Main components:

1. Data Preprocessing
2. Model Training
3. Testing /Model Tuning

Files:

* Classifier.py (pg. )
* DataHandler.py (pg. )
* ImagePreprocessor.py (pg. )
* DataGenerator.py (pg. )
* DICOM\_reader.py (pg. )
* Main.py (pg. )

File Structure

**Bold** are files.

* data
  + dicom-images-train
  + dicom-images-train-filtered
  + dicom-images-test-unofficial
  + dicom-images-test-unofficial-filtered
  + **train\_labels.csv**
  + **test\_labels.csv**
* Documentation
  + **This document**, and other documentation
* documents
  + **Miscellaneous documents** created by the user to aid in the research or model training
* trained\_models
  + Stores temporary model .h5 files
  + **hyperparameters.json**
* training\_sessions
  + binary
    - cnn
      * 2019-10-29
        + 1

**.h5 model**

**hyperparameters.json**

**periphery\_data.json**

**train\_stats\_x\_of\_n.json**

**…**

**validation\_stats\_x\_of\_n.json**

**…**

* + - * + 2

**…**

* + - * + ….
      * …
    - unet
      * …
    - …
  + segment
    - …
* **[Python files]**

Execution flow of the program:

1. User places DICOM images in “./data/dicom-images-train” and “./data/dicom-images-test-unofficial”. (More detail on pg. )
2. User performs image preprocessing through Main.py’s menu. (More detail on pg. )
3. User trains different types of models with an assortment of hyperparameters through Main.py’s menu.
4. User tests their best models on the test dataset through Main.py’s menu.

Future features:

* Programmatically split DICOM images into training and test set based on ground truth so that training dataset is 50% positive and 50% negative, while test dataset is a natural representation of the ground truth’s ratio.

Step 1: Handling the dataset.

The user currently determines how to split the dataset into train and test datasets, because there’s currently no implemented way to do it programmatically. The test dataset is populated by 20% of the initial dataset. The train dataset is populated by the other 80%.

Location to place the training dataset DICOM files: “./data/dicom-images-train”.

Location to place the testing dataset DICOM files: “./data/dicom-images-test-unofficial”.

The user can place the DICOM files in folders if that’s their preference, but the code retrieves all DICOM images in the aforementioned locations, and considers them all equal.

Dataset labels are annotations in run-length-encoded (RLE) masks. This is the only target label format supported, currently.

EX: 378999 2 1018 8 1013 13 1009 15 1007 17 1006 17 1005 19 1002 21 1001 22 1001 22 1000 23 1000 23 1000 23 998 25 997 26 996 27 996 27 995 28 994 29 994 29 993 30 993 30 992 31 992 31 992 31 991 33 990 33 990 33 990 33 990 33 990 33 990 33 991 32 991 32 991 32 991 32 990 33 990 33 990 33 989 34 989 34 988 35 988 35 988 35 989 34 989 34 989 34 989 34 989 34 989 34 989 34 990 34 989 34 990 33 991 32 991 32 992 31 992 31 992 31 993 31 992 31 993 30 993 31 993 30 993 30 994 30 993 30 994 30 994 29 994 29 995 29 994 29 995 29 994 29 995 29 995 29 994 29 994 30 994 30 993 30 993 31 993 31 993 30 994 30 994 30 991 32 990 34 990 33 991 33 990 33 991 33 991 33 990 34 990 33 991 33 990 34 990 34 990 34 990 33 991 33 990 34 990 34 990 34 990 34 990 34 989 35 989 35 989 35 989 35 989 35 989 35 989 35 989 35 989 35 989 35 989 35 989 35 989 35 989 35 989 35 989 35 989 35 989 36 988 36 988 36 988 37 987 37 987 37 987 38 986 38 986 38 986 38 986 39 985 39 985 39 985 40 984 40 984 41 983 41 983 42 982 42 982 42 982 43 981 43 982 42 982 43 981 43 981 44 980 44 981 43 981 44 980 45 980 44 980 45 979 45 980 45 979 46 978 46 978 47 977 47 978 47 977 48 976 49 975 50 974 50 975 50 974 51 974 50 974 51 974 50 974 51 973 52 973 51 973 52 973 51 973 51 974 50 974 51 974 50 974 50 975 50 974 50 974 50 975 49 976 48 977 47 978 46 979 45 980 43 982 42 983 39 986 37 988 34 991 32 993 29 997 25 1002 21 1006 16

The dataset labels shall be placed in csv files named “train\_labels.csv” and “test\_labels.csv”. These shall be located in “./data”. Each file will have two columns, one for the name of the image file without the extension, and one column for the RLE masks, where -1 means no mask.

|  |  |
| --- | --- |
| ImageId | EncodedPixels |
| 1.2.276.0.7230010.3.1.4.8323329.6904.1517875201.850819 | -1 |
| 1.2.276.0.7230010.3.1.4.8323329.13666.1517875247.117800 | [RLE mask] |
| 1.2.276.0.7230010.3.1.4.8323329.11028.1517875229.983789 | -1 |
| 1.2.276.0.7230010.3.1.4.8323329.10366.1517875223.393986 | [RLE mask] |
| 1.2.276.0.7230010.3.1.4.8323329.10016.1517875220.992175 | [RLE mask] |
| 1.2.276.0.7230010.3.1.4.8323329.11444.1517875232.977506 | -1 |
| 1.2.276.0.7230010.3.1.4.8323329.32219.1517875159.70802 | -1 |

Step 2: Image Preprocessing

In this step, the DICOM images in either the training or testing dataset will have preprocessing performed on them, and then saved as .png files in another folder.

Current image preprocessing techniques available:

* Crop
* Gaussian Blur
* Gradient detection
* Threshold
* Brightening
* Canny Edge Detection
* Custom edge filter

Menu presented:

-- Preprocessor Menu --

Which dataset portion to preprocess?

1) Training dataset

2) Testing dataset

Choice:

The user can specify whether to perform preprocessing on the training dataset images or testing dataset images.

After specifying, the user will be presented with another menu:

Wish to replace existing preprocessed images? (y/n):

If yes, all existing preprocessed images will be replaced.

The preprocessing step is now underway, with the custom preprocess() function in the class specific to the project.

# Custom preprocessing:

1. Custom crop of the radiograph to only include relevant body part
2. resizes to 512x512 pixels
3. Performs Canny Edge Detection, which currently executes
   1. Perform threshold calculation
   2. Subtracts the threshold from the image
4. If the mean of current image is less than a certain intensity, a variable is modified and the image has Canny Edge Detection performed again.
5. If the intensity of the original DICOM image was less than a certain intensity, then the image has its contrast increased.

The final result is a 2D array of pixel intensities, which is saved as a png file in either “./data/dicom-images-train-filtered” or “./data/dicom-images-test-unofficial-filtered”.

Depending on the computer, bulk preprocessing 10,000 images might take 30 minutes+.

# Custom Crop function:

Detecting anomalies in the lungs will be used as an example for a custom crop function.

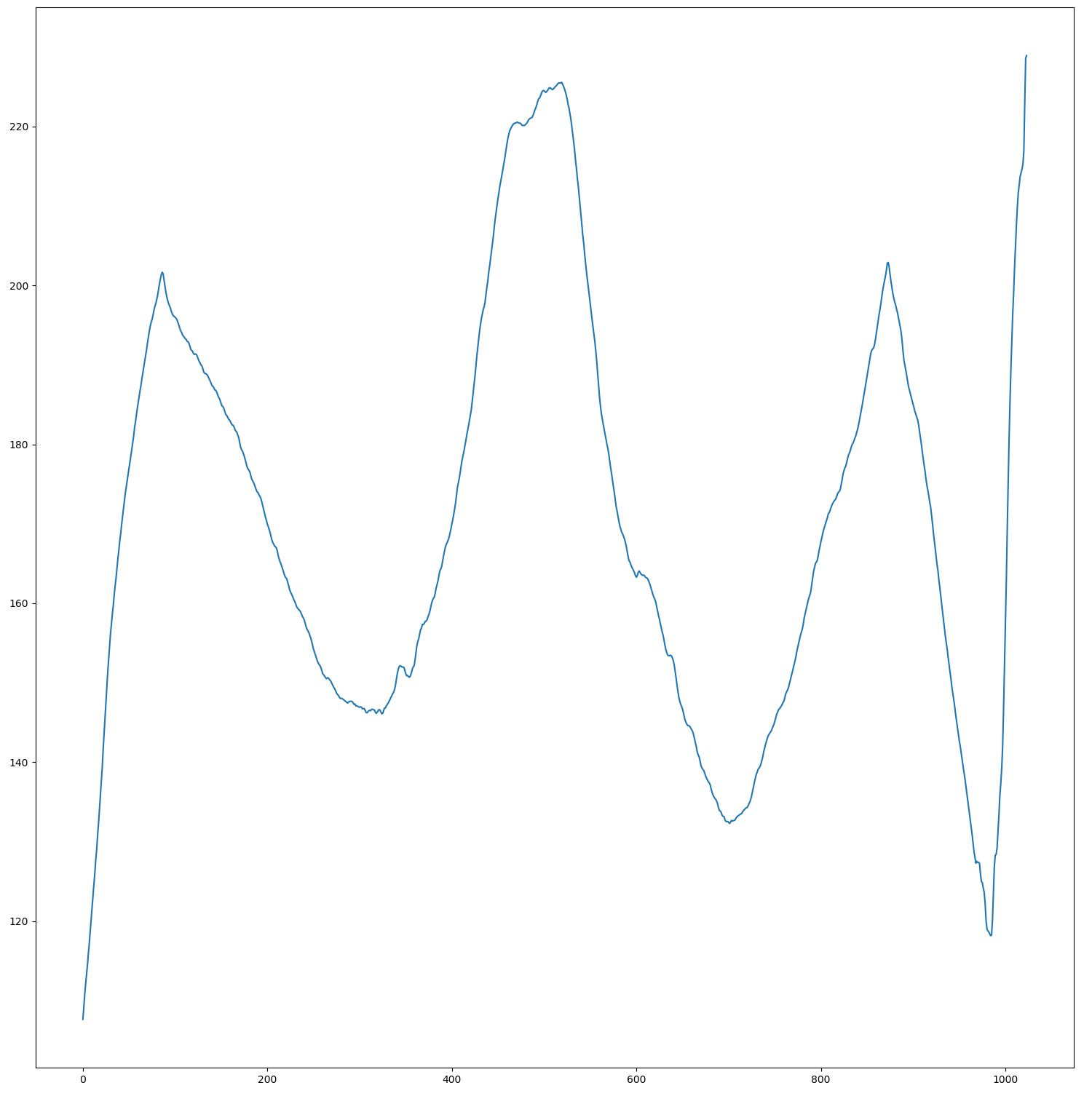
A chest radiograph may include a portion of someone’s neck, shoulders, arms, abdomen, and even be misaligned massively. All of these are unnecessary in capturing anomalies in the lungs, so we want to crop our radiographs to include contain the rib cage. This can be done by programmatically or manually.

Programmatically, we first crop the sides of the image so that the edge of the right ribcage is touching the left side of the image, and the edge of the left ribcage is touching the right side of the image.

Determining where the sides of the ribcage are, can be done by

1. Average each column’s intensity.
2. Run through a local maximum filter to get local maxima of the intensities.
3. Use binary erosion to determine local peaks.
4. Return list of indices pointing to local and global peaks in intensity.
5. Get index closest to middle of image.
6. Get closest peaks to that index.

An example of column intensities:



In this example of the ribcage, the maximum in the middle of the image is the spine, since the spine is bright in comparison to the lungs. The minimums on both sides of the middle maximum, are the lungs themselves. The maxima towards the edges can either be the person’s arm(s), light leakage in the scan, or the edge of the ribcage.

The list of indices of max peak intensities will look like this for the above example:

[98, 515, 866, 1024]

Once we have a list of maxima intensities, we determine the middle of the ribcage by getting the middle maximum, which is usually the spine. The sides of the ribcage are usually the maxima closest to the spine.

In our example, 515 is denoted as the spine since it’s closest to the middle of the image (1024/2 = 512). Because 515 is considered the spine, 98 and 866 are considered the edge of the ribcages. Our image will then be cropped from 98 to 866 on the sides.

Now that we know where to crop horizontally, we determine where to crop vertically. This is so that we only include from the top of the ribcage to the bottom of the ribcage, and not include the person’s neck, abdomen, etc.

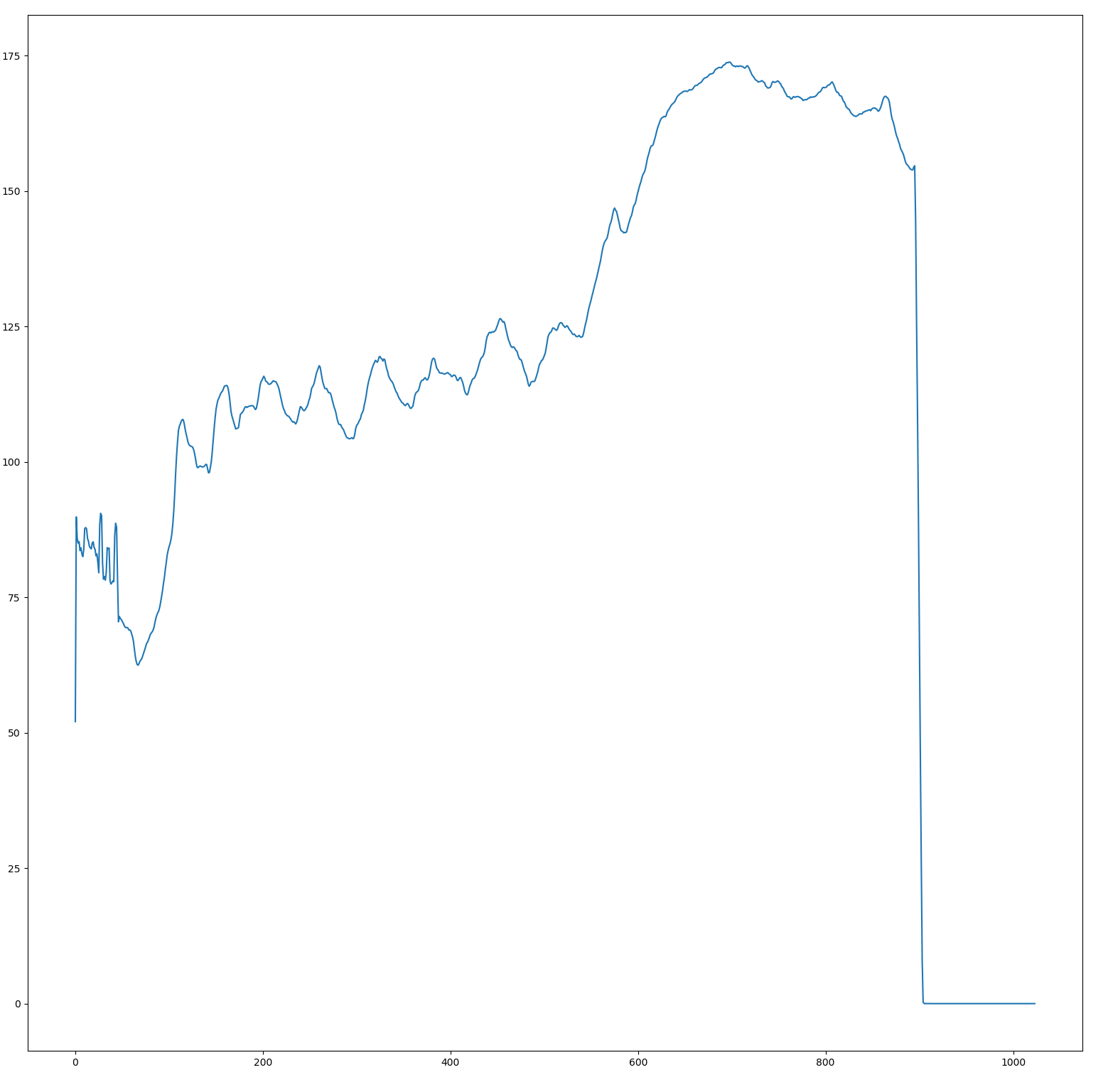
A similar process is performed, but is a different “algorithm”.

Programmatically, we crop the top and bottom of the image so that the top of the ribcage is touching the top part of the image, and the bottom of the ribcage is touching the bottom part of the image.

Determining where the top and bottom of the ribcage are, can be done by

1. Average each row’s intensity.
2. Run through a local maximum filter to get local maxima of the intensities.
3. Run through a local minimum filter to get local minima of the intensities.
4. Use binary erosion to determine local peaks.
5. Return list of indices pointing to local and global minima in intensity.
6. Find local minimum closest to middle of the image.
7. Get index of nearest maxima to the middle minimum.

An example of row intensities:



In this example of the ribcage, the

The list of indices of maxima intensities will look like this for the above example:

[5, 690, 1024]

The list of indices of minima intensities will look like this for the above example:

[0, 89, 307, 1024]

Now we determine the vertical middle of the ribcage by getting the area of least intensity in the middle of the image. This will be considered the middle of the lungs, since they have the lowest pixel intensity when they’re widest.

In our example, 307 is denoted as the vertical middle of the ribcage since it’s closest to the middle of the image (1024/2 = 512, which is closer to 307 than 1024). Because 307 is considered the middle of the ribcage vertically, the maxima closest to this middle minimum, are considered the top and bottom of the ribcage; 5 and 690 respectively. Our image will then be cropped from row of pixels 5 to row of pixels 690.

This method is not perfect, since many scans can have imperfections and bright spots leading to the cropping function to be too aggressive or too passive. It is meant to mass crop thousands of images, then for a human to review the images and manually fix any errors.

# Future functionality

Discussed further in the Future Functionality section on pg.

Future functionality includes:

* user being able to specify which preprocessing techniques to perform.
* Custom cropping function for radiographs of other body parts.

# Step 3: Model Training and Tuning

In this step, the user trains multiple models with multiple architectures using multiple combinations of hyperparameters and multiple ensembling techniques. This is all facilitated with a central menu system, avoiding hard coded values and for streamlining hyperparameter searching.

Current model architectures available:

* CNN
* U-net

Current classification types available:

* Binary classification
* Segmentation

Current model training and tuning techniques available:

* Split of 50% positive and 50% negative for training dataset
* Early Stopping
* Early Saving

Menu presented:

-- Preprocessor Menu --

Which dataset portion to preprocess?

1) Training dataset

2) Testing dataset

Choice:

The user can specify whether to perform preprocessing on the training dataset images or testing dataset images.

After specifying, the user will be presented with another menu:

Wish to replace existing preprocessed images? (y/n):

If yes, all existing preprocessed images will be replaced.

The preprocessing step is now underway, with the custom preprocess() function in the class specific to the project.

Early Saving: Saves model with lowest loss during training

# Statistical Analysis

Step 4: Model Testing