ROSE-HULMAN
INSTITUTE OF TECHNOLOGY
Engineering World Health

Image-Based Jaundice Detection App

Project Submitted for Review

EWH Design Competition
May 30, 2018

Submitted by:
Rose-Hulman Institute of Technology Student EWH Club

Team Members:
B. Katz, M. Baker, H. Bach, D. Conrad, D. Moore, K. Morozin, S. Peterson, M. Routon
A. Boyer, B. Chang
Dr. D. Walter

Problem Definition

Jaundice occurs when the liver stops functioning correctly and bilirubin levels build up in the body creating a yellowish tint in the skin and eyes. Diagnosing jaundice in newborns and adults is very difficult and very unlikely since there are often times no medical professionals around and tests are expensive. What our team would like to do is make this diagnosis easier to attain. We are proposing an android app that uses a cellphone's camera to potentially diagnose jaundice in newborns. This diagnosis will be done in software using image recognition techniques to detect discoloration in the skin and eyes.

Statement of Impact in the Developing World

Jaundice is a condition that affects about 60% of newborn infants (about 80% of premature infants) (Greco, et al. 2016), and if left untreated can lead to brain damage or death. This is especially concerning in developing countries since around 70% of women give birth at home with no skilled healthcare workers present (Montagu, et al. 2011). The detection of jaundice can be tricky especially on infants with darker skin tones. Early detection of jaundice decreases the risk of permanent brain damage and leads to an easier treatment.

Currently there are a few methods of detecting jaundice, the most common being to perform a bilirubin blood test. Another method is to use a handheld device that is placed on the skin. The problem with these methods is that you either must visit a hospital or own this handheld device, and both of these options are very pricey. Our solution will be using the existing android phones that the users likely have. They will simply download the app that we create and they will have a free jaundice detector. Our app will help the user make the decision on if they need to travel to a doctor. Our hope is to help parent make safer decisions about the healthcare of their newborns, to know whether or not the long and expensive trip is worth it.

Required Performance Specifications

The code used to create the SVM as well as the programming of the app must be robust enough that the app is able to be downloaded once from the google play store and then able to be used for multiple families. In order for the app to be considered effective, the SVM must have first achieved an accuracy of at least 90% from the images used to train, validate, and test the SVM. Once this SVM has been determined, the app will be able to be used offline so that it can be used in communities that do not necessarily have reliable service or internet and still provide useful feedback and information to the families in need.

Implementation of Prototype

To create a prototype, the design group was split into two teams: the SVM team and the app development team.

The SVM team developed code using MATLAB that analyzed groupings of pictures in order to determine an appropriate support vector machine (SVM) to distinguish between the two groups being analyzed: patients with jaundice and healthy patients. The SVM was determined by sectioning each picture used into a grid and determining the average LST values (as well as the standard deviation) for each subsection in the grid and comparing the LST values of training images to those used to validate and test the SVM developed. Once this SVM was developed, it was extracted from MATLAB and used by the app development team.

The app development team was made up of two students who used their android app development skills to create an app that used the SVM determined to classify new images of eyes taken in the app as either “Jaundice” or “Non-Jaundice.” Based on the SVM results, the app would either recommemend the user to seek medical help for further diagnosis or run the app again after a certain amount of time had passed to check on the progression of symptoms. This app stores these tests for any future reference as well as helps open maps to show the nearest doctor and provides a page of information about the condition and the symptoms associated.

The app was designed to be easy to follow and not require much reading since the exact education level and language of the user is unknown. Because of this stipulation, there are only a few pages within the app. The app was also designed to work offline (which is why the SVM method was used) and store results for multiple families so that even remote villages with limited access to phones would be able to make use of the technology.

Proof of Performance

The SVM was trained on 100 images split evenly between jaundice and non-jaundice images. It was then validated with 15 jaundice images and 15 non-jaundice images. The validation accuracy was 90%. This was achieved with a kernel scale of 2 and a box constraint of 1. It had the highest accuracy and the smallest number of support vectors to prevent overfitting. This final SVM was then tested with 39 jaundice images and 39 non-jaundice images. This lead to an accuracy of 80.7%. An ROC curve was produced on the final SVM [Appendix 1]. An optimal threshold was found of -.0915 that would both maximize the true positive rate while minimizing the false positive rate. This improved the overall accuracy to 83.3%. Table 1 shows how well the threshold preforms. The upper left is the true positive rate while the the bottom left is the false positive rate. It's also important to note that the number of false negatives is the smallest percentage in the upper right. This is where the SVM says that the patient doesn't have jaundice but they actually do which would be the worst case situation.

Table 1. Accuracy rates for the SVM determined using the provided images.

	Predicted Jaundice	Predicted Not Jaundice
Actually Jaundice	84.62%	15.38%
Actually Not Jaundice	17.95%	82.05%

Business Plan

Before proceeding, further development of the SVM is needed. Only 104 images of eyes inflicted with jaundice and 104 images of healthy eyes were used to train, validate, and test the SVM (this is due to the limited resources available). With those numbers, we were able to produce a respectable accuracy, but before this product is to be released to the market, better accuracy is needed. To accomplish this, many more images will need to be acquired to better train, validate, and test the SVM used in the app. Once the accuracy desired has been accomplished (at least 90%), the app will be released onto the Google Play store for free download.

Once the app has been released, the club advisor has agreed to run support and to train future member of the club on how to update and make the app better for the user.

The app will be marketed and available for use by anyone who has an android phone. This ensures that people who do not have easy access to a doctor can still make sure to take the best care possible of those around them and determine if further intervention is needed.

References

Greco, Chiara, Gaston Arnolda, Nem-Yun Boo, Iman F. Iskander, Angela A. Okolo, Rinawati Rohsiswatmo, Steven M. Shapiro, et al. 2016. "Neonatal Jaundice in Low- and Middle-Income Countries: Lessons and Future Directions from the 2015 Don Ostrow Trieste Yellow Retreat." *Neonatology.*

Montagu, Dominic, Gavin Yamey, Adam Visconti, April Harding, and Joanne Yoong. 2011. "Where Do Poor Women in Developing Countries Give Birth? A Multi-Country Analysis of Demographic and Health Survey Data." *PLoS One.*

Appendix

Link for the code:
Android App: https://github.com/boyeram1/JaundiceDetection

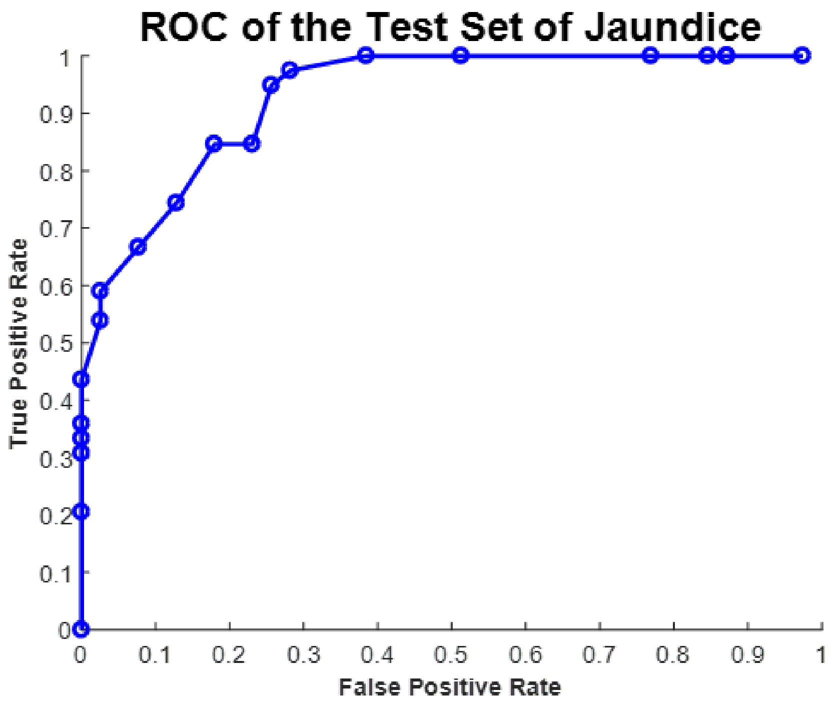
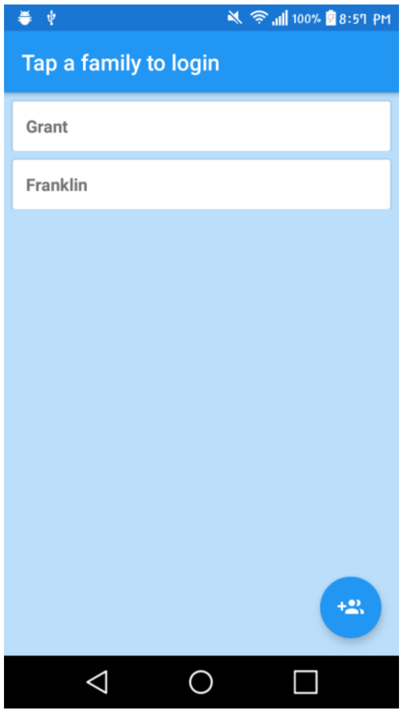
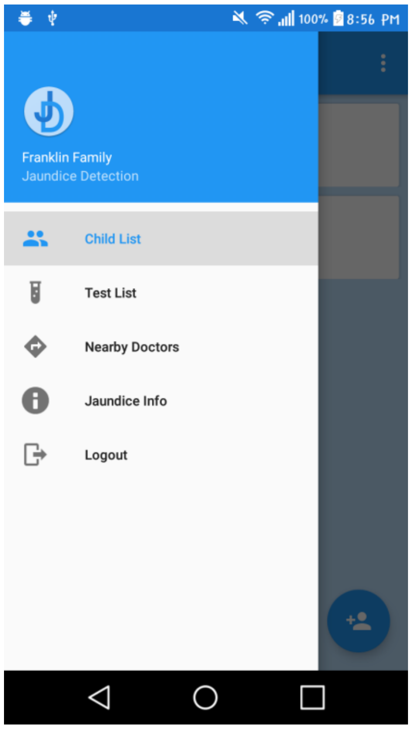


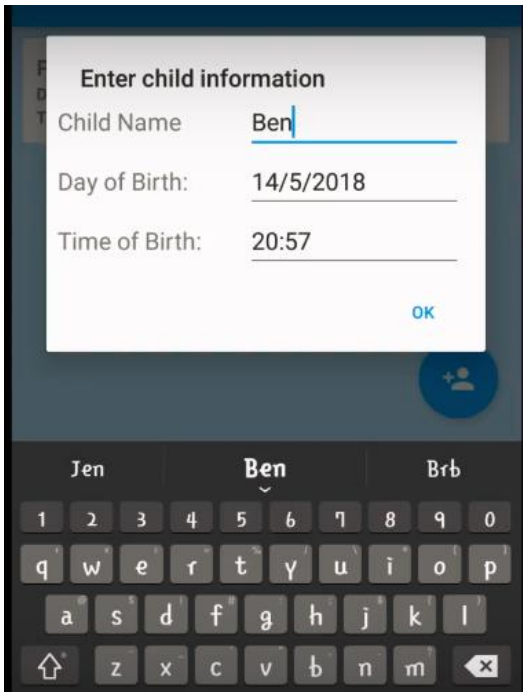
Figure 1. The ROC curve showing the false positive rate and true positive rate for the SVM.


Tap a family to login
Grant
Franklin

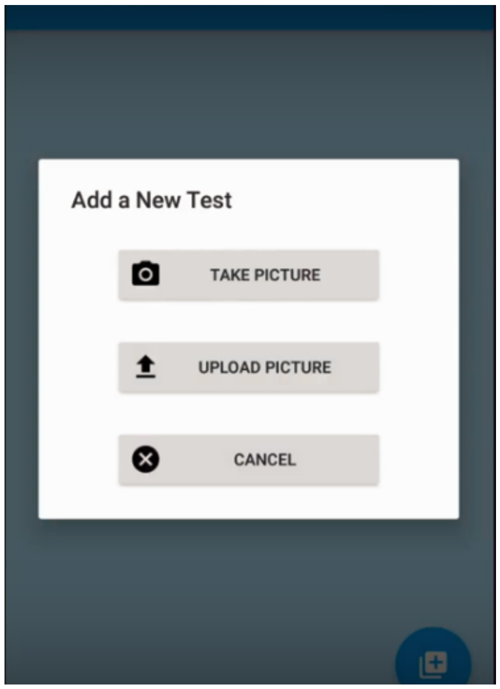
(a)


Franklin Family
Jaundice Detection
Child List
Test List
Nearby Doctors
Jaundice Info
Logout

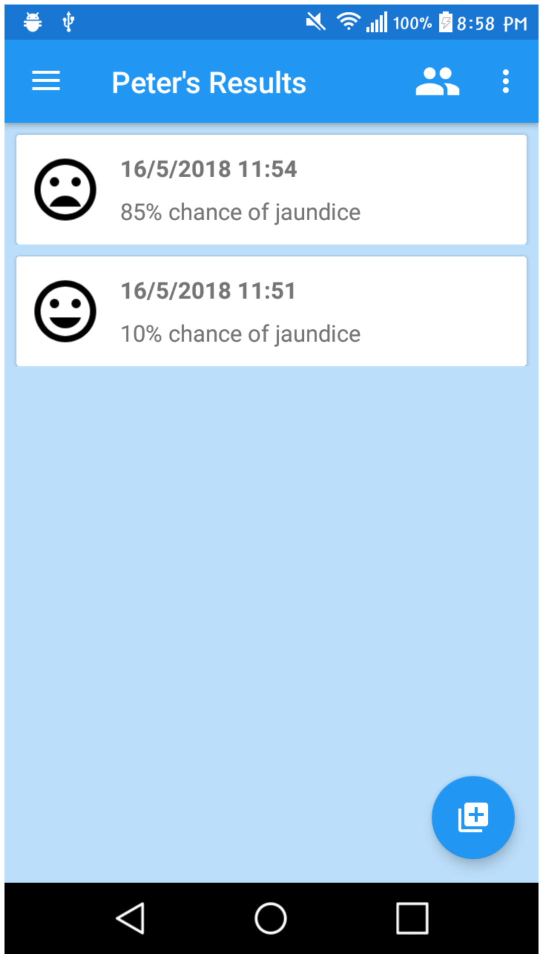
(b)


Enter child information
Child Name
Ben
Day of Birth:
14/5/2018
Time of Birth:
20:57
OK

(c)


Add a New Test
TAKE PICTURE
UPLOAD PICTURE
CANCEL

(d)



Peter's Results

16/5/2018 11:54
85% chance of jaundice

16/5/2018 11:51
10% chance of jaundice

(e)

Figure 2. Screenshots of the app showing (a) the login page, (b) the main menu, (c) how to enter child information, (d) how to take a new test, and (e) what the results look like.

Code for SVM:

```
%% Load all data and extract features
clc;
clear all
%FLAGS
RELOAD_IMAGES = 1;
RETRAIN_SVM = 1;
RETEST_SVM = 1;
CREATE_ROC = 1;

if ~exist('totalData.mat','file') || RELOAD_IMAGES
    rootdir = [pwd '/'];
    subdir = [rootdir 'train'];

    trainImages = imageDatastore(...
        subdir, ...
        'IncludeSubfolders',true, ...
        'LabelSource', 'foldernames');

    % Make datastores for the validation and testing sets similarly.

    fprintf('Read train images into datastores\n');

    dataTrain = imageDatastoreReader(trainImages); %returns the feature data
2d
    % array, rows are pictures, columns are datapoints
    % column datapoints [Lmean, Lstdev, Smean, Sstdev, Tmean, Tstdev] LST is
a
    % color space that combines red, green, and blue
    labelTrain = trainImages.Labels;



    subdir = [rootdir 'validate'];
    validImages = imageDatastore(...
        subdir, ...
        'IncludeSubfolders',true, ...
        'LabelSource', 'foldernames');
    % Make datastores for the validation and testing sets similarly.

    fprintf('Read validation images into datastores\n');

    dataValid = imageDatastoreReader(validImages);
    labelValid = validImages.Labels;



    subdir = [rootdir 'test'];
    testImages = imageDatastore(...
        subdir, ...
        'IncludeSubfolders',true, ...
        'LabelSource', 'foldernames');
    % Make datastores for the validation and testing sets similarly.
    fprintf('Read test images into datastores\n');

    dataTest = imageDatastoreReader(testImages);
    labelTest = testImages.Labels;


    total = [dataTrain; dataValid; dataTest];

    %need to normalize data all together to have global max and global min
    normTotal = normalizeFeatures01(total);

    [r1, c1] = size(dataTrain);
    [r2, c2] = size(dataValid);
    [r3, c3] = size(dataTest);

    dataTrain = normTotal(1:r1, 1:c1);
    dataValid = normTotal((1+r1):(r1+r2), 1:c2);
    dataTest = normTotal((1+r1+r2):(r1+r2+r3), 1:c3);
    save('totalData.mat');
else
    load('totalData.mat');
end

%% Train and evaluate an SVM
if ~exist('SVMTraining.mat','file') || RETRAIN_SVM
    k = 100; %kernel scale
    b = 10; %box constraint
    rate = 1;
    information = zeros(k/rate, b/rate, 2);
    fprintf('\nStarting Training (Finding best SVM\n');

    for kernelScale = 1:rate:k %start with higher kernel
        for boxConstraint = 1:rate:b
            %create the svm called net with the given parameters
             net = fitcsvm(dataTrain, labelTrain, 'KernelFunction', 'rbf',
'KernelScale', kernelScale, 'BoxConstraint', boxConstraint);
             %see how different it is with predict
             [detectedClasses, distances] = predict(net, dataValid);
%IMPOTANT use validation data to compare

             %find the stats of the svm like its missclassifications and
number
             %of support vectors
             error = 0;
             for i = 1:length(labelValid)
                  if labelValid(i,1) ~= detectedClasses(i,1)
                      error = error + 1;
                  end
             end
             numSuppVec = length(net.SupportVectors);
             information(kernelScale, boxConstraint, 1) = error;
             information(kernelScale, boxConstraint, 2) = numSuppVec;
        end
    end

    %% find which SVM to choose
    %normInfo = normalizeFeatures01(information);
    figure
    mesh(information(:,:,1));
    figure
    mesh(information(:,:,2));
    fprintf('Done');

    minimumError = min(min((information(:,:,1))));
    [scaleMinError, boxMinError] = find(information(:,:,1) == minimumError);
    scaleMinError = scaleMinError(1);    %convert to matrix to integer
    boxMinError = boxMinError(1);

    minimumVect = min(min((information(:,:,2))));
    [scaleMinVec, boxMinVec] = find(information(:,:,2) == minimumVect);

    %use min error
    net = fitcsvm(dataTrain, labelTrain, 'KernelFunction', 'rbf',
'KernelScale', scaleMinError, 'BoxConstraint', boxMinError);
    save('SVMTraining.mat');
else
    load('SVMTraining.mat');
end

%% Test Data
if ~exist('SVMTesting.mat','file') || RETEST_SVM
    [detectedClasses2, distances2] = predict(net, dataTest); %try out the
test pictures
    %find accuracy
    error = 0;
    for i = 1:length(labelTest)
      if labelTest(i,1) ~= detectedClasses2(i,1)
          error = error + 1;
      end
    end

    accuracy = (length(labelTest)-error) / length(labelTest);

    fprintf('\nFinal Kernel Scale = %d\n',scaleMinError);
    fprintf('Final Box Constraint = %d\n', boxMinError);
    fprintf('Final Accuracy = %1.4f\n', accuracy);
    fprintf('Final Number of Support Vectors = %d\n',
length(net.SupportVectors));

    save('SVMTesting.mat');
else
    load('SVMTesting.mat');
end

%% ROC
if ~exist('classification.mat','file') || CREATE_ROC
    %find more stats about the svm's classifcations
    score = distances2(:,2);
    start = min(score);
    stop = max(score);
    rate = .1;
    classify = zeros(length(labelTest));
    accuracy = zeros(int32(1+(stop - start) / rate), 1);
    truePosRate = zeros(int32(1+(stop - start) / rate), 1);
    falsePosRate = zeros(int32(1+(stop - start) / rate), 1);
    trueNegRate = zeros(int32(1+(stop - start) / rate), 1);
    falseNegRate = zeros(int32(1+(stop - start) / rate), 1);
    index = 1;
    for thresh = start:rate:stop
        for i = 1:length(labelTest)
            if thresh <= score(i)    %-if threshhold is less than or equal to 
the score then classify it as sunset
                classify(i) = -1;
            else                    %otherwise it is a non sunset
                classify(i) = 1;
            end

            if classify(i) == 1 && labelTest(i) == 'jaundice'
                truePosRate(index) = truePosRate(index) + 1;
            end
            if classify(i) == 1 && labelTest(i) == 'nonjaundice'
                falsePosRate(index) = falsePosRate(index) + 1;
            end
            if classify(i) == -1 && labelTest(i) == 'nonjaundice'
                trueNegRate(index) = trueNegRate(index) + 1;
            end
            if classify(i) == -1 && labelTest(i) == 'jaundice'
                falseNegRate(index) = falseNegRate(index) + 1;
            end
        end
        accuracy(index) = 
(truePosRate(index)+trueNegRate(index))/length(labelTest); %at this point 
they aren't rates yet so accuracy can be calculated
        truePosRate(index) = truePosRate(index)/(length(labelTest)/2);
        falsePosRate(index) = falsePosRate(index)/(length(labelTest)/2);
        trueNegRate(index) = trueNegRate(index)/(length(labelTest)/2);
        falseNegRate(index) = falseNegRate(index)/(length(labelTest)/2);
        index = index + 1;
    end

    %create a region of convergence curve
    roc(truePosRate, falsePosRate);
    save('classification.mat');
else
    load('classification.mat');
end

%% find optimal threshold
%find the best point on the ROC curve  that maximizes the true positive
%rate, while minimizing the false negative rate
bestDistance = 1.5; %worst case
bestTPR = 0;
bestFPR = 0;
index = 1;
for i = 1:length(truePosRate)
    d = pdist([falsePosRate(i), truePosRate(i); 0, 1]);
    if d < bestDistance
        bestDistance = d;
        bestTPR = truePosRate(i);
        bestFPR = falsePosRate(i);
        index = i;
    end
end


optimalThresh = start+(index-1)*rate;
fprintf('Optimal Jaundice Threshold = %1.4f\n',optimalThresh);
accuracyThresh = accuracy(index);
fprintf('Accuracy using new threshold = %1.4f\n', accuracyThresh);
```

Functions:

```
function toReturn = featureExtract(img, nBlocks)

[r, c] = size(img(:,:,1));
pixR = r/nBlocks;
pixC = c/nBlocks;

img = double(img);

LSTimg(:,:,1) = img(:,:,1) + img(:,:,2) + img(:,:,3);
LSTimg(:,:,2) = img(:,:,1) - img(:,:,3);
LSTimg(:,:,3) = img(:,:,1) - 2*img(:,:,2) + img(:,:,3);

toReturn = double(zeros(1, 294)); %allocate memory

index = 1;
for i = 1:nBlocks
    for j = 1:nBlocks
        for k = 1:3
            startR = int32(1 + pixR*(i-1));
            stopR = int32(pixR*i);

            startC = int32(1 + pixC*(j-1));
            stopC = int32(pixC*j);
            box = LSTimg(startR:stopR, startC:stopC, k);

            toReturn(1,index) = mean(box(:));
            index = index+1;
            toReturn(1,index) = std(box(:));
            index = index+1;
        end
    end
end

end



function features = imageDatastoreReader(datastore)
% Example of using an image datastore.

nBlocks = 7; %
nImages = numel(datastore.Files);

features = zeros(nImages, nBlocks * nBlocks * 6);
row = 1;
```

```
for i = 1:nImages
    [img, fileinfo] = readimage(datastore, i);
    % fileinfo struct with filename and another field.
    %fprintf('Processing %s\n', fileinfo.Filename);
    % TODO: Write and call a feature extraction here to operate on image.
    % Hint: debug this code ELSEWHERE on 1-2 images BEFORE looping over lots
of them...
    featureVector = featureExtract(img, nBlocks);

    features(row,:) = featureVector;
    row = row + 1;
end


% This function normalizes the features to the range [0,1]. For each feature
type,
% for example, Lmean, the min becomes 0 and max becomes 1. This isn't that
robust, because
% a single outlier could compress the rest of the data too much.

% The data is assumed to be in the the format specified in the paper.
% Features is a N x 294 matrix, where N is the number of images.
function features = normalizeFeatures01(features)

nFeatures = size(features, 2); % 294

% Loop over Lmean, Lstd, Smean, Sstd, Tmean, Tstd
for selectedType = 1:6
    % extract all values of the given feature from all 49 blocks from all
images
    selectedFeature = features(:,selectedType:6:nFeatures);
    % Find the min and shift the data so that the min is now 0
    minValue = min(min(selectedFeature));
    selectedFeature = selectedFeature-minValue;

    % Find the max and divide by it to make the max 1
    maxValue = max(max(selectedFeature));
    selectedFeature = selectedFeature / maxValue;

    % We could have combined: f = (f-min)/(max-min)

    % Re-insert into the feature matrix
    features(:,selectedType:6:nFeatures) = selectedFeature;
end

features = double(features);


function toReturn = roc(truePosRate, falsePosRate)
% Plot an ROC curve for an image.
% This is basically just an example of the plot function.
% The help documents provide nice ways of changing these parameters
% to suit your tastes.
% I make things bold because they show up better in print.

% The next points are just shown as an example.
% You will need to calculate your own TPR and FPR, of course.
% Using more points than 10 will give you a smoother graph, too.
%truePosRate = [.5 .6 .7 .8 .85 .9 .92 .96 .98 .99];
%falsePosRate = [0.01 0.03 0.05 0.07 0.1 0.16 0.23 0.30 0.45 0.6];

% Create a new figure. You can also number it: figure(1)
figure;
% Hold on means all subsequent plot data will be overlaid on a single plot
hold on;
% Plots using a blue line (see 'help plot' for shape and color codes
plot(falsePosRate, truePosRate, 'b-', 'LineWidth', 2);
% Overlaid with circles at the data points
plot(falsePosRate, truePosRate, 'bo', 'MarkerSize', 6, 'LineWidth', 2);

% You could repeat here with a different color/style if you made
% an enhancement and wanted to show that it outperformed the baseline.

% Title, labels, range for axes
title('ROC of the Test Set of Jaundice', 'fontSize', 18); % Really. Change
this title.
xlabel('False Positive Rate', 'fontWeight', 'bold');
ylabel('True Positive Rate', 'fontWeight', 'bold');
% TPR and FPR range from 0 to 1. You can change these if you want to zoom in
on part of the graph.
axis([0 1 0 1]);
end
```