Capstone 2 Project Report

Objective

The initial objective of this project was to implement a use case model to do diagnostics with biomedical images. Biomedical applications in deep learning are inspiring with medical image data said to be growing at a fast rate along with an increasing demand to scale diagnostics. The major task in this deep learning project is to classify brain tumors of differing severity across 4 mri modalities. Techniques such as convolutional neural networks are used to classify tumors by segmenting subsets of the image.

Impetus for this project originates from deep learning approaches to analyze images and a personal interest in the biotech industry. As it happens, the most thought provoking and reasonable application of image analysis is medical, geospatial, and cosmological. And though the particular application to biomedicine currently lies outside of my domain knowledge, I believe it would be in my best own interest to have such biological knowledge when analysts like myself receive ground truth labels, especially while many research fields have not yet been integrated with computer science methods across all domains.

Just as well, clinicians who provide ground truth labels may not in most cases have the technical knowledge to scale diagnostics. It would be unnecessary to restrict the possibility to scale when many patients can benefit from the attempt to do so. Nevertheless, I am currently taking steps to fill the gap in my domain knowledge before I make a whole hearted attempt to apply for positions in the healthcare industry.

The skills acquired in this deep learning project can be applied to several other type projects in deep learning. The objective once again is to understand how to do a variety of deep learning tasks. The first is to understand how to preprocess image data. Second is how to deal with large data sets that have been generated with high technology such as mri. Third is how to implement a use case model on a biomedical dataset and to ultimately understand the model architecture.

Approach

The approach to this deep learning project was intentionally made as easy as possible. The decision to implement this specific use case model among several others had to do more with the clarity of the methods and classes than anything. The choice even had to do more with clarity than model accuracy. The choice to use Sachin Mehta's model was a fortunate occasion since he and my mentor Ajith Apatnaik are colleagues. Mehta's model is easy to interpret. He begins with the removal of the black area in the image. The method used to remove black area is a cropping transformation. Since the image contains slices(155), imshow() will display the masks in the segmentation files and Mehta uses the geometric boundaries of the brain mask to find the maximum coordinates of the image for cropping purposes. He does this for all three dimensions in the image, including the height, width and channels. This allows us to do operations on the pixel intensities of interest, pixels focused on brain area. Similarly, indices of non zero image slices are being saved and it is a happy coincidence that null images tend to appear at the beginning and end of the MRI scans. A continuity of brain slices in an

image file is generally ensured for each patient scan by the end of the cropping transformation.

The preprocessing side of the project can be reduced to just a few transformations, including the cropping transformation previously mentioned. The images need to be cropped to remove unnecessary black area. Images without brain need to be removed as they can offer no information. The image slices need to be reindexed and saved to perform modeling in the final phase of analysis.

A method cropVolumes() in the file preprocess.py standardizes the indexing of dimensions for all image objects within each patient folder for a total of four reindexed image objects per patient including flair, t1, t1ce, t2. The standard only applies to one patient folder and needs to be extended to all images and image slices.

The methods in transforms.py provide final solutions for resolving other preprocessing issues. Since the images all need to be standardized in size, class ScaleToFixed() provides an answer to that problem. Class RandomFlip() is useful for randomizing the orientation of the image to provide some level of fairness to the classification process. The values of the images are then passed to MinMaxNormalize() for normalization. This generates a higher resolution image and finally, ToTensor() converts all images to tensors for the modeling phase.

Even though this project is simply a use case model, I can see a clear distinction in the preprocessing phase between manipulation of pandas dataframe with standard numeric data, column names and manipulation of images with pytorch and numpy arrays. Image data comes in much larger samples and the dimensionality is different

and in most cases higher. Preprocessing images is far more laborious. For example, Mehta's code provides a geometric solution to the issue of black area with cropping transformation. The solution method does not just return values of interest, it includes the creation of directories and file paths where all return values are then saved for later use.

File path manipulation is just as essential to preprocessing as it is to the entire process. Numerous files are used, reused, modified and created in the processing phase that it immediately distinguishes deep learning tasks from ordinary machine learning tasks dealing with csv files, which are cleaned and merged in smaller quantities prior to the modeling phase.

Several aspects of this use case appear to be unique in design such as the use of the ArgumentParser(). This method creates an object with a list of methods and allows you to access them as if they were columns in a dataframe.

Data

To obtain the data you must first go to the UPenn BraTS Challenge website and follow the instructions for requesting data. The instructions guide you to create a CBICA account and submit a request for the data. You are not required to participate in the challenge but the submission info may lead you to believe that you are. The submission may take some time to process and you are allowed to check the status of your submission as you wish. Once you are approved, you download the request document. The document contains a link that will begin downloading the data for you. If you find

that your submission is taking too long for approval, you are recommended to email a faculty member named Spyridon Bakas. Be aware that if you do not register for the challenge your request will be deprioritized and may take longer than anticipated. Your interactions with Spyridon and other affiliates will be misleading and pointless. Nevertheless, it is important to be persistent as in my case. I was able to obtain 3 years of brain mri data including csv survival data. The directory structure is more or less the same for each brats dataset. The contents of the image files of 2018 and 2019 data sets are said to be different from the older ones. The dataset has two folders. The LGG and HGG folders each containing patient image files of different modalities. Each modality provides a different resolution due to a change in imaging techniques. The patient folder will contain 4 different modalities such as the flair, t1, t1ce, t2 and ground truth labels(segment.file). The old datasets use .mha files and the new datasets use nii.gz files. Python packages can conveniently handle them just the same.

The class in the file loadData.py contains 3 methods including compute_class_weights(), readFile(), and processData(). The method compute_class_weights() normalizes the class weights, as in weights attributed to each class of this classification problem. The method readFile() does the necessary work to read image file paths written into the generated text file for more preprocessing modifications and then data loading. The method processData() is responsible for reading the text files and returning a data dictionary with class weights, and image file paths.

An important note regarding the data, the task of training diagnostic data has only been a small portion of the ultimate task here. The ultimate task has been learning the entire process of implementing big data and not just that but following another author's lead. To use another author's work as a guide has been quite a laborious task in and of itself. Though, the work is much less than having to write the code from scratch. Without a doubt, the more unique aspect in the implementation of this big data implementation has been the amount of double checking of file path modifications, making sure all class files have been imported before running main and 'terminal' methods. Lastly, it has been crucial to debug by printing outputs and objects wherever suspicion arises, especially when model metrics are not churning out correct values. Mehta's code, with the help of my mentor, has been incredibly insightful and useful in breaking open the task of deep learning. Modeling biomedical data has inspired me to pursue the area of deep learning and bioinformatics. There is much more to be done with this type of data and still other biological data, especially genetic.

MRI

The technique used to produce the images being models is Nuclear Magnetic Resonance. The technique is quite advanced in how it is able to capture pixelated imagery hidden inside the body. Without delving too much into the theory behind it, NMR is able to manipulate the direction of the water molecules via nuclei with vectorization through the application of magnetic orthogonalization and radio frequency that is in tune with the precession of the water vectors. Pixels are described and illuminated

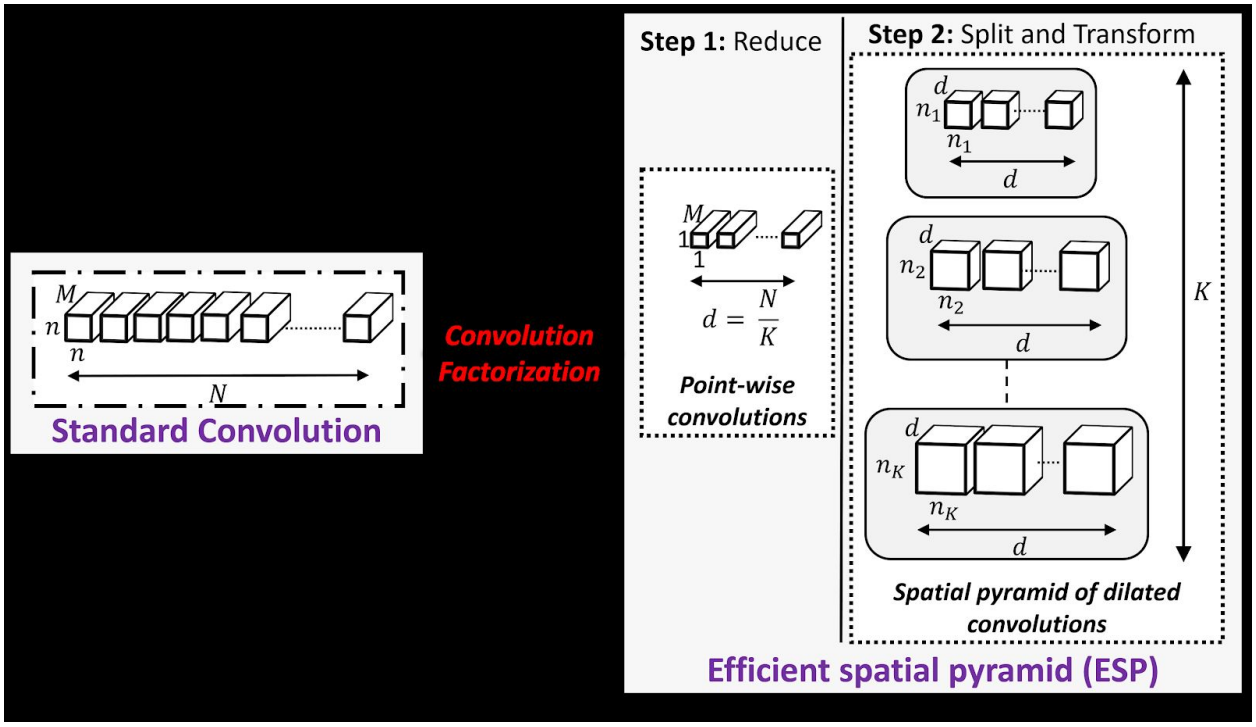
through the phenomenon of precession at the lamar frequency. This phenomenon is also analogous to the precession of the celestial bodies as a consequence of angular momentum. When image files are finally downloaded, they are originally provided on the grey scale but can be transformed into colorized images. The reason why MRI works so well and specifically works well in brain mri is that for brain tumor targeting, tumor tend to be identified as white pixels, which means they tend to contain an excess of water. So we are able to capture the deviations in the body and especially the brain, if we are looking for tumors. Outside of the theory, there are numerous packages that deal with MRI file types and so would simple to build on and improve such libraries for bio diagnostics.

Mehta's Paper

The ESPNet model is based on a convolutional factorization principle. In other words, the standard convolution layer is factorized or decomposed into point wise convolutions. Note that factorization is a matrix method used to get rid of linear dependence in the column space.

For a set of N filters with M slices, each standard convolution kernel is size nxn. The diagram indicates that with fixed depth and length d as the shape of the nxn filter grows stacking k layers of length d in parallel will give a pyramid shaped architecture called the spatial pyramid of dilated convolutions. This ESP module architecture resamples feature maps to learn representations from a large effective receptive field. The effective receptive field is a region of input values the units of a cnn depend on. The

strategy for increasing the effective receptive field is by dilating kernels with zero padding. The dilated kernels learn weights with different receptive fields.



Additionally, ESPNet has an encoder to learn representations with convolution and downsampling. Following encoder operations, a light-weight decoder produces the segmentation mask. The entire encode-decode process is guided by downsampling and upsampling of spatial resolutions.

According to the paper, matrix factorization as it relates to convolution factorization is said to reduce computational complexity. Along with factorization, sparse decomposition is used to improve computational complexity by removing redundancy in the network.

PReLU is the activation function used, which is a variant of the leaky ReLU with an adjustable slope for negative weights. The input and output feature maps are combined with an element wise sum to improve gradient flow within the network.

ESPNet Model

The ESPNet stands for efficient spatial pyramid. It is a variant of the former ESPNet neural networks with a proven record of improvement on computational complexity, power and other areas. The model is based on a convolutional neural network and has a basic structure that follows the shape of a pyramid. The initial number of filters is divided by some positive value k to determine the number of filter layers where each filter is of size nxn and grows in each layer. The resizing of filters at each level explains the shape of the architecture.

Getting the model to work was quite an undertaking. Succeeding in implementing the model by no means implies a complete understanding of the model itself. My current understanding of the model lies within aspects of the model that I am already familiar with through my various other implementations in machine learning. The approach to this model is unique as previously stated. The use of images infers big data. As a result, the use of GPU and cloud based modeling may be necessary for faster implementations. This particular project requires a lot of file manipulation. And there are already numerous tools in python waiting to be utilized for this process.

Metrics and Results

The model is trained on only one class but whenever that class is identified, it is identified with an accuracy above 80%. This is why the mIOU(intersection over union) scores are so low. It is correctly guessing one class with an accuracy close to 1 and all others 0 because training is only done on one class. The results were as follows:

Epoch: 0
Learning rate: 0.0005
Train Loss = 0.9929
Val Loss = 0.9285
mIOU(tr) = 0.2489
mIOU(val) = 0.2788

UNet Model

This implementation of the UNet is far easier. The fastai library(https://docs.fast.ai/vision.learner.html) contains all the tools for its implementation and is essentially no more difficult to implement than a random forest classifier with an ordinary customer transaction csv file. Brats data is from 2015. These are mha files. The processing of the data is as straightforward as extracting the file paths and normalizing images slice by slice. This is as far as the hands on transformations go before handing off the data to the fastai library. Up to that point, the data gets converted into numpy arrays and then pngs before finally being fed into the fastai methods for one last transformation and learning. It's important to note that data has been stored in google drive to take advantage of free gpu you usage in google colab. In addition, it is important to note that complications regarding the size of the data have been encountered various times. GPU and TPU availability on colab have been recurring issues as well as the performance of the GPU. Implementation can be

performed on AWS, if you're willing to incur the cost of a notebook instance with several GPUs and however much storage space you wish. Having said that Colab GPUs have often failed during processing and modeling, which I assume may have been caused by the large amount of data being dealt with.

Because the source is shorter, only two notebooks are used and can be found in the GitHub repository.

From processing to modeling, UNet implementation contains only two notebooks. In brats_processing.ipynb, there only a just a few methods including grabmhas() responsible for parsing training/ file paths from label/mask file paths, loadData(), basic_norm() and slice_norm(), which normalize the image slices. Finally, there is a main function which implements the processing of data and conversion to numpy arrays before writing the processed file paths to new directories as png files, an appropriate file type before modeling.

The brats_fastAI.ipynb is even more scarce in its treatment of the processed data. There are two methods including get_y_fn() responsible for collecting all the mask labels and acc_classwise() responsible for providing us with the mean of our class accuracy.

Conclusion

The UNet is a viable solution, especially when you need to achieve results in a short period of time. Accuracy scores on the 2015 BraTS data set in a matter of 10 epochs can reach 60%. With more time, epochs and GPU power one can achieve an 80% accuracy without tuning hyperparameters. ****