# COSC 499: Peer Testing #1 Report

*Medical Image Matching (1)*

*Marieke Gutter-Spence, Emily Medema, Alex Rogov, Niklas Tecklenburg*

## Medical Image Matching (MIM) System Overview

### *Overview*

The MIM project's goal is to improve the quality of the mammogram database at BC Cancer in the long term by being able to determine if two mammograms are from the same patient. It has been shown that there are regular mismatches in the data such as the same name or the date of birth, while the other data does not match and vice versa. In order to determine if the mammograms are from the same patient, a radiologist has to be brought in, which is both time-consuming and expensive.

This is where our project comes into play. We can estimate similarity and minimize the number of mismatches by reducing the two mammograms to their most essential information (embeddings) with the help of a CNN and determining the distance between these embeddings (and thus the similarity).

In this report, we provide an update on the progress of the development of this software. We will briefly discuss the user perspective, but the main focus will be on the MIM features themselves. These include the preprocessing, a breast detector that extracts the area of interest, a batch builder that feeds the model with optimal data during training, and the model itself as well as its training progress and the problems we are currently facing.

### *User Perspective*

The user should find our project easy to use. First, the user needs to make sure the images file names are placed in the relative location of *"data/mismatches.csv"*. Then the user simply needs to run the "main.py" file in a terminal window. Once the script begins, the following steps occur:

1. First, the script calls the query() function which iterates through each image in mismatches.csv and checks if a preprocessed image version exists. It returns all the paths of the images that are not preprocessed (ie. they are new images to the algorithm).
2. Then, the script calls the preprocess() function for each unprocessed image which sends the image for preprocessing. The preprocessed image is then returned back to the script.
3. Finally, all the processed images are sent to our model for a percentage similarity to be calculated.

In the end, the user should see on the terminal which images were run and see the percentage similarity of each image processed.

**MIM Features**

*Preprocessing*

Our preprocessing section of the code deals with formatting and editing the mammograms we are analyzing. This ensures that the images we feed the model are of the size and file type we expect. Specifically, this section of code will clean and remove noisy data, resize the images (we are currently working with 256x256 images), and save the image in the correct location (according to the type of mammogram) in the expected file type (NumPy array). We are also able to augment the images via zooming, flipping, and other methods. This is helpful for us as it allows us to artificially grow our sample size and therefore increase the accuracy of our model.

Data cleaning is the first step in our preprocessing script. Our code will iterate through the mammograms, clean them, remove any duplicates, handle any missing data, and flag numeric and non-numeric columns. We then move to `preprocess_data` which is the main chunk of the preprocessing code. This script will iterate through the stated location of the mammograms. It will then check to see if the mammogram has already been processed or not. If it has, it will return the location of the already processed mammogram, if it has not, it will continue with the preprocessing. The preprocessing consists of loading the image, resizing it, and then saving it in accordance with the mammogram type (ie. LCC, RCC, RMLO, LMLO) as a NumPy array.

The preprocessing section of our code is run any time the model is run. If we do need more images, we will run the data augmentation scripts which create altered images of the original mammograms. Specifically, it will create a zoomed image, a shifted image horizontally, a shifted image vertically, a brighter image, a vertically flipped image, a horizontally flipped image, and a rotated image.

*Breast Detection*

This algorithm uses a machine learning model that was trained on mammograms to find and crop the breast. This breast detection algorithm is a part of the preprocessing component of our algorithm since it provides two important services:

1. Our main similarity model (CNN/Triplet Loss) works better the more mammograms we use. However, these mammograms are relatively large and take up between 1.5MB and 5MB of memory. Thus, training a model with hundreds of these mammograms would take a long time. Therefore, we want to decrease the size of the mammogram while not excluding any part of the breast. Each mammogram has a different size breast which means we are unable to statically crop at a given part of the mammogram. To solve this we created our breast detection algorithm, which can recognize where the breast is and crop around it.

2. The algorithm improves the accuracy of the similarity model (CNN/Triplet Loss) by

removing unnecessary data. In a mammogram, the black region around the breast can potentially skew the matching process because the algorithm will see that the black regions of the two mammograms are the same, resulting in a false positive! Most mammograms will have this black region which means each mammogram will be very similar to each other. To remove this unwanted matching, the algorithm removes the black spaces by cropping in only the breast along with a little bit of surrounding black space.

This breast detection algorithm can be broken into 3 steps:

1. First, raw images are placed in a specific folder. The algorithm will preprocess all these images.
2. Through a script, the pre-trained model is used on each of these images:
    a. First, it finds the breast in a mammogram
    b. A box around the breast is created
    c. Then, the algorithm crops out everything around the box.
3. Finally, the parsed images are then returned into another folder and are ready to be used

*Batch Building*

The Batch builder feeds the model with optimal data during training. For our model, we need to train a customized neural network that yields good embeddings. The process of obtaining the values of the constants, called weights, that fundamentally constitute a neural network is known as training. This is accomplished by taking training data with known output values and then determining weights and biases that minimize the error between computed output values and the known values. We can obtain these weights more efficiently through the batch building process.

However, we first need to decide how to determine the batches with which we will be providing the CNN model. For any arbitrary sample, one can obtain the embedding vector, but it may not be helpful for our model. In order to build batches that are helpful for the training of the model, we need to create our batches in a more purposeful way than simply randomly selecting from our training data.

Each of our batches needs to be a set of three mammograms. The concept of the triplet loss function we are using for training is to evaluate small distance embeddings for similar photos and high distance embeddings for different images as we want to compare two pictures. This means we need three mammograms that fit the following criteria:

- The Anchor is the first image.
- The Positive is a photograph taken in the same class as the anchor.
- The Negative is a photograph from a class other than the anchor.

Based on our research, we have also found that in order to increase our model's performance, we need to build batches that focus on the hardest triplets first. We have also determined that our batch builder must store embeddings using a call-back function as that is the most efficient way.

Once our batches have been built, they are sent through the aforementioned call-back function to the CNN/Triplet Loss for training.

*CNN/Triplet Loss*

The CNN takes the mammogram as input and returns a low-dimensional embedding vector with a size that still needs to be determined. These embeddings can be seen as the condensed content of the mammogram. Given these embeddings for two or more mammograms, we can comment on the breasts' similarities, shown in the mammograms. To train the CNN to return relevant embeddings, we utilized a triplet loss function and fed it two mammograms from the same patient and one from a different patient. We then used the triplet loss function gradient descent to build the embeddings for the CNN. By adjusting the weights, the embeddings from the breast from the same patient get closer together and from different patients separate more, the model learns to condense the mammograms down to their main content.

At peer testing, we had this main architecture in place and working for very simple but similar tasks using a very simple CNN Architecture. Starting from there, we spent the last months experimenting with the actual data on the GPU server, using different architectures of CNNs, and transfer learning to get a “kick-start” in the learning process. We have been using mainly VGG architectures, ResNets, and pre-trained weights on the ImageNet database.

One issue we are facing is the fact that mammograms can be represented using a grayscale without losing a lot of information. On the other hand pre-trained models all rely on RGB images, this makes the further training more complex than it would need to be. Therefore, we are also experimenting with training the mentioned architectures on grayscale images without transfer learning. Likewise, we are confronted with the problem that ImageNet weights give very similar if not the same embeddings for all mammograms, which results in our cost function returning a loss of zero, causing the model to not train.

We are currently still researching and working on this training problem. Other than exploring several hyperparameters and setup options, we are validating our implementation with a simpler dataset, to see whether we can learn from these. If this is the case, it would imply we would need more data than we currently have but the results in this test are not yet complete.

**User Feedback**

We met with our sister team in order to gain feedback and assess the progress of our projects during our peer review session. Our team showed our prototype video and each member of our team went over the parts of the video we had completed. Our sister team had no pressing questions for us.

Team (0) shared their screen with us and went over snippets of their code for the transformer model, as well as carrying out a sample run to show how the model is working as of now. We questioned how long it may take to train their model; if re-training would be necessary; how come their model is not training correctly; and if they are facing similar issues to us using the transformer model.

**List of Issues**

The biggest issues we are currently facing are in the training of the CNN. As explained earlier, our model is not able to make any progress or progresses very slowly. We are currently researching and experimenting to eliminate potential causes one by one.

Additionally, we need to develop more testing for our model. Specifically, we need to work on integration testing between our features.