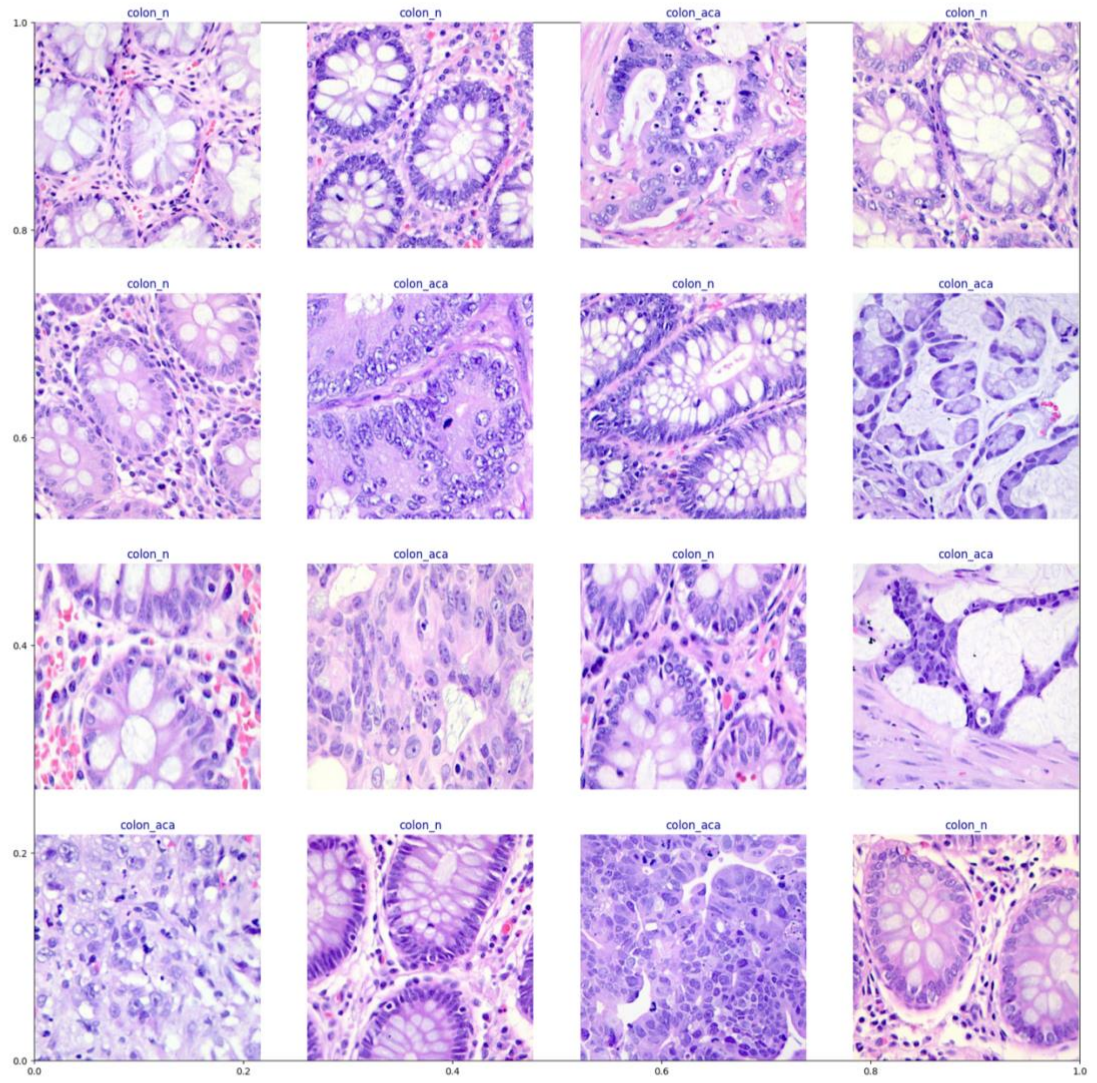
Colon Cancer Classification Using Neural Networks

Brandon Ling

December 23, 2024

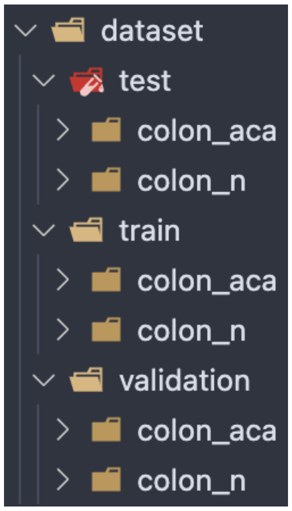
Overview

Currently, diagnostic measures diagnosing colon cancer involves a relatively simple procedure called a colonoscopy. While physicians are generally able to extract potentially cancerous polyps, they cannot tell simply by looking at them whether the polyps are cancerous without sending them to the lab for further analysis. It is in the lab where errors can be made when determining the cancerous nature of polyps, since certain polyps may be mistakenly overlooked. In this regard, machine learning holds immense promise for improving the screening efforts of physicians to prevent colorectal cancer cases and improve early-detection efforts. By training a neural network on existing lab samples of colon tissue, we can hope to accurately identify polyp samples (which are simply extensions of tissue) when they reach the lab. The below graphic contains 16 labelled image samples that were used in the training of the model. "colon-aca" and "colon-n" represent colon adenocarcinoma and benign colon tissue, respectively.



Data

When I began working on this project in September 2024, I was just beginning to deeply understand how neural networks worked. I spent my time learning about everything from feed forward and convolutional to long short-term memory neural networks to the modern transformer architectures that enable large language models to function, but I wanted to put that into action. To do that, I needed to find myself a dataset.



```
dataset
  test
    colon_aca
    colon_n
  train
    colon_aca
    colon_n
  validation
    colon_aca
    colon_n
```

Of course, the first place I looked in search of a meaningful dataset was Kaggle. Eventually, I stumbled upon the one used in this project, a collection of high-quality colon and lung samples. This dataset stuck out to me for a few main reasons. It was relatively large with 10 000 images of colon samples alone (example shown above), it represented a potential solution of using AI to enable quicker diagnosis, and it was processed in a consistent manner. This enabled me to have to do very little data processing myself, aside from organizing my broader data directory into test, train, and validation sub-directories, as shown to the left. The approximate split that I used to separate them into these distinct categories was 80/10/10 for training data, test data, and validation data, respectively.

Design

Choosing model architectures is often self-revealing, in that the optimal neural network to use is usually apparent before even testing them. In this case, it was very clear to me that I should train a convolutional neural network (CNN), given that this type of architecture is centered around image classification and computer vision applications. As such, that is where I began. I started with a simple design involving four CNN layers with ReLU activation layers, as well as 2D max pooling layers of kernel size two to down-sample the neuron outputs into increasingly smaller vector spaces. Essentially, this was the basis of the two main blocks of the CNN model. Finally, the model utilized a classification layer that first flattened the input into a one-dimensional tensor, then applied a fully connected layer followed by a ReLU activation, and

I also needed to be able to feed the image data into any sort of neural network. As such, it was necessary to apply transformations to it. When looking into what transformations I should apply, it was clear that I should convert the image data to tensors, but usi

Results

As previously mentioned, the convolutional neural network (CNN) based model that I aimed to train from scratch did not perform up to expectations, both in terms of training speed and model accuracy. While I was able to once achieve 95.7% test accuracy, it was impossible for me to recreate this in future attempts using any hyperparameters. Consistently, using the SGD optimizer and by varying batch size, hidden layers, pooling kernel size, and more, I ended up getting around 55.6% accuracy. This abysmal accuracy can likely be attributed to imbalances in the data, or perhaps it was an issue with the GPU cluster I was using to train these models, but this was an extremely confusing phenomenon for me.

Deployment

Once I completed my analysis of the various model architectures, I wanted to enable others to take advantage of the capabilities of my model. Up until this point, I had simply been training the models using a loop that I had created, printing the accuracy of the model on test and validation sets, and letting the weights disappear after the training script finished running. However, to

Conclusion

This has been a rewarding project that has allowed me to explore the fundamentals of finding a viable dataset, designing and training an AI model, and deploying said model for practical usage.