Preliminary Design Configuration & Architecture

All code will be hosted within our GitHub organization (https://github.com/MLApx/) in the "Main" repository. Whole-slide images with unique anonymized IDs will be hosted on Amazon Web Service's S3 storage cloud. We will use AWS Sagemaker for our cloud instances and all other computational resources, which provides access to customizable GPU, CPU, and memory requirements. Sagemaker instances will be connected to our GitHub repository for code and S3 directories for images to ensure seamless integration and runtime. Below is a figure illustrating the filesystem architecture we use and how information flows through our model. The red lines illustrate how data flows through the system and which programs are used. The black lines indicate directory hierarchy within our GitHub repository.

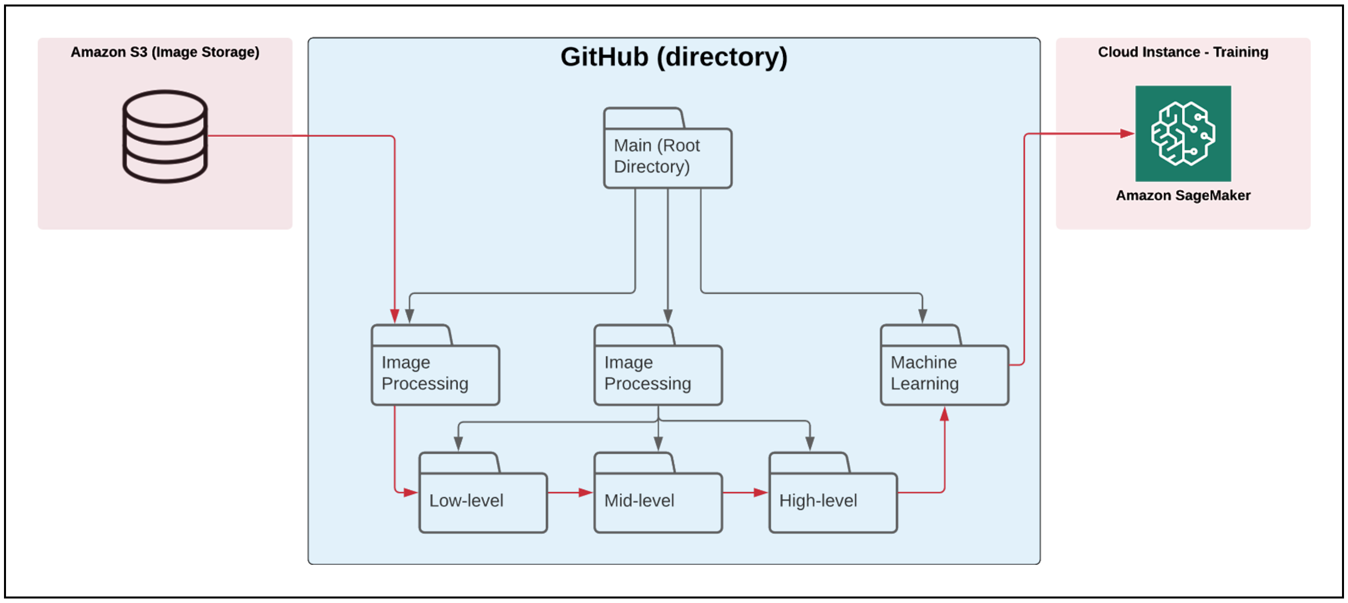

Figure 1: Shows GitHub directory structure and origin and destinations of data.

When examining the system from a functional perspective, we see we can organize the sequence of operations into four primary modules.

1. Image preprocessing
2. Image processing
3. Feature processing
4. Machine learning

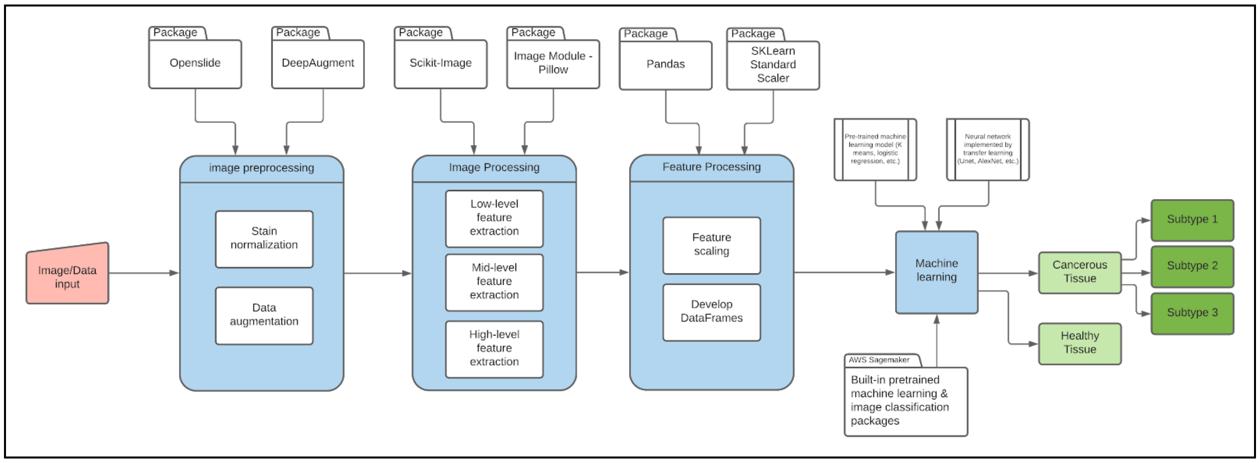

Figure 2: The illustration above details the separate functional components of the system and which Python packages are used in each step. For a full resolution look, see Appendix I.

Below, we will provide an in-depth explanation of each module.

Module 1: Image preprocessing

The preprocessing module starts when the original images (whole-slide .tiff files) are read in from AWS S3 storage into the image pre-processing subcomponent. The package "Openslide" (a C library with a Python API that allows users to read WSIs) is used to separate whole-slide images into a grid of RGB-formatted static (e.g. jpeg) images so that each image represents a smaller surface area of the tissue and is the correct resolution to feed into the image processing algorithms. The next step involves utilizing a package called "DeepAugment" to perform data augmentation on the images. This involves manipulating pixel information to produce rotations, inversions, and color distortions so the machine learning model is able to operate over a wider range of stain and imaging conditions. The next step in the preprocessing stage is to ensure colors, resolution, and other metrics are normalized within predetermined ranges that match real-world conditions. This normalization step will occur in the same program as the static image configuration and data augmentation, but in a separate function. Throughout this entire process, a data structure of our static image database stores all individual images in separate addresses to ensure all sub-functions can perform their relevant operations on each image. This database also stores the geographical relationship of each static image and the correct subtype classification.

Module 2: Image processing (i.e. feature extraction)

After the image preprocessing stage is complete, the image processing (feature extraction) program begins. Using the same database of static images that have now been augmented and normalized, sub-functions within the image processing program begin extracting features and storing them in separate tables with key-value pairs to ensure correspondence with the original static image. The first component of this stage is low-level feature extraction, which uses gradient descent, recursive partitioning, and other methods to extract features such as lines, curves, and contrast regions. The pixel coordinates of these features are stored in tables using vectors.

This information is passed along to the mid-level feature extraction stage (a separate function within the same program). Here, the Scikit and OpenCV libraries are used alongside AWS's semantic segmentation and object detection algorithms to determine the locations of shapes and objects. At this point, features such as mucin pools and cell membranes have been differentiated and their coordinate ranges stored in the data structure.

This data structure is then passed along to the high-level feature extraction algorithm, which uses Scikit along with AWS's built-in object classification and pre-trained ResNet architecture to identify high-order features like signet ring cells, tumors, and layers of tissue. These features are stored in a table of coordinate maps which are then sent to the feature processing stage to be scaled into a DataFrame.

Module 3: Feature processing

The next stage is feature processing, which involves taking the output from the image processing stage and fitting them into a properly scaled object called a DataFrame. This ensures a seamless tracking of all extracted feature objects so they can be quantified, scaled, and fed into the ML subtype classification algorithm. The Python package Pandas is used to convert the high-level extracted features into a DataFrame and SciKit-Learn's StandardScaler module is then used to scale (i.e. normalize) the data to ensure proper reading by the machine learning model. It is important to note that DataFrames are highly useful and manipulatable. Many machine learning algorithms and data visualization packages operate smoothly on DataFrames. Therefore, the team can perform many more functions on the extracted data such as visualization. After a

DataFrame is developed, image attributes such as “quantity of signet ring cells” and “surface area of mucin pools” can be calculated and passed along for predictive modeling.

Module 4: Machine learning

The DataFrame is passed from the feature processing module as input data into the Machine Learning model. Each column of the DataFrame would be selected as input for the model and each row acts as one image or “entry”. The output would be a number representing the probability of a given subtype (during the training stage, the correct subtype output is set to 1 which represents 100 percent probability, and all other subtypes are set to zero). The model will be developed in the AWS SageMaker environment and deployed into production on a cloud server for ease of use. The specific machine learning method to be deployed will be determined by testing various architectures (KNN, SVM, etc.) in parallel and determining which yields the greatest accuracy.

SK-Learn is a python library built for rapid development and testing of different models. It includes all the major models for classification tasks such as K-Nearest Neighbour, Naive Bayes, Decision Tree, and Support Vector machine. The library is so comprehensive that it includes pre-built models that can be easily adjusted without editing a significant amount of code. This will allow us to focus on tuning the models and perform rapid adjustments to create the most accurate architecture possible. The library also has built in functions to visualize and compare performance between models.

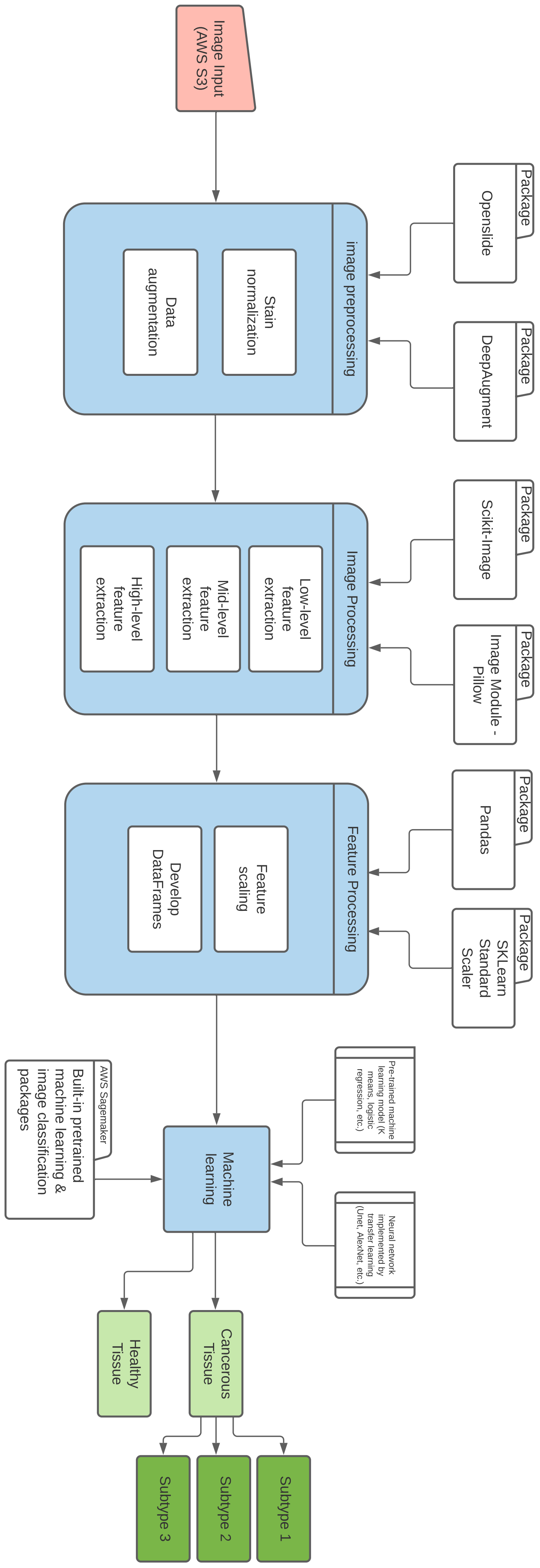

Image Input (AWS S3)
image preprocessing
Stain normalization
Data augmentation
Package
Openslide
Package
DeepAugment
Image Processing
Low-level feature extraction
Mid-level feature extraction
High-level feature extraction
Package
Scikit-Image
Package
Image Module - Pillow
Feature Processing
Feature scaling
Develop DataFrames
Package
Pandas
Package
SKLearn Standard Scaler
Machine learning
Pre-trained machine learning model (K means, logistic regression, etc.)
Neural network implemented by transfer learning (Unet, AlexNet, etc.)
AWS Sagemaker
Built-in pretrained machine learning & image classification packages
Cancerous Tissue
Healthy Tissue
Subtype 1
Subtype 2
Subtype 3