1. Hi I’m Ian and I did the default medical imaging project. Here I list my uni as well as the link to my git hub repo for the project in addition to my full submission on course works.
2. In this presentation I’ll discuss how I dealt with the data, my attempt at semantic segmentation, shifting to classification, my results, some experiments, and lastly a code walkthrough
3. The data we were given to work with for this project was a sub set of 20 slides from Camelyon16 that each contain the same image at several different magnification levels shown below. As you can see, some of these files can be very large, and can quickly fill the RAM and disk of regular google colab environments. Another thing to consider is that there is inherently less cancerous pixels in each image than non-cancerous pixels and different amounts of cancer in each slide, so I had to be careful to work around some of these issues when handling the data for training my models
4. Of the 20 slides, I chose to use 17 of them to make training and validation sets and held out 3 for testing. When testing, I used the entirety of the slide images, but for training and validation I saved unique 299x299 sub images into folders for each magnification level. This made it easy to train each model on only one level and keep balance the cancer positive and negative examples. But I’ll get more into this later.
5. In order to deal with the limited amount of data from the class imbalance and the 20 slide subset, I implemented the data augmentations used in the paper discussed in class (Detecting Cancer Metastates on Gigapixel pathology images). From the excerpt I show below you can see that they do some 90 deg rotation, flip, another rotation, and several tweaks to the color appearance. On the right you can see some of the outcomes of my version of these augmentations
6. Lastly I wanted to discuss the accuracy metrics I’ll be using to benchmark my results for this data. Since the labels come packaged as a binary mask for each slide, I decided to use the following:
   1. Intersection over union
   2. Precision to highlight the number of false positives
   3. Recall to highlight the number of false negatives
   4. and F1 score; which is just an overall metric that’s a function of the precision and recall
7. I won’t spend too much time on this because this didn’t end up being the route I chose, but I did spend a decent amount of time at first trying to implement a true semantic segmentation model in which I would output a tensor with similar dimensions to the input, but each channel would represent the confidence that that pixel belongs to either the positive or negative class. To do this I tried implementing a Unet to incorporate spatial information into the pixel classifications because I had seen that used for similar medical imaging segmentation problems in the past
8. In principle this could work, but in practice I had a lot of trouble with overfitting and never did much better than ~0.6 F1 while still overfitting
9. Here you can see some arguably decent predictions in green and some very bad predictions in red
10. I decided to move away from this approach due to the low convergence speed, high spatial cost of implementing a lossless multi-model architecture, and the easy overfitting I was experiencing
11. It turns out that the solution to a lot of the issues that I was struggling with were fixed by moving to classifying each of these relatively small sub images which at high magnification levels, end up being small enough to produce a good segmentation of the image. Now that I only really care about what information is in these sub images rather than where things are, I could use a much smaller and simpler CNN. Fewer parameters means faster convergence and less overfitting. And when combining predictions from different zoom levels, I could now downsample high magnification predictions without loss into a much lower resolution base image which I’ll explain more about in a second
12. The model I implemented to accomplish this sub image classification was a simple CNN with a few 2D convolutional layers plus pooling, and to make predictions I eventually flatten the output and pass it through a few dense layers with drop out, narrowing down to a final 1 unit fully connected output layer with sigmoid activation to predict a confidence that the input sub image is cancer positive
13. Then for my multi model architecture, I took three CNNs, each trained at their own magnification level (1,2,3 in this case) and averaged their predictions for a given area together. At inference time, a sliding window is moved across a slide image at each of these magnification levels, and the scalar confidence predicted by the CNN for that level is broadcast into a tensor whose size is proportionally set by the lowest magnification level. This is equivalent to downsampling the constant higher magnification prediction from its original image size, and this allows me to only store the full image of the lowest magnification at once. I should also mention that this is slightly different than the method Liu et al used, in that they used a fully connected layer that took in the logits of each network to make predictions, but I found my method to be much easier and nearly the same other than that a final dense layer would effectively be a learned weighted average of each models output. So maybe it could learn to trust one level prediction more than others, but for my purposes of rapid prototyping I found this to work fairly well.
14. I also wanted to define some new things for the classification model. I’ll be referring to binary mask that’s given as a label for each slide as the “high res ground truth mask” and then the perfect possible prediction at the highest mag level inferred from that label I call the “low res ground truth mask”. Similarly the ground truth heat map will be the best possible soft prediction for the mag levels we are using.
15. First lets qualitatively look at the predicted heat maps and binary masks thresholded at 0.8 confidence for my 3 slide test set. You can see that in cases where there are many or an intermediate amount of positive regions in the slide, the model does pretty well, but still has some false positive and negatives. However when most of the image is largely negative the model predicts that almost nothing is cancer in the heat map and nothing makes it through the threshold. Possibly this could be made better by using even higher mag levels or more data.
16. Here’s the quantitative metrics averaged for each of those test slides, also shown excluding that third example that had very little positives. I think generally an IoU greater than 0.5 is good and an F1 score ranges from 0 to 1, so I concluded that my multi-model classification method does a reasonable job at localizing cancerous regions on full unseen slide data, especially when the data contains detectable positives.
17. For reference here’s the training curves for each zoom level’s CNN. I’ve also marked with vertical green lines the epochs that I loaded weights from to do inference on the test data.
18. I don’t have too many visuals for my experiments because a lot of them didn’t really have much of an impact, but the main thing I tried was varying my learning rates for each zoom level as well as other things listed here. I had some things I wanted to try but kind of ran out of time, mainly I wanted to try to have a quantitative metric for splitting my slide data and try to use a segmentation model to search for flesh rather than just use the threshold from the starter code which does still let through some non-flesh images
19. Here’s some data from varying the learning rate at level 2. You can see that the I didn’t really get anything out of trying to stretch out the training process and the higher learning rate actually does better
20. But here you can see that the low magnification level behaves slightly differently so I ended up using higher learning rates for lower the magnifications as that you can see in my code walkthrough
21. As I’ve said before we had some great starter code that I use throughout to deal with opening the slides at specific zoom levels, picking out regions of them, and overlaying them with their masks
22. The first thing that I did was try to split a slide image into unique equally sized regions. There’s a lot of code to do this to the left, but I wanted to single out this section of code outlined in blue that I also use throughout this project. It basically calculates the dimensions of a grid of NxN sub images and reads each of them at a given level from a slide. This is very useful but in this case I’m just append them to a list
23. Here’s what that looks like when visualized. Note that the slide images do not always fit perfectly into the grid of NxN images, so some get padded with zeros, and I eventually cut these regions out during inference. Another simple test that I don’t show here is having a CNN memorize the labels from this grid of sub images.
24. Once I was sure that my model could learn to classify sub images I moved to getting the needed data into colab. I hosted the tif files that I used for training and testing in a google cloud bucket and I have some utils for downloading them, but I eventually moved to hosting zip files for my level datasets once I was happy with them because splitting level 1 and 2 can take a very long time
25. Here’s some utils for splitting the positives and negatives into level folders
26. And here’s the main routine for saving my datasets. Forgive me that its kind of ugly, but it basically opens each slide at a given level, goes through each 299x299 region of them and first saves anything that’s mask isn’t empty
27. Then knowing how many positive sub images there are, this function again goes through each slide, but now randomly samples sub images from the grid and saves them to the negative directory only if there is nothing in the mask and > 25% flesh in the image. It also attempts to sample an equal number of negative examples from each slide
28. With that data saved I then set up the augmentations I discussed earlier being sure to normalize both the training and validation data
29. Here’s I setup models for each level. Again please forgive how ugly the copy and paste is.
30. Now we come to the main training loop. Here I just set up the folder paths for the data sets as well as checkpoints for each level model
31. For each of the level datasets, I setup train and val sets from the positive and negative directories. Then I map the augmentation to the train set and a rescale to the val set. I then compile each model with binary cross entropy loss since Im just predicting one value, and I use a predetermined learning rate to fit the model to its respective level training set
32. Lastly I wanted to show some of my code for visualizing the results on the test set. You can see that the first thing I do is make an ndarray of zeros that’s the size of the lowest resolution image. This is what I will fill with my predictions to make a heat map. Then this function slides through each image at a given zoom level, pulling out a small sub image to predict on and a mask region for comparison. It also calculates this downsample factor to later find the correct size in this full\_output ndarray to add the predicted confidence to.
33. Here’s where that numpy magic happens. Sorry this is a little ugly and hard to read but I circled the most important part. It reshapes and normalizes the sub image for inference making sure to set training to false in the model so it doesn’t use the dropout layers. Then it does some fancy indexing to map the predicted confidence for this level to the correct area in the output heat map with that downsample factor.
34. Here’s a little more numpy magic to throw away any of those padded black regions we saw earlier as well as threshold the heat map to get a binary mask
35. Lastly here’s how I calculate the accuracy metrics for high res with the true mask and low res with the inferred mask. Nothing too special here just using the tf metrics
36. And here’s what my predictions look like
37. Here’s my references
38. Finally just wanted to say thank you for the semester. I definitely learned a lot. Thanks and have a good break!