Mammogram Classification for Breast Cancer Detection using Machine Learning and Deep Learning

Xuhui Wang

March 2021

Introduction

In this project, I mainly focus on the different steps involved in a medical image classification pipeline: data preparation, data preprocessing, image feature extraction and image classification. The data extraction and classification phases will be done using two distinct approaches: traditional machine learning and deep learning. For classification using machine learning, I mainly use support vector machine using the feature extracted based on Histogram of Oriented Gradients. For classification using deep learning, I mainly use transfer learning, ResNet-50 network. For these two different methods, I achieved the accuracy up to 63% and 80% with very limited size of dataset.

Dataset

(Note that, in this report, the wordings that describe the questions, algorithms and dataset are all from the course material by Dr. Albu and I will give the appropriate citation at the end of each corresponding paragraph. On the other hand, all figures, tables, methods implementation, statistics, analysis, explanations and conclusions are all from my personal work.) The dataset used in this assignment is the **Mammographic Image Analysis Society (MIAS) MiniMammographic Database**, which is composed by 322 mammograms of 1024×1024 pixels. These annotated mammograms can be used to train and evaluate a system for the autonomous detection of breast cancer. Each sample is annotated based on multiple sub-classes of three characteristics, 1) character of background tissue; 2) class of abnormality present and 3) severity of abnormality. For simplicity purposes, in this assignment we will classify each mammogram based only on their **character of the background tissue**, which can be one of the following:

1. **F**: Fatty
2. **G**: Fatty-glandular
3. **D**: Dense-glandular

To complete this project, we use MATLAB's Deep Learning Toolbox and ResNet-50 Toolbox. The use of a GPU will considerably accelerate the training processes.[1]

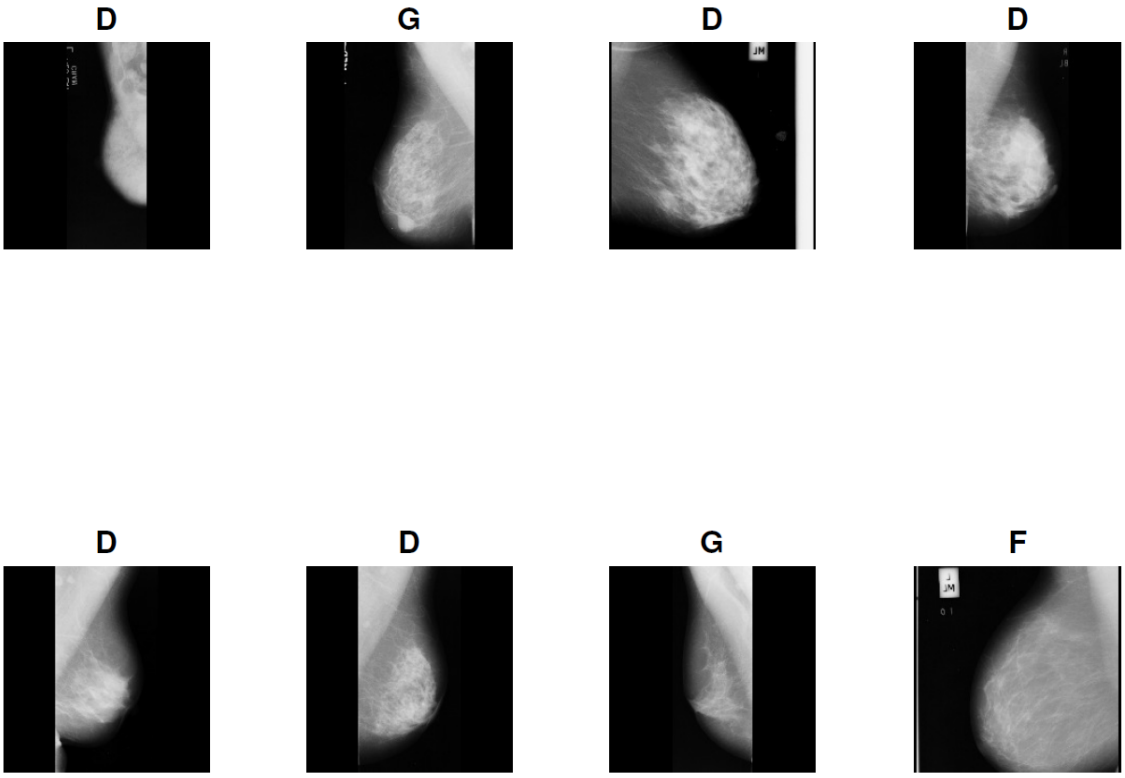


Figure 1: Sample mammograms from the MiniMammographic Database. Character of tissue background classes are highlighted on top of the samples [1].

Data Preparation and Preprocessing

The data provided in medical imaging datasets often presents different file formats (e.g., DICOM, pgm, png) and custom annotation styles. In order to illustrate that, the first step of the assignment is to read the "labels.txt" file containing information about the dataset samples. Each of the 7 columns in this text file have a specific meaning, however we are interested only on the first and second ones, which refer, respectively, to the name of the sample and the classification of its background tissue (i.e., F, G or D).[1]

Data organization

We create one folder for each of the three classes and write a script that reads the "labels.txt" file and places a .png version of each sample (.pgm) into its appropriate folder.[1]

Data preprocessing

Specific data preprocessing steps will vary depending on the characteristics of the images composing each dataset. For the mammograms of the MiniMammographic Database, we will apply a simple routine of transformations for each sample:

1. Resize the image to 224×224 pixels (allowing for the training of a specific deep learning-based network).
2. Turn the gray-scale image into a 3-channel image by duplicating its values into three channels.
3. Apply a histogram equalization process.[1]

We apply the aforementioned transformations. All the images in the dataset are preprocessed and saved in their adequate folders (i.e., F, G or D) as .png images. Figure 2 presents an original mammogram and its preprocessed version. At the end of this phase, only the preprocessed samples are kept in the folder.[1]

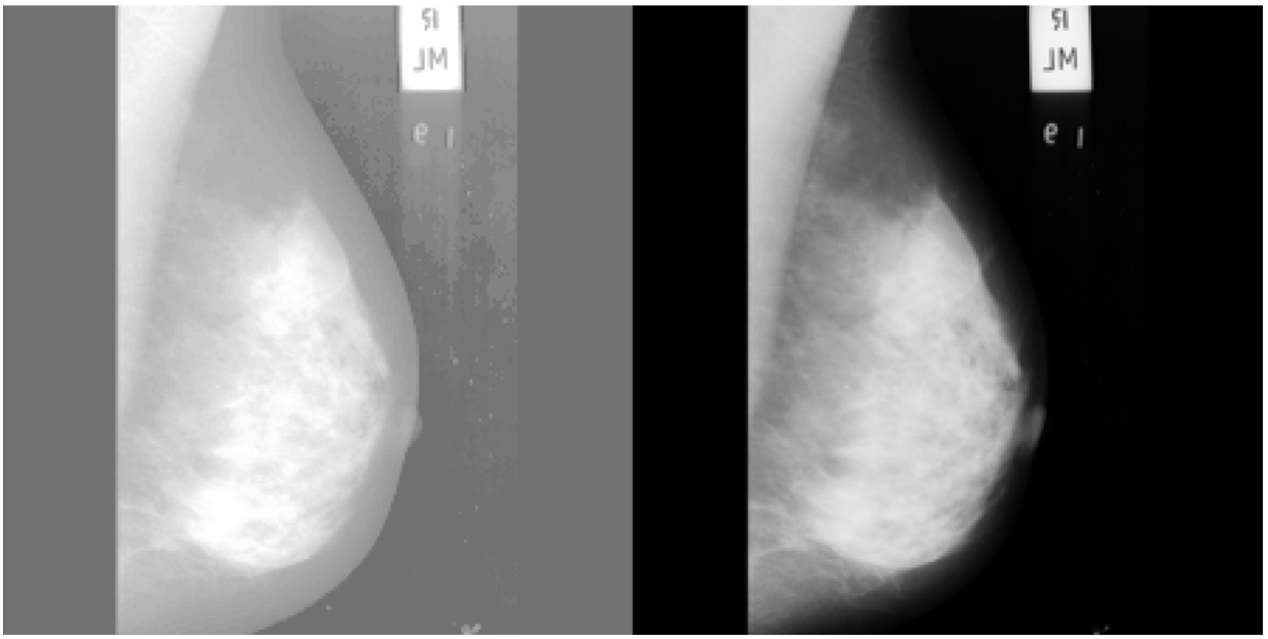


Figure 2: Left: Preprocessed mammogram. Right: Original mammogram (resized to 224 × 224 pixels for visualization purposes)[1].

Data division

In order to train a machine-learning-based classifier, one must divide the data samples into training and testing sets. The samples from the training set are only used to train the classifier, while those from the test set are only considered when evaluating it.[1]

We load the data into an imageDatastore element using MATLAB's imageDatastore function with 'foldernames' as 'LabelSource'. That will facilitate the eventual shuffling, training and evaluation processes. We report the number of samples in each class of the imageDatastore using MATLAB's countEachLabel function.

Label	Count
D	112
F	106
G	104

Number of samples in each class

Then we use MATLAB's splitEachLabel function to randomly divide the data into two sets: imdsTrain and imdsTest. The training set must contain 70% of the data, while the testing set must hold the remaining 30%.

Classification using Machine Learning

In traditional pattern recognition, hand-crafted features of the data are first extracted and then used as input for some classification method (e.g., decision trees, random forests, nearest neighbors, linear classifiers, support vector machines).[1]

We are going to extract the image's features using the popular Histogram of Oriented Gradients (HoG) method, which determine feature descriptors based on the number of occurrences of gradient orientations in different

positions of an image. These features are going to be used to train and evaluate a Support Vector Machine 6 (SVM) classifier.[1]

Histogram of Oriented Gradients

We create the extractHOG function to calculate the HoG features from all the images of an imageDatastore element using an specified cell size (e.g., 2 × 2, 4 × 4). We use MATLAB's extractHOGFeatures to extract the HoG features form each sample. The feature array holding the HoG features of each mammogram should be stored in an individual row of a matrix. Then we extract the HoG features from the mammograms in the training and testing imageDatastores created previously with a 4 × 4 window. Then, we report what would be the size of the feature array of each sample for 2 × 2, 4×4 and 8×8 windows.

Given that other variables, including block size = [2, 2], block overlap = block size / 2, number of bins = 9 and so on, are set to default values in Matlab, the sizes of the feature array of each sample for different windows are

Window	Size
2×2	$(224/2\text{-}1)^2 \times 4 \times 9 = 443556$
4×4	$(224/4\text{-}1)^2 \times 4 \times 9 = 108900$
8×8	$(224/8\text{-}1)^2 \times 4 \times 9 = 26244$

Size of the feature array for each window size

Classification using a Support Vector Machine

We create an SVM classifier element using MATLAB's fitcecoc function. The inputs of this function should be the training HoG features (extracted previously) and their equivalent labels (i.e., F, G or D).

Then we use MATLAB's predict function to test the classifier by using it to predict the classes of the test HoG features (extracted previously). The inputs of this function should be the trained SVM and the testing HoG features.

We use MATLAB's plotconfusion function to display the results of the SVM-based classifier's predictions. The test labels and classifier predictions are the inputs of this function.

Figure 3 shows that the overall classification accuracy is 62.9%. 61 images are correctly recognized.

For D class, the classification accuracy is 47.1%. 16 images with D label are correctly recognized, while 5 images with D label are incorrectly recognized as F class and 13 images with D label are incorrectly recognized as G class.

For F class, the classification accuracy is 78.1%. 25 images with F label are correctly recognized, while 1 images with F label are incorrectly recognized as D class and 6 images with F label are incorrectly recognized as G class.

For G class, the classification accuracy is 64.5%. 20 images with G label are correctly recognized, while 2 images with G label are incorrectly recognized as D class and 9 images with G label are incorrectly recognized as F class.

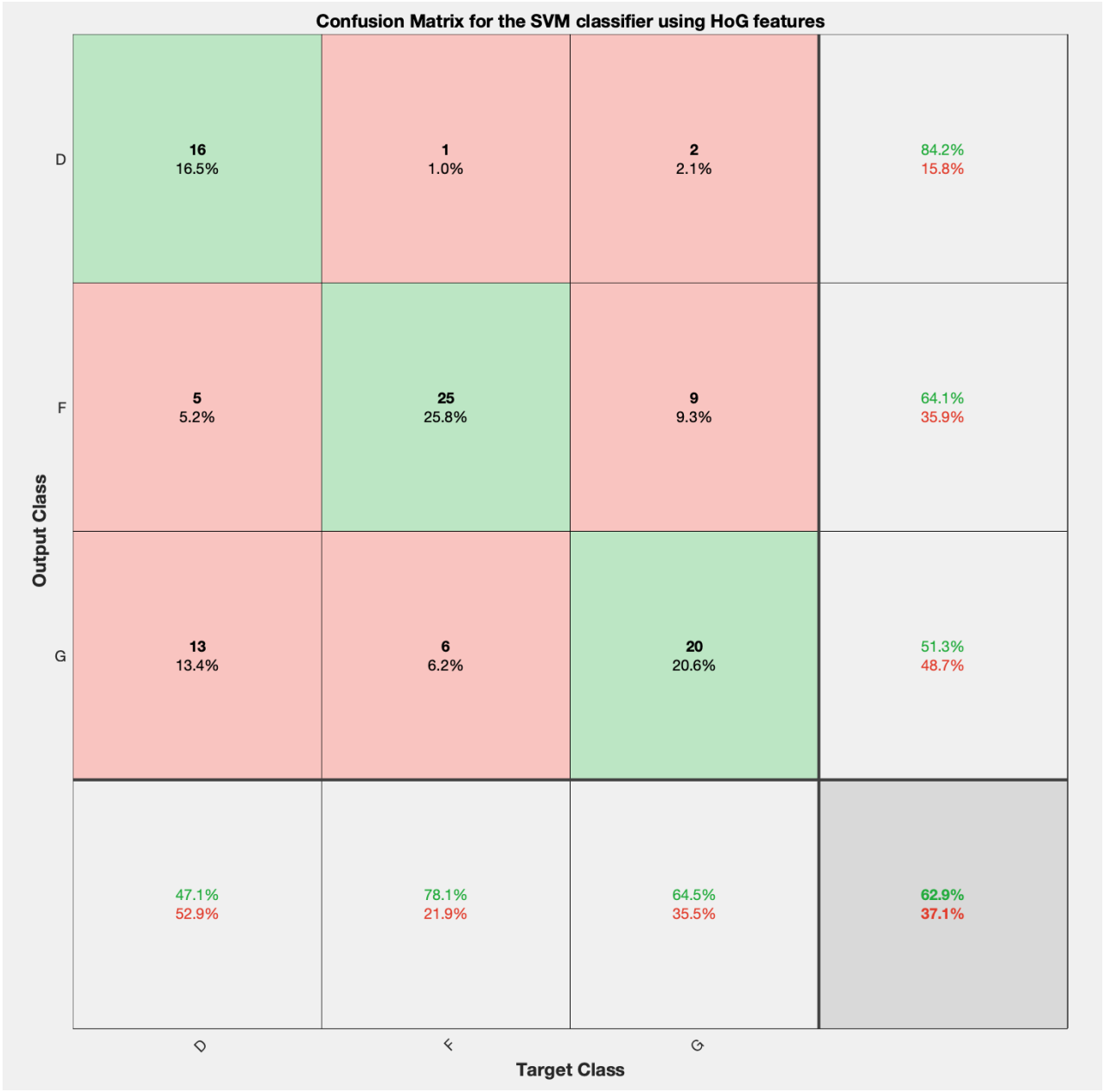


Figure 3: Confusion Matrix for the SVM classifier using HoG features

Classification using Deep Learning

Deep learning-based methods are data-driven: they rely on a large number of samples from which patterns from the data are extracted and eventually recognized. Among the various sub-fields of deep learning, the image classification one uses Convolutional Neural Netowrks(CNNs) to extract features from images and classify them into classes using Neural Networks (NNs). We are going to explore the use of a popular CNN architecture - ResNet.[1]

The main advantage of deep learning-based systems with respect to pattern recognition methods is the fact that the convolutional kernels used in CNNs can capture complex and comprehensive characteristics of the data in different image compositions, offering a generic and efficient approach for feature extraction and classification.[1]

Validation Set and Augmentation

In order to analyze the progression in performance while the classifier is trained, a common practice in deep learning is to create a validation set. During set intervals in the training, this set is going to be evaluated by the in- training network. However, the loss it generates will not be used to update the parameters of the network.[1]

Another typical data preparation step is to do data augmentation on the training set. This generates samples that are slightly different from those on the training set, but still valid to its labels (e.g., cars that are different from those in the original images, but should still be classified as cars). This process ensures that the network can generalize better to previously unseen samples.[1]

We use MATLAB's splitEachLabel function to randomly divide the training data into two new sets: imdsTrain and imdsValid. The new training set must contain 70% of the old, while the validation set must hold the remaining 30%. Report the new number of training and validation samples.

In order to augment the new training data, create an imageDataAugmenter element in MATLAB that adds a random reflection along the X-axis of the mammograms (so that mammograms from both breast orientations can be correctly classified). Then, use MATLAB's augmentedImage- Datastore function to create a new ImageDatastore that uses the imageDataAugmenter element to create training samples with random X-axis reflections.

Label	Count
D	55
F	52
G	51

Number of samples in each class for the training set

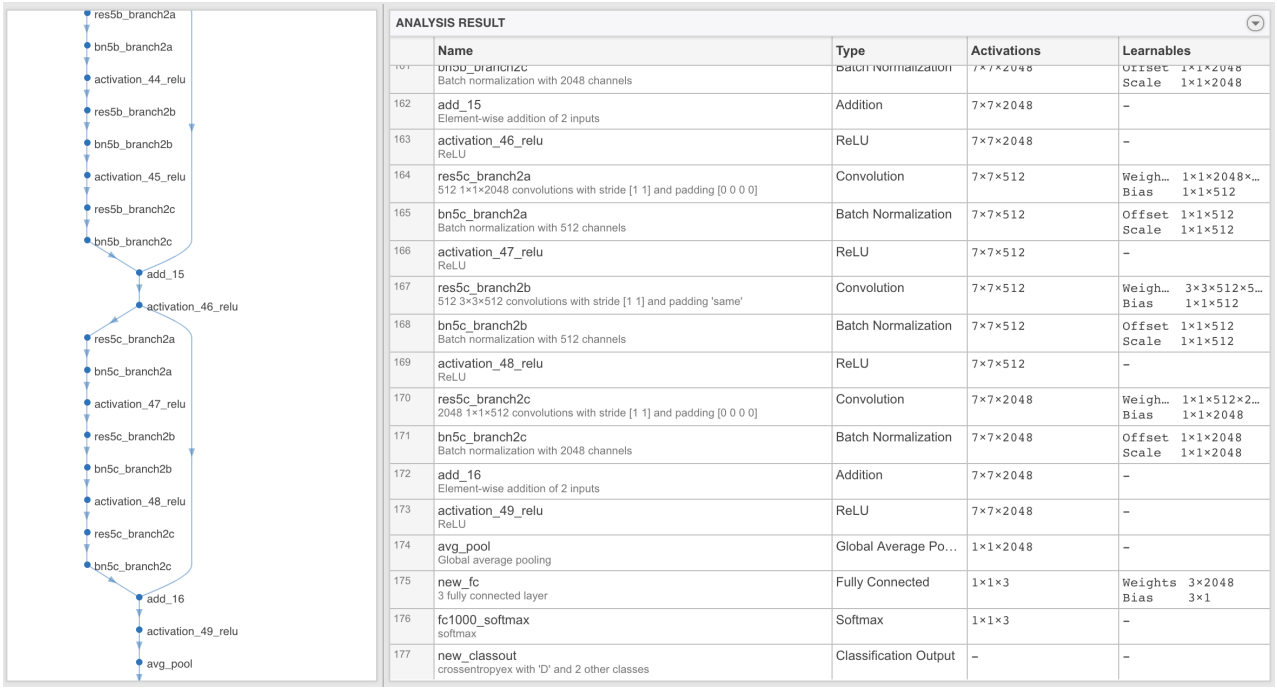
Label	Count
D	23
F	22
G	22

Number of samples in each class for the validation set

Creating, Modifying and Training a ResNet-50

ResNet is a CNN originally trained to classify an image in one of a thousand classes (e.g., goldfish, tree frog, desk). Since we only have three classes, the ResNet architecture has to be changed. In particular, the last two layers, 'fully-connected' and 'classification', need to adjusted to the reduced number of classes.[1]

We create a resnet50 network. Then, we create two new layers called 'new fc' and 'new classoutput', and substitute the last two fully-connected and classification layers from the ResNet-50 by them.



ANALYSIS RESULT

	Name	Type	Activations	Learnables
	bn5b_branch2c Batch normalization with 2048 channels	Batch Normalization	7×7×2048	Offset 1×1×2048 Scale 1×1×2048
162	add_15 Element-wise addition of 2 inputs	Addition	7×7×2048	–
163	activation_46_relu ReLU	ReLU	7×7×2048	–
164	res5c_branch2a 512 1×1×2048 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	7×7×512	Weigh… 1×1×2048×… Bias 1×1×512
165	bn5c_branch2a Batch normalization with 512 channels	Batch Normalization	7×7×512	Offset 1×1×512 Scale 1×1×512
166	activation_47_relu ReLU	ReLU	7×7×512	–
167	res5c_branch2b 512 3×3×512 convolutions with stride [1 1] and padding 'same'	Convolution	7×7×512	Weigh… 3×3×512×5… Bias 1×1×512
168	bn5c_branch2b Batch normalization with 512 channels	Batch Normalization	7×7×512	Offset 1×1×512 Scale 1×1×512
169	activation_48_relu ReLU	ReLU	7×7×512	–
170	res5c_branch2c 2048 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	7×7×2048	Weigh… 1×1×512×2… Bias 1×1×2048
171	bn5c_branch2c Batch normalization with 2048 channels	Batch Normalization	7×7×2048	Offset 1×1×2048 Scale 1×1×2048
172	add_16 Element-wise addition of 2 inputs	Addition	7×7×2048	–
173	activation_49_relu ReLU	ReLU	7×7×2048	–
174	avg_pool Global average pooling	Global Average Po…	1×1×2048	–
175	new_fc 3 fully connected layer	Fully Connected	1×1×3	Weights 3×2048 Bias 3×1
176	fc1000_softmax softmax	Softmax	1×1×3	–
177	new_classout crossentropyex with 'D' and 2 other classes	Classification Output	–	–

Figure 4: The analysis result of the CNN model

Figure 4 shows that my network have the last three layers exactly same as those illustrated in the example.

We use MATLAB's trainNetwork function to train the network, while providing the augmented imageDatastore, modified Resnet-50 network and training options as inputs. We report the training curves (accuracy and loss) provided by the trainNetwork function. Note that the accuracy of the training set is higher, indicating that the network is overfitting (i.e., "learning" how to identify the training set very well, while becoming less general).

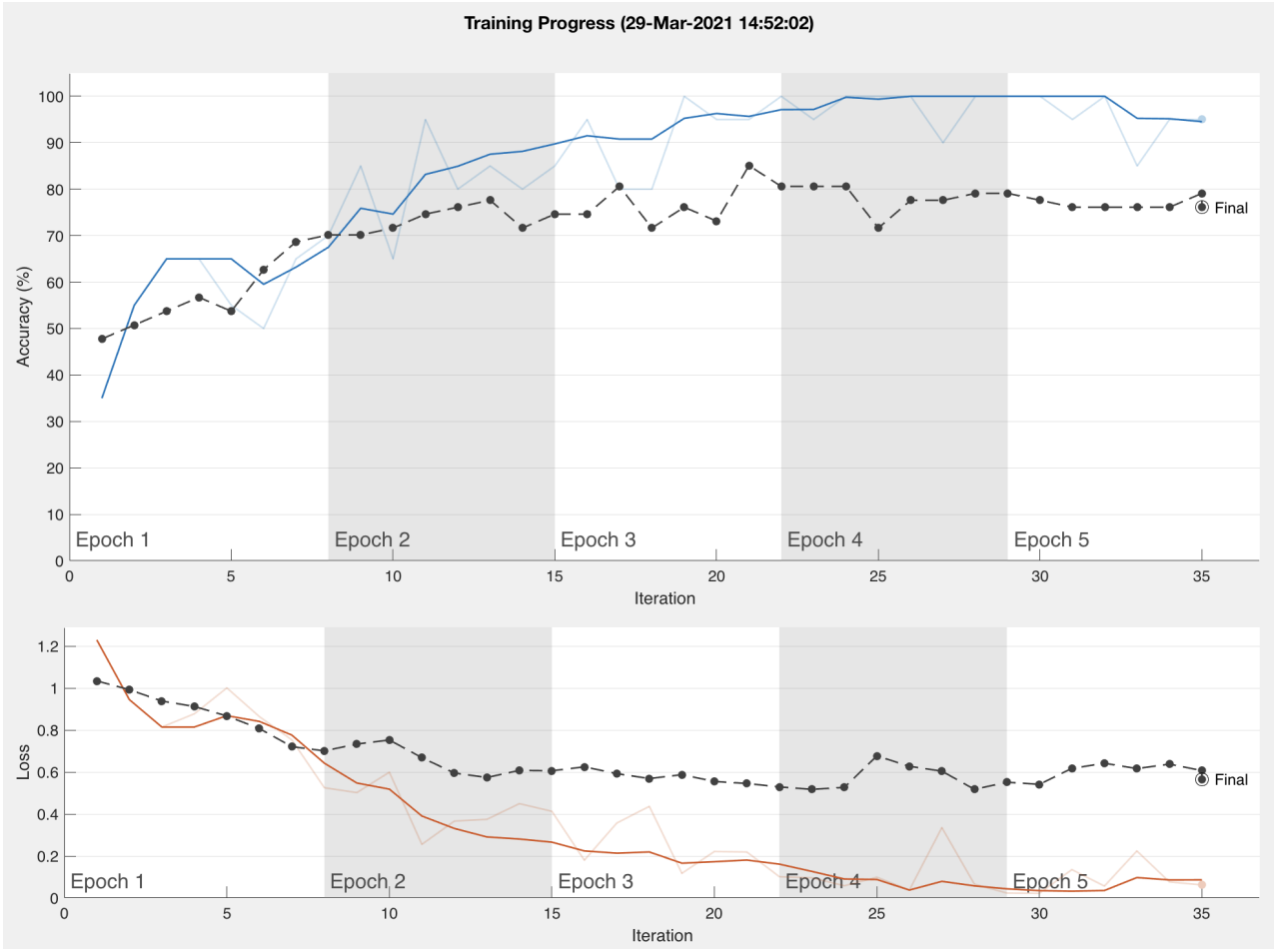


Figure 5: The training, loss and validation curves

Figure 5 shows the accuracy and loss curves for the training process. We could observe that the accuracy of the training set is getting higher, which indicates that overfitting does exist. But given the accuracy of the validation and the loss value, this performance is acceptable. Further details about the performance on the testset would be discussed in the next sections.

We use MATLAB's classify and plotconfusion functions to evaluate and present the results of the deep learning-based classifier on the test set. Then we report this confusion matrix. According to the statistics, the CNN model result is better than the SVM-based one.

Figure 6 shows that the overall classification accuracy is 76.3%. 74 images are correctly recognized.

For D class, the classification accuracy is 82.4%. 28 images with D label are correctly recognized, while no image with D label are incorrectly recognized as F class and 6 images with D label are incorrectly recognized as G class.

For F class, the classification accuracy is 90.6%. 29 images with F label are correctly recognized, while 1 images with F label are incorrectly recognized as D class and 2 images with F label are incorrectly recognized as G class.

For G class, the classification accuracy is 54.8%. 17 images with G label are correctly recognized, while 9 images with G label are incorrectly recognized as D class and 5 images with G label are incorrectly recognized as F class.

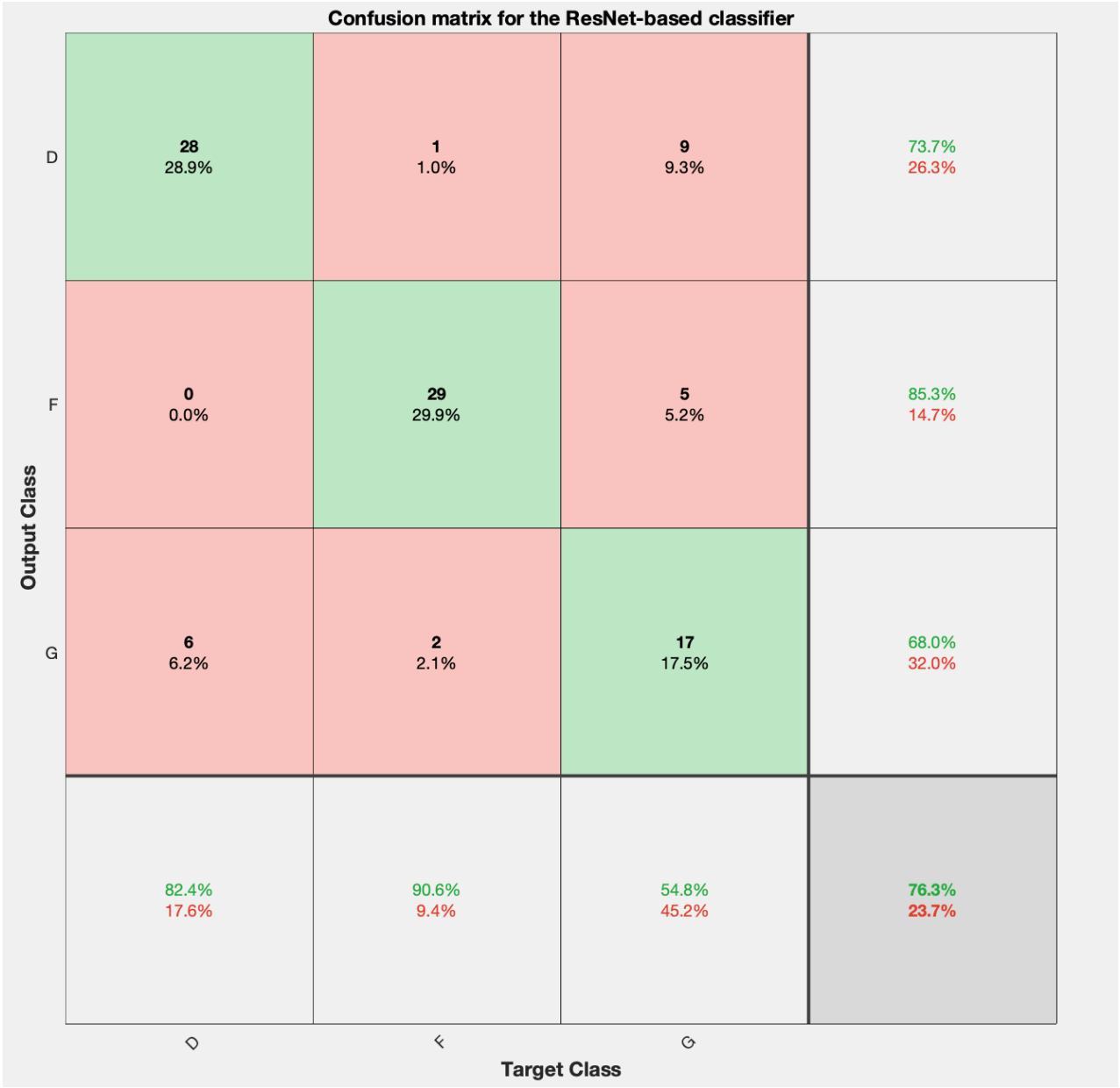


Figure 6: Confusion matrix for the ResNet-based classifier

Tuning and Better Results

Note that the parent folder is "*NNImages*". For the data preprocessing, images are still resized to [224, 224]. Given the fact that, by countless tests, several data normalization methods, such as z-score, do not give better performance and the deep learning model we use already uses batch normalization, We choose to use the raw data normalized to [0, 1], without histogram equalization, to feed into the model. The hyperparameters I use is as follows:

- BatchSize = 16
- MaxEpoch = 4
- LearningRate = $1e^{-4}$ and so on.

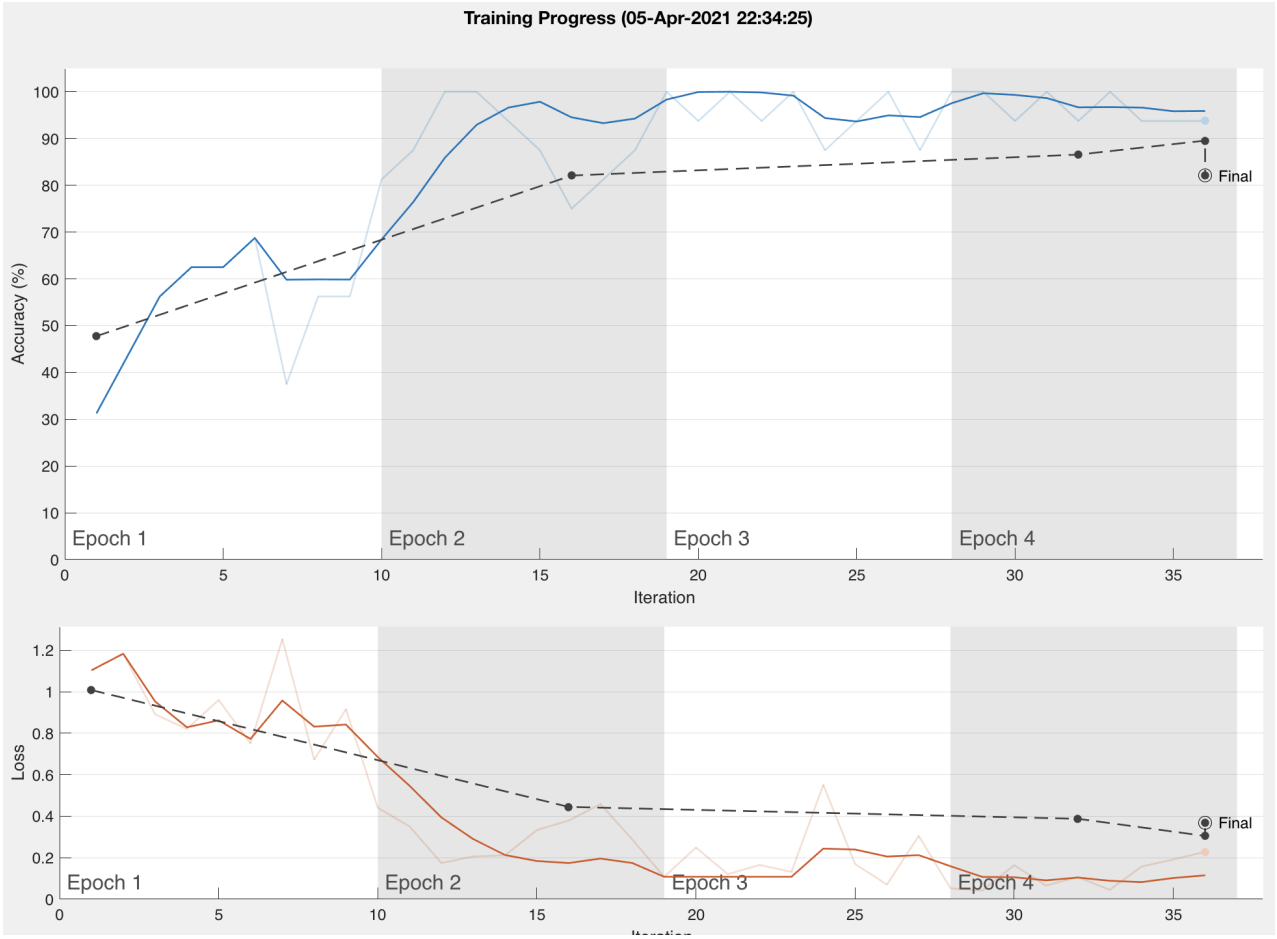


Figure 7: The training, loss and validation curves

Figure 7 shows the accuracy and loss curves for the training process. We could observe that the accuracy of the training set is getting higher, which indicates that overfitting does exist a little bit. But given our limited training set, this performance is highly acceptable.

Figure 8 shows that the overall classification accuracy is 81.4%. 79 images are correctly recognized.

For D class, the classification accuracy is 67.6%. 23 images with D label are correctly recognized, while 3 images with D label are incorrectly recognized as F class and 8 images with D label are incorrectly recognized as G class.

For F class, the classification accuracy is 96.9%. 31 images with F label are correctly recognized, while no image with F label are incorrectly recognized as D class and 1 images with F label are incorrectly recognized as G class.

For G class, the classification accuracy is 80.6%. 25 images with G label are correctly recognized, while 6 images with G label are incorrectly recognized as D class and no image with G label are incorrectly recognized as F class.

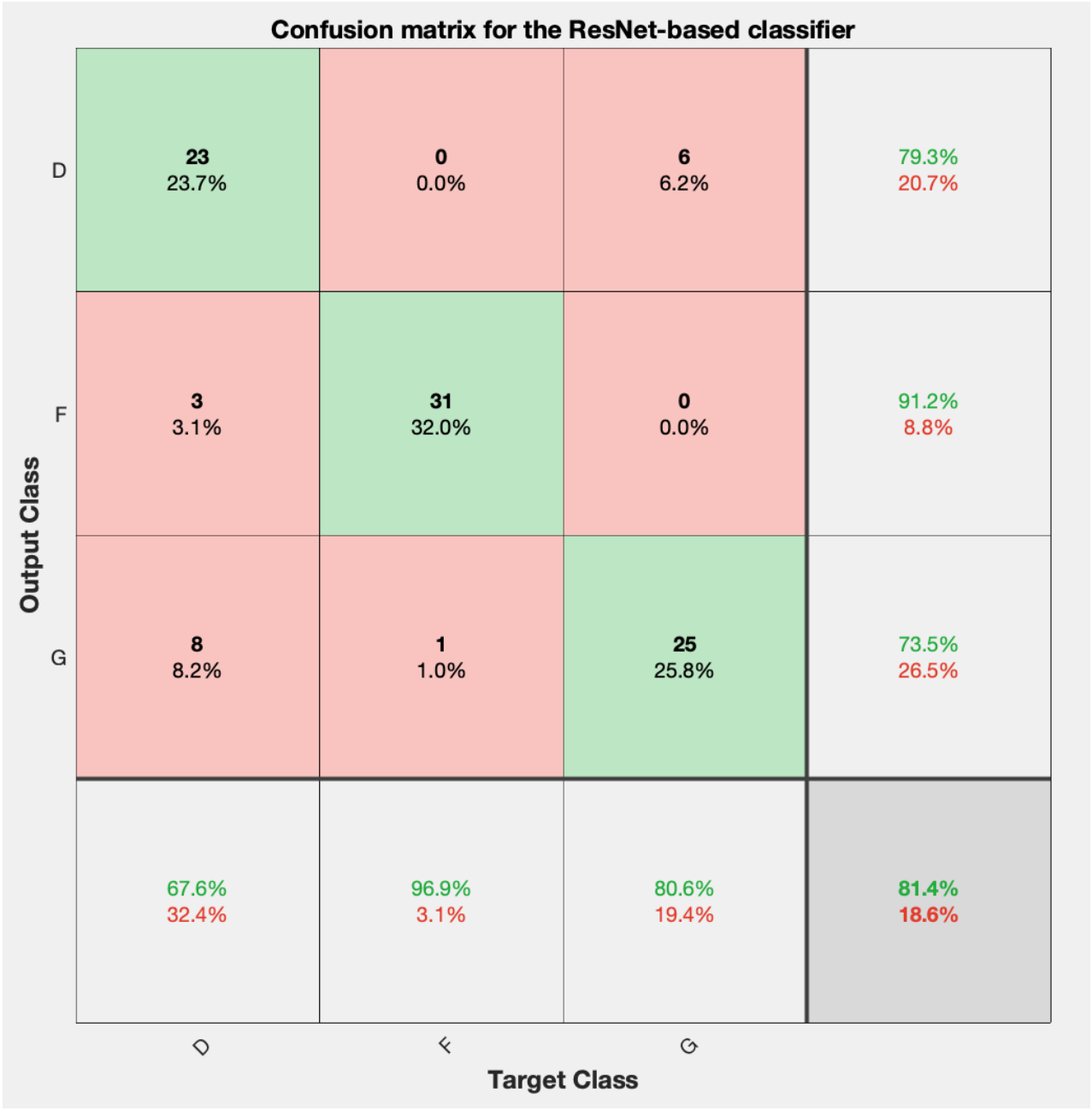


Figure 8: Confusion matrix for the ResNet-based classifier

Conclusion

In this project, I mainly focused on the different steps involved in a medical image classification pipeline: data preparation, data preprocessing, image feature extraction and image classification. The data extraction and classification phases were done using two distinct approaches: traditional machine learning and deep learning. For classification using machine learning, I mainly used support vector machine using the feature extracted based on Histogram of Oriented Gradients. For classification using deep learning, I mainly used transfer learning, ResNet-50 network. For these two different methods, I achieved the accuracy up to 63% and 80% with very limited size of dataset.

References

[1] A. Albu, *Ece435: Medical image processing.*