**Outline**

1. Introduction
   1. Pathology workflow
   2. Advantages of digital pathology
   3. Current public pathology corpus
2. Database
   1. Image Digitization
      1. Slides scanned via Aperio AT2 scanner that scans 400 slides at an approx of 8 hours
      2. Slide scanning is not completely automated as a snapshot is needed before scan which has to adjusted by manipulating the focus points.
      3. Conversion from analog to .svs images
      4. .svs images is a layered .tiff
      5. Use of open source software to view and manipulate .svs files.
   2. Data Anonymization
      1. Utilized python script to remove slide labels.
      2. Used macros to remove patient information from patient files.
      3. All files were assigned a randomized MRN and specimen ID
   3. Data Organization
      1. Slides are placed in broad categories according to their specimen type
      2. Two directories were created; one for the original slides and one for the anonymized slides.
      3. The naming convention of slides according to the information extracted from the slide label.
   4. Computing Infrastructure (brief.... this isn’t terribly novel)
      1. Sizes of the slides range from 100 of megabytes to 5 gigabytes
      2. Cloud computing was enticing but was not used due to higher costs.
      3. A ZFS filesystem was used at the filesystem level and a Gluster FS was used at the highest level.
      4. SLURM was used as the workload manager
3. Annotation
   1. Integration of the corpus into the TUH workflow relied on the eSM.
   2. Slides are assigned to individual pathologists for annotation
   3. Annotation was done with Imagescope or Leica Image viewer.
4. Deep Learning Approaches
   1. Preliminary dataset (preprocessing)
      1. Marked / no mark
         1. Images averaged around 5K by 2K in size
         2. 1000 total images
      2. Annotation of SVS slides (Imagescope -> Noah’s tool)
         1. Polygon to frame based for more efficient annotation / sacrifice accuracy for efficiency
         2. Use of thresholding to determine a frame’s label
         3. Annotators labeled 50x50 frames of windows if a frame had >2% marker
            1. Lead to human error (agreements at 91%)
   2. Training
      1. CNN, LSTM, Batch Normalization, Relu (Dense), Max-Pooling, Dropout (ported over Meysam’s)
         1. Initially used the best EEG system and adapted it for images
            1. Talk about success of the EEG system
         2. Why CNNs are used for image classification
         3. Limiting the amount of Max Pooling Layers
         4. Normalizing the batch as it propagates through the network
      2. Model (as of writing this 03/06/19)
         1. Five convolutional layers
         2. Five Max-Pooling layers
         3. Relu Activation
         4. Adam Optimizer
         5. Softmax Function
         6. Categorical Cross-entropy
         7. CNN -> Fully Connected Layer -> Output
         8. Model outputs weights after each epoch (checkpoints)
            1. Can load model at a certain checkpoint in case of unexpected job cancellation
      3. Keras with Tensorflow backend (sequential model)
         1. Simplicity of Keras compared to Tensorflow (high level API providing layers of abstraction)
            1. Can focus less on the background, and can fiddle around with the architecture more
         2. Theano outdated on 2017, discontinued
      4. Generator Function
         1. SVS vs. JPG
            1. Model can accept any file format offered by cv2 and openslide

Any valid file format besides TIFF/SVS are read in all at once

* + - * 1. Aperio’s SVS file format can be read in section by section (or random access)

Use openslide to handle SVS images

SVS images have different slide levels (0 being the lowest). Dataset was run on slide level 2, which is 16x downsample from original resolution

Take advantage of random access by reading in a row of windows at a time to avoid reading the entire image at once

* + - 1. Balance sample weights
         1. To balance the under represented classes, a function balance\_weights was created
         2. Returns class weights along with balanced weights

Ex: labels = [0,0,1,0,0] weights = [1,1,4,1,1]

* + - * 1. Prevents the model from classifying everything as null, and forces to pay attention to underrepresented classes by punishing the loss function
      1. Out of Memory
         1. Running out of memory was common in the initial LSTM model
         2. Removing the LSTM and 1D convolutions reduced memory usage
         3. Training is run on GTX 1070s and Tesla P40 GPUs for ample memory availability

Talk about tech specs of each GPU node

* + - 1. Runtime
         1. In the process of running experiment with different frame/window size

Report on the effects of runtime and memory

* + - 1. Getting frame class
         1. Annotations can be free hand and not always rectangle
         2. Need method to determine the label of the frame
         3. Get\_class\_id originally was run on every frame of each image

Creates a grid for the current frame

Loops through all the annotations

Use Matplotlib to see if two points overlap in the annotation

Annotations are stored as Matplot Paths

If the overlap was greater than or equal to a given threshold, return the class id and overlap percent

Percentage represents the amount of the annotation that lies in the frame

* + - * 1. Frame\_contains\_anno function created

Checks to see if any of the points of the annotations lie in the frame

If not, the points on the perimeter of the frame are checked (annotation might be larger than the frame)

* + - * 1. Updated method to get frame label

Check to see if any annotations exist in the frame, if not, the label is null/bckg.

Call get\_class\_id and create the grid for the frame

Loop through all the annotations

Call frame\_contains\_anno on each annotation (if annotation does not exist inside the frame, no need to check for overlap).

Once we find an annotation that overlaps, we check the overlap between the annotation and the frame.

If the threshold is met, we exit and return the label. If not, the label is bckg/null.

* 1. Decoding
     1. Load a model architecture and weights
     2. Predict each frame of every image and output to hyp file
     3. Raw output, no processing of the output
  2. Scoring
     1. Frame Based
        1. Poly to Frame
        2. Prepare frames for full image
           1. Convert TSE files to frame by frame annos
           2. Join HYP file annotations
           3. Compare TSE and HYP overlap
     2. Full-image
        1. Heuristic postprocessor
           1. Sweep through Values N, T on training eval set
           2. Choose optimal N,T combination to post-process the hyp files
           3. Score full image TSE/HYP files (one event per image)
        2. CNN postprocessor
           1. Possibility of using a CNN to output full image label
           2. Train on HYP output frames (histogram of predictions)
           3. Decode on HYP output of eval set to convert to full image label and score

1. Results
   1. Database
   2. Evaluation Methodology
   3. Pilot experiment
      * 1. Existing deep learning systems for cancer detection
        2. Performances
   4. Pilot experiments
      1. Full image results
      2. Frame level results
      3. Parameter sweep results
      4. ROC Curves
2. Conclusion and Future Work

**Tables**

1) Inter-rater agreement

2) Statistical analysis with hardware performances

3) Epoch based results

**Figures**

1) Slide Examples from databases

2) A Flowchart of the Deep Learning System

3) ROC curves

4) Frame-based labeling method

5) The original figures from the paper