Research Memo 2: Identification of Foraminifera via Convolutional Neural Networks (CNNs)

Antonio Leonti, 1/8/20

The purpose of this document is to disseminate the progress as well as roadblocks we (myself, Devin Cannistraro, Dylan DeFlorio, and Grayson Kelly) have had over the course of winter break. I start by restating the project goals, our current "big picture" approach, and finally the materials we have developed over the course of the last month.

Project Goals

- Using a CNN, automatically identify select species of benthic foraminifera (forams for short) from microCT scans of material from the seabed (which may consist of rocky or fine material)
    - We have been supplied messy, unlabeled microCT scans of "shelly sand" as test material ("batch scans") and of pre-identified individual samples for training material.
- Find the volume occupied in the batch scans by the target species' shells - a simple percentage of occupied volume may make the most sense.
- Make the entire process as foolproof as possible for the end user (for consistent results)

Current Approach

Our current approach can be summed up as the following:

1. Create training dataset:
    - First, threshold out irrelevant data (background, other objects) using Drishti (done by hand, but only once). Then perform patch wise cropping (using size threshold to ensure "garbage" doesn't affect cropping bounds), and finally resize. Before training, the entire dataset is converted to a 4D tensor (each image is represented by 3 color channels) and normalized.
2. Train CNN:
    - Iteratively cast PyTorch training spells until satisfied with loss values.
3. Create test dataset / *the hard part*:
    - First, as with the training images, threshold out irrelevant data (backgrounds) using Drishti. Then, *intelligently* segment batch scans (ignoring things like single-pixel bridges). Throw away irrelevant data via a size threshold. Once properly segmented, group result-

ing images vertically by overlaps (overlaps in consecutive images indicate representation of the same object), then merge images from the same source image. Finally, perform patch wise cropping (no threshold needed this time) and resize like training dataset. Also like the training set, the dataset will be converted to a 4D tensor and normalized before being handed over to the CNN.

4. Present test dataset to CNN:

    - The exact process is TBD, but it will involve inferencing the class of whole objects (which is why we vertically group them in the previous step) as opposed to single images. Once we know the identity of all objects, we can calculate the volume taken up by the target species by summing the area of each individual constituent image.

Materials

These are the materials we have developed to help us solve this problem. In this section, I describe the purpose, tools used, and implementation for each.

Convolutional Neural Network

Our CNN is written in Python using the deep learning library PyTorch. The CNN's current design is very simple, because we lack have experimental data to gauge its baseline performance (why fix what isn't *verifiably* broken?). I have simply tried to make a "basic CNN" modeled after things I have seen online. Once we have experimental data, it will be fast and easy to change any aspect of the network to our liking as we are not writing anything from scratch.

This is the current architecture of the CNN:

1. Convolution: in channels: 3, out channels: 32, kernel size: 5, stride: 1, padding: 1
2. ReLU
3. Max Pool: kernel size: 2, stride: 2, padding: 0
4. Convolution: in channels: 32, out channels: 16, kernel size: 5, stride: 1, padding: 1
5. ReLU
6. Max Pool: kernel size: 2, stride: 2, padding: 0
7. Fully Connected Linear Layer (FC): in features: 3136, out features: 64 (tensor is appropriately reshaped first)
8. FC (Output layer): in features: 64, out features: 6 (5 target species + noise; explained more below)

The CNN defaults to using the Adam optimizer and a cross entropy loss function, but I have made it so these can be defined as anything else at the time of training if we choose.

Patch-wise Cropping Algorithm

Our training images are not square (to my surprise), which is required by the CNN. They also vary in resolution, while our CNN requires a set resolution. Finally, the content does not take up the entire image, wasting real estate which should be occupied by useful data (even more important when downsampling). All of these problems will apply equally to the test data, as well. My patch wise cropping algorithm (followed by resizing) presents an easy fix for all these issues.

To implement this, I used NumPy (large-matrix Python library) and OpenCV (image manipulation library). First, the images are thresholded, and the resulting contours are enumerated. The contours are then iterated through, with ones that meet a provided size threshold increasing the size of the cropped region. The cropping region is then expanded to be a square and the image is resized.

Noise Generation & Noise Class

Consider Plato's allegory of the cave. It describes prisoners trapped in a cave facing a blank wall. From behind, a fire projects shadows onto the wall for the prisoners to watch. If the prisoners never leave the cave, they wouldn't know anything else exists other than the shadows.

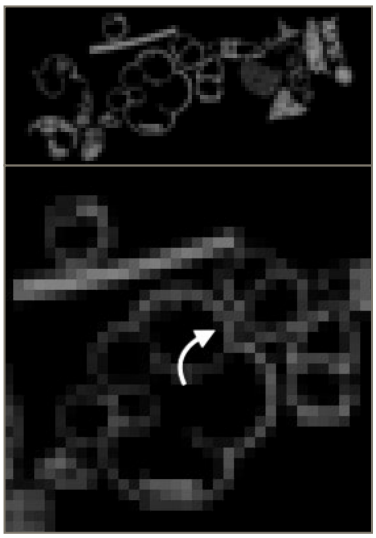
Similarly, our CNN only sees whatever images we show it. The only classes of object which exist to the CNN are the ones hard coded in its output layer. The CNN also has been optimized to always give confident answers. Classification is like a multiple choice test lacking a "none of the above" option. We show it an image, and it must give an answer within the bounds of a predefined output space. Classifying everything as one of the target species is just as useless as classifying none.

This problem has arisen ultimately because we are applying a tool designed for classification to an identification problem. Ideally, every object in the batch scan would have an applicable class. This is impractical, so I propose we implement a "none of the above" option for the CNN using a noise classifier.

The noise is synthesized by simply creating a one dimensional matrix, populating it with random values between 0 and 255, then saving it as an image file. This is done many times to create a dataset whose size is the average of that of every other class. Before training, the dataset is normalized (making it Gaussian noise). When training, this is just another class, but the hope is that after training, the other classes will be less "flat," ie more selective, as a result. We will not be able to test the efficacy of this, however, until we have a reliable segmentation algorithm for the batch scans.

Batch Scan Segmentation

Our training images are all images of isolated samples. For the best results, our test data should follow suite, else the data will not be similar and our CNN will have no way of classifying it. YOLO should have been a wonderful luxury in this aspect; YOLO approaches identification (defining bounds of objects of interest) as a regression problem, eliminating the need for segmentation of images before classification. For whatever reason, our data did not play nicely with YOLO, so we must segment the images ourself. This is *much* easier said than done!



Top to bottom: example of segmentation gone wrong; detail of top image (arrow points to a particularly bad connection)

Devin's current algorithm for this involves ray casting along each row of the batch scans and flood-filling any connected non-black pixels he finds. The resulting disjoint region is then cut (copied & erased from original image) and pasted to a separate image.

The problem is that "disjoint region" often does not mean "single object" in the way we would like. Many objects are connected by a pixel or two (in many cases a lot more), against which our algorithm simply has no defense. This results in many of the segmented images containing dozens of objects. While the connected objects are mostly distinguishable by eye, they are proving to be extremely difficult to separate algorithmically. The images on the right should shed some light on the kinds of problems we are having with the batch scans.

Devin is in the process of implementing an additional feature which looks for rapid changes in the distance from one side to the other. The idea behind this is that if a shape gets really skinny in the middle, it may actually be two shapes.

I have approached the problem with my own ideas of how to solve it. The short of it is that none of my approaches has been fruitful. The closest I got was by pruning the edges of each object (in an attempt to get some separation between objects), resulting in a dilation of the space between objects. This worked sometimes, but many objects (including those belonging to our target species) are too small to survive this process.

Getting something useful out of the batch scans is imperative to the success of this project. The usefulness of all the work we have done up to this point is determined by our ability to do so.

GitHub

A GitHub repository has been set up and contains most of the code for this project. It can be viewed at: https://github.com/antoniovleonti/foram-cnn