Milestone Requirements

Medical Imaging Matching (1)

Team: Alex Rogov, Emily Medema, Marieke Gutter-Spence, Niklas Tecklenburg

Project Information

Project Description

The Medical Imaging Matching project is aimed at developing a model for BC Cancer that will return a similarity index for two mammogram images. This will help BC Cancer eliminate duplicate mammograms from their data increasing accuracy for further medical research, specifically in the use of AI as duplicates or missing information within a dataset can cause bias. Within this project we have developed various models to detect image similarity as well as developed a preprocessing script to curate datasets and split them into training and testing groups.

User Groups

Group 1: BC Cancer Researchers

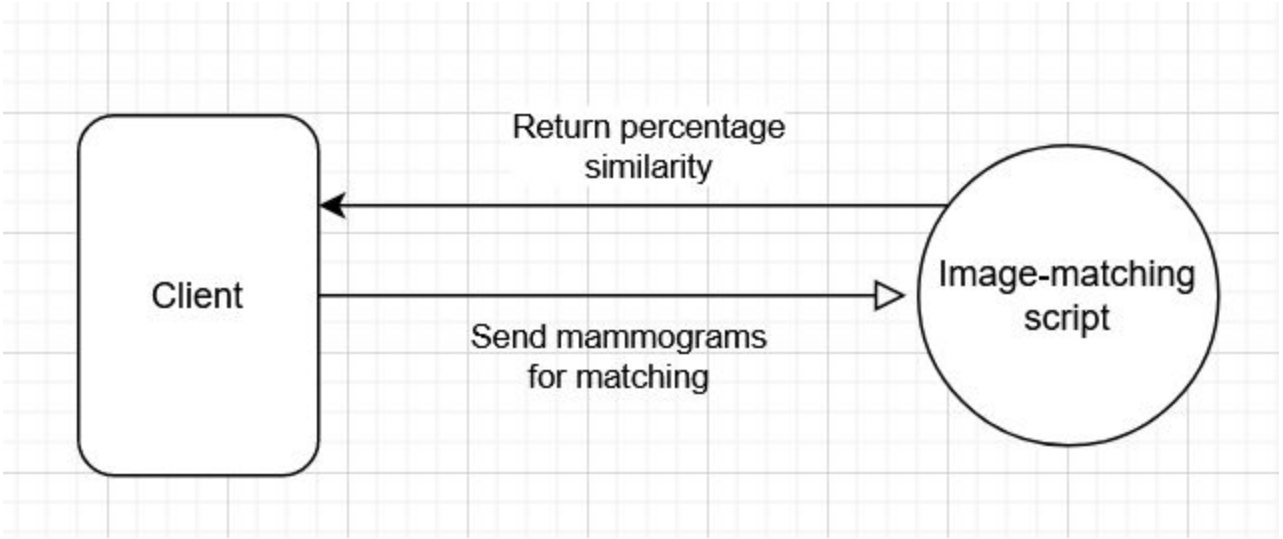
- They will be using the model we develop to consolidate and go through the mismatch in our database.
- End result: a script that they can run, that returns the similarity index of two images.

Possible Group:

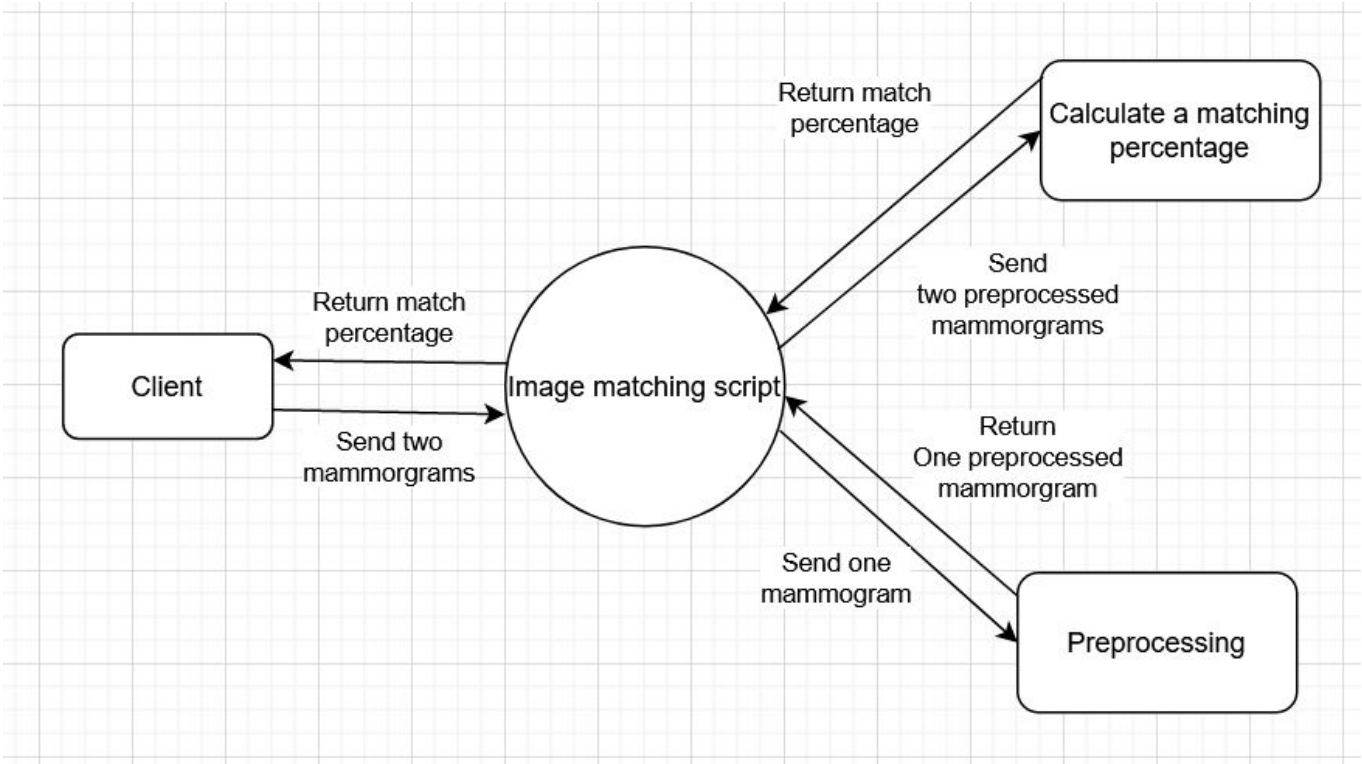
- Eventually publicly available for anyone who wants to curate a large set of images from an institution

System Architecture

Data Flow Diagram - Level 0



Data Flow Diagram - Level 1



Data Flow Diagram - Components

Client - Completed by Peer Testing 1

Clients will see a match percentage response to their mammogram input via the command line. If time is given, a UI can be created for better user experience.

Image Matching Script - Completed by Peer Testing 2

This component holds the logic of the script. It will hold the mammograms and send them for preprocessing. Once enough preprocessed mammograms are collected, it will send it to a separate component to calculate a match percentage. Once a percentage is returned, it will return this to the client. An additional feature could be holding a list of mammograms to match with each other.

Data Flow Diagram - Components

Preprocessing - Completed by Peer Testing 1

This component “cleans” and prepares the mammogram for matching. This would include cutting the file and highlighting/tagging important sections of the mammogram. This will improve the speed and accuracy of the similarity match .

Calculate a matching percentage - Completed by Peer Testing 2

This component takes two preprocessed components and compares the similarities of the image. Based on built-in parameters of comparison, it will return a similarity match percentage to the main “image matching script” component. Aswell, many smaller tests will occur in which mammograms will be altered and scored. These values will come together to form an overall score/percentage which is returned.

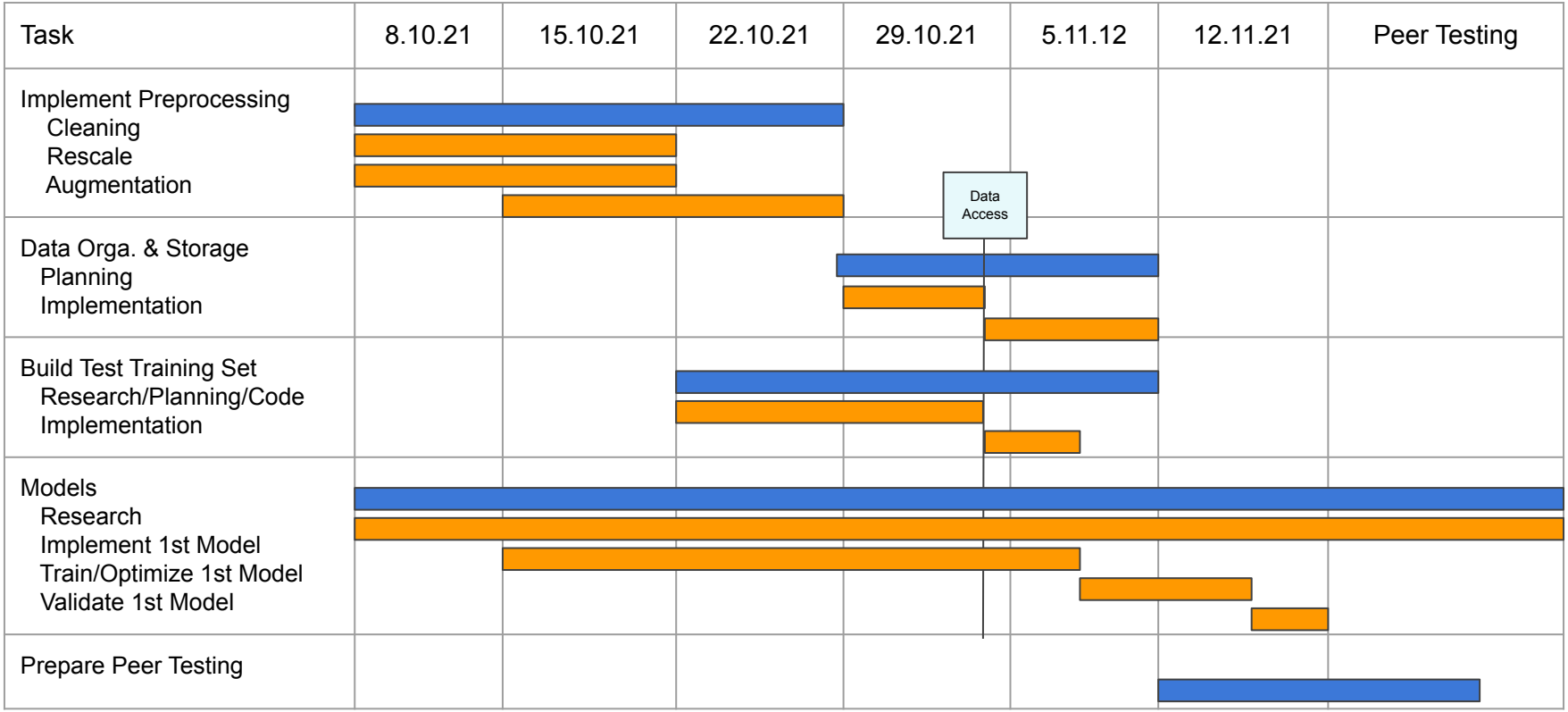
Requirements

Functional Requirements

Requirement Milestone Features

- Python library research and decision.
- Model research.
  - Implementing our own model.
  - Adapting already existing model.

Peer Testing #1 Milestone Features



Peer Testing #2 Milestone Features

- Develop more complex models based on the gained insights
  - Apply Transfer Learning (Using different CNN/DNN)
- Optimize Preprocessing
- Adjust Test/Training set building
- Research on Hyperparameters
- Adjust Hyperparameters

Final Milestone Features

- A model with 70-90% accuracy.
- Possible UI implemented for easier use of model.

Non-Functional Requirements

Non-Functional Requirements + Constraints

- Model must use Python as the language.
- BC Cancer data must not leave their network.
- Model must read in mismatches, compare, and return information about similarity in an output (mostly csv).

Tech Stack

Tensorflow vs. PyTorch

Tensorflow

- Well documented.
- Models sometimes feel like they are behind a wall.
- Tensor board offers good visualisation.
- Easier to deploy.

PyTorch

- More dynamic implementation, feels more native than tensorflow.
- Easier to debug.
- Easier to parallelize.

SciKit-learn

https://scikit-learn.org/stable/

Pros:

- Very user friendly.
- Efficient for predictive data analysis.
- Well documented, open source.
- Deliberately limited scope.

Cons:

- Less extensive than tensorflow for example.
- Not specifically useful for deep-learning.
- Not the best choice for in depth projects.

Model Comparison

Euclidean Distance Model

- Use when calculating the distance between two rows of data containing numerical values, for example 2 pixels.
- Measure the ‘similarity’ between vectors.

RootSIFT

- Used to describe and detect features within digital images, through locating key points and descriptors, often used for object recognition.
- RootSIFT is a descriptor that is used for feature extraction, due to its invariance to scale, rotation, illumination, viewpoint, and translations.

Model Comparison

Siamese Networks:

- Very powerful networks used in face recognition.
- Contains 2 or more identical subnetworks.
- Requires more training time than other network models.
- Finds similarity through comparison of feature vectors.

MSE/SSIM model:

- Simple technique in order to see how similar 2 images are.
- MSE will calculate mean square error (not highly indicative of similarity), SSIM will look for similarities of pixels between two images.
- SSIM takes texture into account.

Adapting an Existing Facial Recognition Model

Adapting an existing facial recognition model would be a great option to use for our project, but with this idea arises a few issues. The main issue with this approach is finding an open source model with identifiers that match. The open source data we found for facial recognition, is not alike due to the fact we have existing potential matches already. Training this model without matching data will be difficult.

Testing

unittest

- Implement unit tests within Model
  - for each method within the model, we will write a unit test.
  - if we adapt a model, we will add any missing unit tests.
- Implement unit tests within Preprocessing
  - for each method within the preprocessing script, we will write a test.

We will be implementing unit tests within our scripts using the built-in python library, `unittest`

- Each method, function, or model will be tested by individual tests within the testing script.
- We will test that everything is functioning and returning the value expected.

Integration Testing

As there are multiple scripts and interesting data requirements within this project, integration testing is a must.

We will need to test that:

- Preprocessing and model work together properly.
- Data can be accessed safely and securely.
- Data is not leaving their network.

These tests will also be written and performed using `unittest`

Regression Testing

In order to ensure we aren't breaking our code whenever we add new code, we will utilize regression testing.

In particular we will use the `pytest-regtest` python plugin to check that the format of the output remains the same as before (ie. no errors).

Function Testing

The one essential element of our project is the return of a similarity percentage or index. Our function tests will including testing that:

- We are selecting the correct images to compare in accordance to the mismatches.
- We are getting a similarity result that makes sense:
    - If percentage, it is between 0-100 not negative or above 100 etc.

This will use a combination of the aforementioned testing in order to ensure the business requirements are met.

CI/CD

CI/CD

In order to ensure CI/CD and that our code does not break when we make changes we will do the following:

- We will run premade test cases on each model we develop.
  - These test cases include a set of mammograms in which we already know the results (mammograms match or not). If the success rate is too low, we will know we must improve the model.
- Configure static analysis, specifically the `pre-commit` framework so that our code is consistently formatted and linted.
  - This will run via git-hooks and will catch things like whitespace and common issues.