Alzheimer's Analysis with Convolutional Neural Networks

Marcia Hon

Dr. Naimul Khan

Results

Monday July 24, 2017

GitHub Link: https://github.com/marciahon29/Ryerson_MRP

Introduction

Neuroimaging is a rapidly growing field particularly attracted by data scientists. It is a very important field to society as it can result in accurate and timely prediction of diseases such as Alzheimer's. In this project, Alzheimer's Analysis is achieved using Convolutional Neural Networks. Imaging data is obtained from http://www.oasis-brains.org/ . Additionally, the Convolutional Neural Network used is VGG16 that is fine-tuned to work with the Alzheimer's data. In this report, a detailed explanation is provided for Data Preparation, Environment Preparation, Algorithm, Results, Conclusions, and Suggestions for Further Exploration. The GitHub link of all my data and results is provided above.

Data Preparation

MRI images of both those diagnosed with or without Alzheimer's were obtained from www.oasis-brains.org . 32 images for each patient was sorted then, based on entropy, the top 32 were extracted. Alzheimer's was determined by the "CRD (Clinical Dementia Rating)" variable:

- 0 = nondemented
- 0.5 = very mild dementia
- 1 = mild dementia
- 2 = moderate dementia

The assumption for this project is that a 0 implies non-Alzheimer's and those greater than 0 are Alzheimer's. There are a total of 6400 images classified equally as either Alzheimer's or non-Alzheimer's, thus meaning 3200 for Alzheimer's and 3200 for non-Alzheimer's.

Also, 5-fold (80% train and 20% test) was used thus each bucket has 640 images. 4 buckets together for train has 2560 images for each class and 1 bucket of 640 images for test images for each class. Images were randomly distributed into these 5 buckets.

In GitHub, please go to https://github.com/marciahon29/Ryerson_MRP/tree/master/MRI_32_Images . Here are the zipped images and bash scripts: 5_fold_alzheimers.sh, 5_fold_nonalzheimers.sh, and 5_folds_create_data.sh . These scripts randomly distributed the images into 5-fold. Additionally, a specific folder structure and naming convention must be used in order to work with the algorithm. This folder structure is as follows ("YAL" = Yes Alzheimers, "NAL" = No Alzheimers):

```
data_##:
     train
        alzheimers
           YAL0001.jpg
           YAL0002.jpg
           etc...
        nonalzheimers
           NAL0001.jpg
           NAL0002.jpg
           etc...
     validation
        alzheimers
           YAL0001.jpg
           YAL0002.jpg
           etc...
        nonalzheimers
           NAL0001.jpg
           NAL0002.jpg
           etc...
```

Environment Preparation

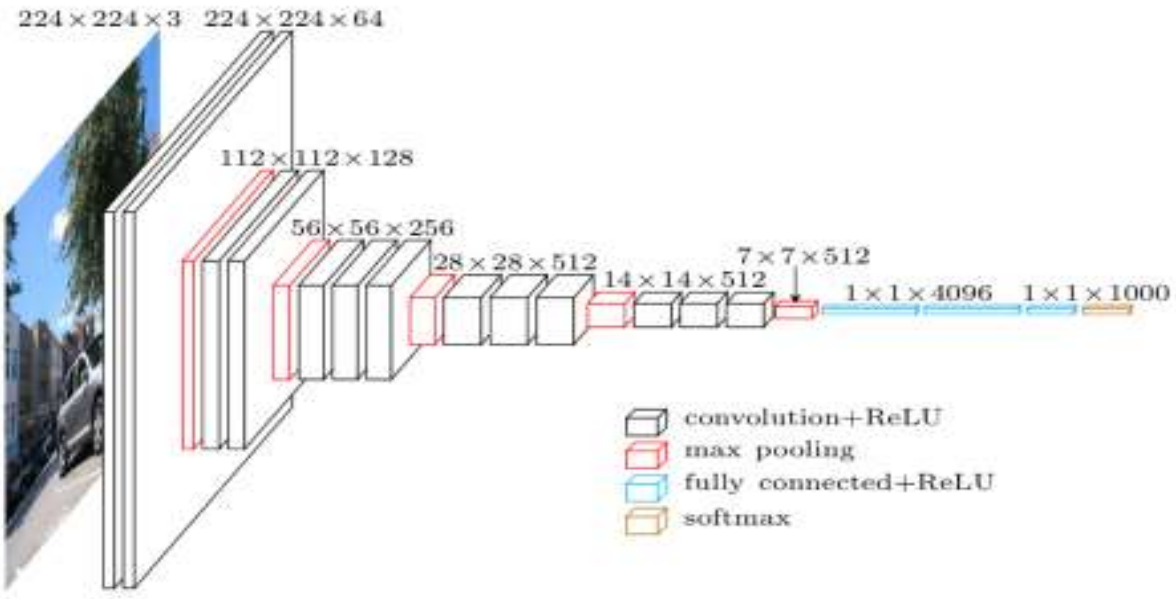
The required deep neural network API are Python, Keras and Tensorflow. Due to the fact that it is easier to setup with Linux rather than Windows, this project has been built using Linux Ubuntu 16. Additionally, the code borrowed is from Python 2.7 . Due to limitations in hardware, I used 4 CPU cores and no GPUs and, thus, the algorithms took one hour to complete. The following are the steps for getting Python, Keras, and Tensorflow to work in Ubuntu:

1. python –v (make sure it is 2.7)
2. sudo apt install python-pip
3. pip install numpy
4. pip install pillow
5. pip install scipy
6. pip install keras
7. pip install h5py
8. sudo apt-get install python-pip python-def
9. pip install tensorflow
10. suto apt-get install python-opencv

Algorithm

Programming Convolutional Neural Networks is a very difficult task. Not only to build one, but also to train one. Accordingly, a "hack" is used successfully. Annually, there is an ImageNet competition whereby participants create and train a convolutional neural network with millions of pre-set images of everyday objects such as chairs, cars, dogs, and cats. Yearly, the results of this competitions improve significantly. Many times, these algorithms are open source and freely available to the public.

A "hack" to easily create and train a convolutional neural network is to take a winning convolutional neural network and fine-tune it. This is accomplished by loading the ImageNet trained model, removing the final layers, and then creating and training with new final layers. The architecture of Convolutional Neural Networks is as follows:



The convolutional and pooling layers work to extract features of the images, whereas, the fully connected and softmax layers work to do classification. In this project, the VGG16 convolutional network was fine-tuned to classify Alzheimers. The algorithms are in https://github.com/marciahon29/Ryerson_MRP/tree/master/VGG16_Experiment and https://github.com/marciahon29/Ryerson_MRP/tree/master/VGG16_Experiment_Models . The difference is that one saves as weights and the other saves as models.

To understand how to work with these algorithms, https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html was consulted. The results of this preliminary exploration is in https://github.com/marciahon29/Ryerson_MRP/tree/master/Initial_Exploration . The goal here was to use VGG16 and InceptionV3 to classify correctly an African Elephant (JPEG). Inception was

correct in the second guess, whereas, VGG16 did not guess African Elephant within the first 10 guesses.

Results

The final script used is (https://github.com/marciahon29/Ryerson_MRP/tree/master/VGG16_Experiment_Models/Scripts) :

```
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dropout, Flatten, Dense
from keras import applications

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

top_model_weights_path = 'bottleneck_VGG16_00.h5'
train_data_dir = 'data_00/train'
validation_data_dir = 'data_00/validation'
nb_train_samples = 5120
nb_validation_samples = 1280
epochs = 100
batch_size = 40


def save_bottlebeck_features():
    datagen = ImageDataGenerator(rescale=1. / 255)

    # build the VGG16 network
    model = applications.VGG16(include_top=False, weights='imagenet')

    generator = datagen.flow_from_directory(
        train_data_dir,
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode=None,
        shuffle=False)
    bottleneck_features_train = model.predict_generator(
        generator, nb_train_samples // batch_size)
    np.save(open('bottleneck_features_train.npy', 'w'),
            bottleneck_features_train)

    generator = datagen.flow_from_directory(
        validation_data_dir,
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode=None,
        shuffle=False)
    bottleneck_features_validation = model.predict_generator(
        generator, nb_validation_samples // batch_size)
    np.save(open('bottleneck_features_validation.npy', 'w'),
            bottleneck_features_validation)


def train_top_model():
    train_data = np.load(open('bottleneck_features_train.npy'))
    train_labels = np.array(
```

```
        [0] * (nb_train_samples / 2) + [1] * (nb_train_samples / 2))

    validation_data = np.load(open('bottleneck_features_validation.npy'))
    validation_labels = np.array(
        [0] * (nb_validation_samples / 2) + [1] * (nb_validation_samples / 2))

    model = Sequential()
    model.add(Flatten(input_shape=train_data.shape[1:]))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))

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

    model.fit(train_data, train_labels,
              epochs=epochs,
              batch_size=batch_size,
              validation_data=(validation_data, validation_labels))
    model.save(top_model_weights_path)


save_bottlebeck_features()
train_top_model()
```

The train-test of this algorithm was difficult to achieve because of the limitations in hardware and, therefore, computational time. At first, these scripts were taking more than one hour. My final script takes almost an hour. Initially, I used a 90%-10% train-test with more than 100 epochs. The accuracy results were around 75%. After some tinkering and research, the data became 80%-20% train-test and using the 5-fold structure. Additionally, the computational time was decreased with 100 epochs. The results are in https://github.com/marciahon29/Ryerson_MRP/tree/master/VGG16_Experiment_Models/Results . The 5 accuracy values are: 0.9383, 0.9336, 0.9203, 0.8820, and 0.9406 with the average of 0.92296.

Here is the sample output:

```
Found 5120 images belonging to 2 classes.
Found 1280 images belonging to 2 classes.
Train on 5120 samples, validate on 1280 samples
…
...
…
4760/5120 [==========================>...] - ETA: 0s - loss: 0.1817 - acc: 0.9193
4840/5120 [==========================>..] - ETA: 0s - loss: 0.1813 - acc: 0.9196
4920/5120 [==========================>..] - ETA: 0s - loss: 0.1821 - acc: 0.9199
5000/5120 [==========================>.] - ETA: 0s - loss: 0.1813 - acc: 0.9204
5080/5120 [==========================>.] - ETA: 0s - loss: 0.1813 - acc: 0.9199
5120/5120 [==============================] - 5s - loss: 0.1808 - acc: 0.9201 - val_loss: 0.1911 - val_acc: 0.9406
```

Conclusions

In conclusions, the average accuracy achieved is 0.92296 using 6400 MRI images within a 5-fold structure of 80% to 20%. The VGG16 convolutional network was leveraged and fine-tuned to classify into two categories: Alzheimer's and non-Alzheimer's. The tools used were Python, Keras, and Tensorflow within the Linux Ubuntu environment. 92.296% is an excellent accuracy and much improved from the preliminary accuracy results of around 70%.

Suggestions for Further Exploration

A suggestion is to use a different convolutional neural network. VGG16 is excellent because it is well-established with plenty of information. There are newer convolutional neural networks that may produce better accuracies. In addition, ADNI (Alzheimer's Disease Neuroimaging Initiative), could also be leveraged as it contains more images and more data.

Overall, this project has been a success. It is hoped that Convolutional Neural Networks improve become accurate enough and safe enough to be successfully used within the Neuroimaging medical field.