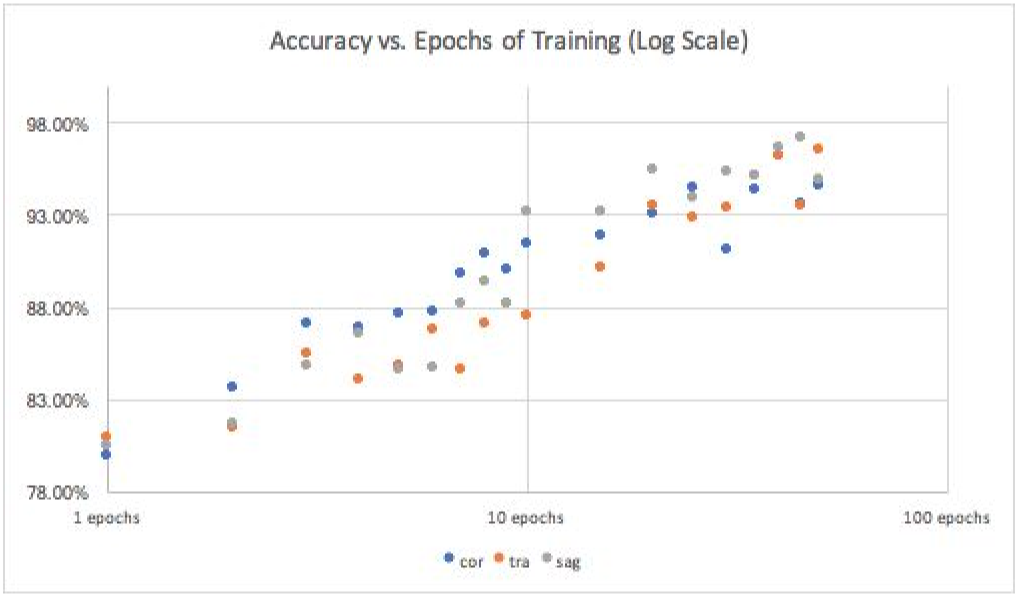
```
# This script was adapted from the blog post
# "Building powerful image classification models using very little data"
# from blog.keras.io.
#
# Code: https://gist.github.com/fchollet/0830affa1f7f19fd47b06d4cf89ed44d
# Post: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
        optimizer='rmsprop',
        metrics=['accuracy'])

train_datagen = ImageDataGenerator(
  rescale=1. / 255,
  shear_range=0.2,
  zoom_range=0.2,
  horizontal_flip=True)
```

Once the dataset was structured, a basic neural network was created using an example use of the Keras library from the Keras blog. This network consisted of three convolutional layers followed by one densely-connected layer. The relu activation function was used. A 50% dropout rate was used. Binary cross-entropy was used a loss function, and the network was optimized using the rmsprop algorithm. Accuracy was measured after each batch of optimization runs using the test dataset. The design of the network is an area where this research could be expanded. Differently designed networks may provide results with higher accuracy.

The weights used in each network were stored for 1, 10, and 50 epochs (batches) of processing for each cross-sectional image type. The efficacy on validation data was measured after each epoch for epochs 1-10 and every 5 epochs afterwards until the 50th epoch.

RESULTS





Within only 50 epochs, which translates to ~9 compute hours of training, the neural network was able to classify the images from the test/validation set with accuracy ~95%. In conclusion, convolutional neural networks appear to be an effective means of detecting Alzheimer's disease in MRI scans. However, more research is necessary to confirm this results and to ensure the data has not been over-fitted.

ACKNOWLEDGEMENTS

Special thanks to Shaun Keys for the use of his bandwidth to assist in downloading the OASIS dataset.

Special thanks to OASIS dataset for making this project possible.

CITATIONS

Buckner, Randy. Open Access Series of Imaging Studies (OASIS): a project aimed at making MRI data sets of the brain freely available to the scientific community. http://www.oasis-brains.org/app/template/Index.vm. P50 AG05681, P01 AG03991, R01 AG021910, P20 MH071616, U24 RR021382.

Chollet, François. “Classifier_from_little_data_script_1.py” 05 June 2016. https://gist.github.com/fchollet/0830affa1f7f19fd47b06d4cf89ed44d

Chollet, François. “Building powerful image classification models using very little data.” 05 June 2016. https://blog.keras.io/building-powerful-image-classification-models-using-very-little data.html

APPENDIX

Experiment_oasis_patient_data.py

This file was used to plot the age and CDR data as well as a normal curve for each dataset.

```
from numpy import mean
from numpy import std
from matplotlib import pyplot as plt
from scipy.stats import norm
from preprocessing.oasis_subject import deserialize_bulk_from_csv_file

INDEX_SOURCE = '../../data/external_raw/csv/oasis_cross-sectional.csv'

subjects = deserialize_bulk_from_csv_file(INDEX_SOURCE)

rv = norm()

def draw_plot_for_freq_data(name, data, fig, subplot, _min, _max, overscan=5, step=1, show_normal=True):
  fit = norm.pdf(data, mean(data), std(data))

  xvals = [x * step for x in range(_min, int(_max/step)+1)]

  freq = {}
  for i in xvals:
    freq[i] = 0
  for i in xvals:
    for datum in data:
      if datum == i:
        freq[i] += 1
  for i in xvals:
    freq[i] /= float(len(ages))

  freq_arr = freq.values()

  plt.figure(fig)
  plt.subplot(subplot)
  plt.title(name)

  actual, = plt.plot(xvals, freq_arr, label='Actual Frequencies')
  handles = [actual]

  if show_normal:
    normal, = plt.plot(data, fit, '-o', label='Normal Distribution')
    handles.append(normal)

  plt.xlim([_min - overscan, _max + overscan])
  plt.legend(handles=list(reversed(handles)), bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

# Age
ages = sorted([subject.age for subject in subjects])
draw_plot_for_freq_data('Ages', ages, 101, 221, int(min(ages)), int(max(ages)))

# CDR
cdrs = sorted([subject.cdr for subject in subjects])
draw_plot_for_freq_data('CDR', cdrs, 101, 223, int(min(cdrs)), int(max(cdrs)), overscan=1, step=0.5)

plt.show()
```

Oasis_crop_images.py

This script was used to load and crop the OASIS gifs images down to the size of the brain image data. See common.py and common_image_bounds for more details on the algorithms used to do this.

```
# Module-required imports
from common import list_files_in_subdirectories_by_extension as list_files
from common_image_bounds import get_bounds_for_image
from skimage import io

# Demo-required imports
import matplotlib.pyplot as plt

# todo build command line interface for this
SOURCE = '../../data/external_raw/gifs'
EXTENSION = '.gif'
DEST = '../../data/external_raw/gifs_cropped'
SAVE_FORMAT = 'png'
SAVE_EXTENSION = '.' + SAVE_FORMAT
THRESHOLD = 100


# Loads image from a path and returns a cropped scikit-image (2D matrix indexed [y, x])
# that is cropped to the area that contains data
#
# path The path of the image to read and crop
def load_and_crop_image(path):
  # Read the image data
  image = io.imread(path)

  # Calculate the bounds
  left, bottom, right, top = get_bounds_for_image(image, threshold=THRESHOLD)

  # Crop the image
  cropped_image = image[bottom:top, left:right]

  # Return the image
  return cropped_image


# Crops all images in the oasis data set based on the global SOURCE and DEST parameters
def crop_oasis_images():
  # Get the list of files from the source data
  files = list_files(SOURCE, EXTENSION)

  i = 0

  # Iterate over each file path
  for f in files:

    # Load and crop the image at path f
    try:
      print i, f

      cropped_image = load_and_crop_image(f)

      # Assign a path for saving. We need to move it from source to dest,
            # so do some path replacement
            save_path = f.replace(SOURCE, DEST).replace(EXTENSION, SAVE_EXTENSION)

            i += 1

            # Save the new cropped image
            plt.imsave(save_path, cropped_image, cmap=plt.cm.grey, format=SAVE_FORMAT)
        except Exception:
            print 'Problem cropping', f


# Crops all images in the oasis data set based on the global SOURCE and DEST parameters
def demo_plot_cropped_image():
    # Get the list of files from the source data
    files = list_files(SOURCE, EXTENSION, result_cap=1)

    # Crop the first image
    cropped_image = load_and_crop_image(files[0])

    # Plot the image using matplotlib
    plt.imshow(cropped_image)
    plt.show()


if __name__ == '__main__':
    crop_oasis_images()
```

Common_image_bounds.py

This library-style code is used to detect the bounds of the non-zero data in images and crop them to size.

```
# These imports are actually needed
from common import list_files_in_subdirectories_by_extension as list_files

# These imports are need only for the demo plot
# Uncomment the following two lines for a better windowing experience on mac
# import matplotlib
# matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from skimage import io  # used in demo_bounds()

# Global data location variables
SOURCE = '../../data/external_raw/gifs'
DEST = '../../data/in_progress'
EXTENSION = '.gif'


# Returns the bounds of an image inside of which there is actual data
#
# image The scikit-image image to analyze. This can be any 2D array indexed as [y, x]
# left, bottom, right, top The bounds of the image
#  Note: bottom is the lowest y value at which there is data, and top is the highest y value. Typically when
displaying
#  an image, low y values are the top and high y values are the bottom, so these bounds may be considered
"upside down"
# threshold The integer threshold to use when deciding if a pixel is within the useful bounds of an image
def get_bounds_for_image(image, threshold=0):
   left = len(image)
   right = 0
   top = 0
   bottom = len(image[0])

   for x in range(0, len(image[0])):
      for y in range(0, len(image)):
         if image[y, x] > threshold:
            if x > right:
               right = x
            if y > top:
               top = y
            if x < left:
               left = x
            if y < bottom:
               bottom = y

   return left, bottom, right, top


# Displays a matplotlib plot of an image surrounded by bounds within which there is data
#
# image The scikit-image image to analyze. This can be any 2D array indexed as [y, x]
def show_plot_with_bounds_for_image(image):
   left, bottom, right, top = get_bounds_for_image(image, threshold=50)
```

```
    xy = (left, bottom)
    width = right - left
    height = top - bottom

    plt.imshow(image)

    current_axis = plt.gca()

    current_axis.add_patch(Rectangle(xy, width, height, edgecolor="red", facecolor="none", linewidth=2))

    plt.show()


# Demos the bounds function by applying it to the first image in the SOURCE dataset
def demo_bounds():
    files = list_files(SOURCE, EXTENSION, result_cap=100)
    i = 0
    image = io.imread(files[i])
    plt.figure(i)
    show_plot_with_bounds_for_image(image)


if __name__ == '__main__':
    demo_bounds()
```