Assignment 1 Report

CPS 584 - Advanced Intelligent Systems and Deep Learning

University of Dayton

February 22, 2023

Authors:
Connar Hite
Ranjita Piratla
Sean Saud
Monika Somu
T.Shiva Harshith Varma

1 Introduction

The purpose of this assignment is to become familiar with the K-Nearest Neighbors (KNN) and multilayer perceptron Neural Networks (NN) methods for image classification using Haar-like features. A set of images of flowers is provided which contains two classes: *Rose* and *Tulip*. For each class, 40 training images and 20 testing images are given. The images are initially gray-scaled and resized to the size of 50 x 50. Then, Haar-like features are extracted from each training image using its integral image. Next, the KNN classification method is applied using four different values of K: 1, 3, 5, and 7. Then, the multilayer perceptron Neural Network is trained and tested using different numbers of hidden layers and hidden layer nodes. For each system, the input is thirty Haar-like features of each image and the output is a predicted class label: *Rose* or *Tulip*. The accuracy rate for each class, as well as the overall accuracy, are reported for each instance of both classification methods. Next, an additional 30 training images for both classes are manually collected and this entire process is repeated.

2 Problems

2.1 Extracting Haar-like Features

For extracting the Haar-like features from the given images, we first calculated pixel values of the integral image. We implemented a function for doing this. In this function, there are three cases. Firstly, for the first pixel, the value in the integral image remains the same as in the original image.

```
int_img(1,1)=img(1,1)
```

Secondly, for the pixels in the first row and first column, the value is the sum of the current pixel value and previous pixel value of the integral image.

```
int_img(i,1)=img(i,1)+int_img(i-1,1)

int_img(1,j)=img(1,j)+int_img(1,j-1)
```

Thirdly, for the rest of the pixels, we add the pixel value of the current pixel with the integral image pixel to the top and right of the current value. Then subtract the repeated pixel area from it.

```
int_img(i+1,j+1)=img(i+1,j+1)+int_img(i,j+1)+int_img(i+1,j)-int_img(i,j)
```

We then created a matrix of 30 dimensions of estimated coordinates of white regions of the given 30 rectangles. Using this, we found the sum of white and black regions of each of the testing and training data. We then extracted the haar-like features using the given formula for the rectangle sum given an integral image.

2.2 K-Nearest Neighbors (KNN) Algorithm

We train and test two machine learning models on the Haar features of rose and tulip images in this project. The K-nearest neighbor (KNN) model is one of them. The KNN model is trained and tested in the first section of the code. The Euclidean distance between each testing and training feature is calculated and stored in a 2D array called `knn` during the training phase.

```
knn(i, j) = euclidean(testing_haar(i, :), training_haar(j, :));
```

The code then uses the `K` array to set the number of nearest neighbors for the KNN model to 1, 3, 5, and 7. The code loops through each testing image and uses the `mink` function to retrieve the nearest K neighbors in the training set for each number of nearest neighbors.

```
[~, idx] = mink(knn(j, :), i);
```

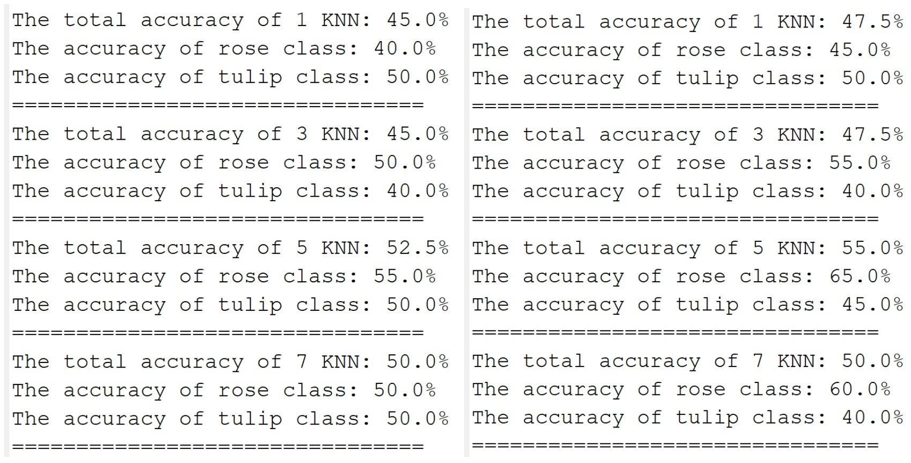
Then it retrieves the labels of these neighbors and chooses one of them. Then it retrieves the neighbor's labels and chooses the label that appears the most frequently as the predicted label for the testing image. All testing image's predicted labels are saved in a 2D array called `knn_output`.

```
knn_output(j, temp) = mode(results);
```

Finally, for each number of nearest neighbors, the code computes the accuracy of the KNN model, and the results are stored in the `accuracy_knn` array.

```
accuracy_knn(i, j) = testing_amount * length(categories) -
        sum(abs(knn_output(:, j) - testing_labels))
```

The percentage of correctly classified images is calculated by comparing the predicted labels with the true labels. The results of this section are shown in Figure 1 below.



```
The total accuracy of 1 KNN: 45.0%
The accuracy of rose class: 40.0%
The accuracy of tulip class: 50.0%
================================
The total accuracy of 3 KNN: 45.0%
The accuracy of rose class: 50.0%
The accuracy of tulip class: 40.0%
================================
The total accuracy of 5 KNN: 52.5%
The accuracy of rose class: 55.0%
The accuracy of tulip class: 50.0%
================================
The total accuracy of 7 KNN: 50.0%
The accuracy of rose class: 50.0%
The accuracy of tulip class: 50.0%
================================
```

```
The total accuracy of 1 KNN: 47.5%
The accuracy of rose class: 45.0%
The accuracy of tulip class: 50.0%
================================
The total accuracy of 3 KNN: 47.5%
The accuracy of rose class: 55.0%
The accuracy of tulip class: 40.0%
================================
The total accuracy of 5 KNN: 55.0%
The accuracy of rose class: 65.0%
The accuracy of tulip class: 45.0%
================================
The total accuracy of 7 KNN: 50.0%
The accuracy of rose class: 60.0%
The accuracy of tulip class: 40.0%
================================
```

Figure 1: KNN Results for 40 Images (Left) and 70 Images (Right)

2.3 Neural Networks (NN) Algorithm

The second machine learning model created is that of a Neural Network (NN). Before inputting the calculated Haar-like features into the training and testing functions it needs to be normalized. The NN function used does do this automatically; however, it does not account for the fact that the greatest value in the training set could be lower than that of the testing set. Normalization was done by finding the lowest value in both sets, adding said value to the rectangle sums, and then dividing everything by the highest value in both sets. When complete, all values will be between 0 and 1. For the training phase, the ‘feedforward()’ and ‘train()’ functions were utilized. These functions were given the training labels and the training Haar-like features. As for the testing phase, the resulting network, ‘net’, was given the testing Haar-like features. The result being the model's predictions for the classes, which were then thresholded. These results were then compared with the actual tag for each image to get the accuracy of the system. When defining the structure of the neural network, many separate tests were performed by changing the number of nodes and number of layers in the Neural Network. In addition to changing the layers, each test was performed 20 times. The functions used to create the Neural Network have a degree of randomness each time they run. Performing multiple tests allows for a better visualization of how accurate the system is.

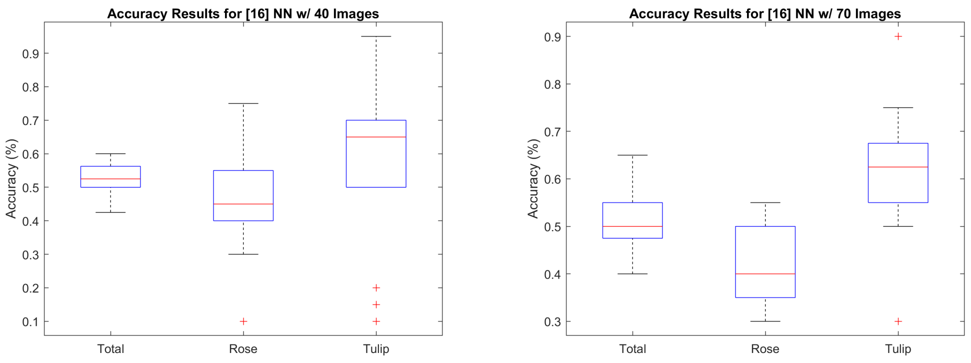


Figure 2: [16] NN Results for 40 Images (Left) and 70 Images (Right)

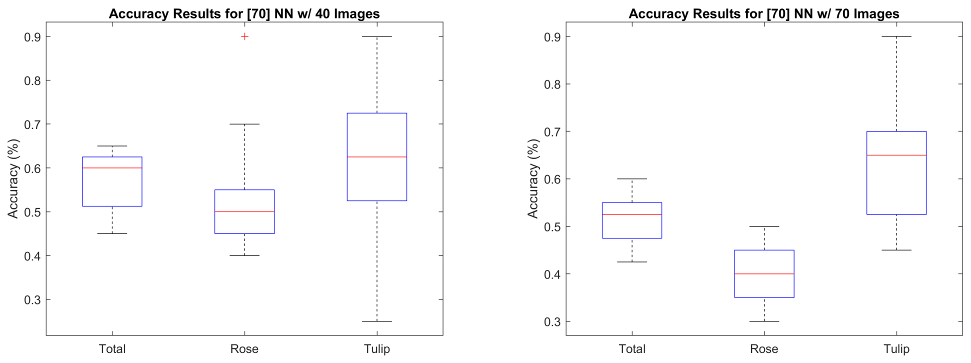


Figure 3: [70] NN Results for 40 Images (Left) and 70 Images (Right)

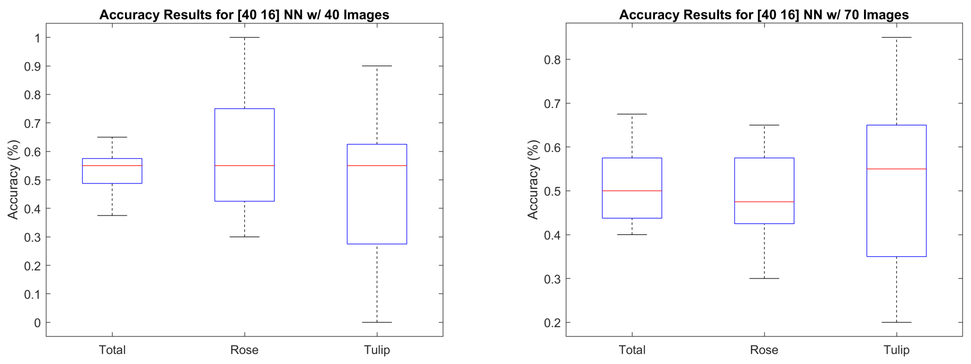


Figure 4: [40 16] NN Results for 40 Images (Left) and 70 Images (Right)

Table 1: Average Accuracy

	[16] w/ 40 Images	[16] w/ 70 Images	[70] w/ 40 Images	[70] w/ 70 Images	[40 16] w/ 40 Images	[40 16] w/ 70 Images
Total Accuracy	52.5%	51.125%	57.25%	51.875%	53.125%	52.5%
Rose Accuracy	47.7%	41.75%	53.5%	41.25%	60.25%	48.75
Tulip Accuracy	57.25%	60.5%	61%	62.5%	46%	52.25%

Figure 2, Figure 3, and Figure 4 show the results of three of these tests in the form of a box-and-whisker plot. Table 1 also provides the average accuracy values for these examples.

3 Issues Encountered & Solutions

- Calculating the integral image values in the third case mentioned above.
  We did not know we have to subtract the repeated pixel values once to get the right values. A deep conceptual dive and a pen and paper interaction helped us achieve the expected result.
- Getting the correct values for the rectangle sums was another issue. When comparing results, there were slight variations between members. To solve this issue, we went step by step until we found the point where the difference had occurred. The causes included how the Haar-like features were calculated and an issue when formatting the images (resize and gray scale).
- Another issue was that the Neural Network was not providing great results. One thing that helped improve the accuracy involved normalizing the inputs. This was not done at first because the function used does this automatically. However, it was found that performing the normalization beforehand slightly improved the accuracy.

4 Conclusion

Through the use of integral images and Haar-like features, we were able to create K-Nearest Neighbor and Neural Network systems to classify images of flowers as either a rose or tulip in this project. Using the Haar-like features, which measured 30 for each image, drastically reduced the inputs for the machine learning algorithms. For both systems each test was performed twice. Once with a training set consisting of 40 images of each class, and again with a training set consisting of 70 images of each class. For the KNN section, a K value of 1, 3, 5, and 7 was tested. The results of which were not great, as shown in Figure 1. They were all around ~50%. Meaning that for 2 classes, Roses and Tulips, the system is basically flipping a coin. However, the best result was probably the KNN utilizing 70 images with a K value of 5. The Neural Networks did not perform much better, but were generally a couple percentage points better than the KNN. The overall accuracy was generally around 50% as well. However, the NN has a degree of randomness when training. Meaning there are two things to consider when evaluating the system: average accuracy and spread of results. The general trend that was found was that having an increased amount of inputs would decrease the spread of results. This makes the system more consistent. Increasing the hidden layer size provided the best result for average accuracy.

Even though the algorithms were able to achieve a decent level of accuracy, there is room for improvement. For instance, the Haar-like features used in this project were pre-defined; however, it would be interesting to investigate the use of machine learning techniques to produce Haar-like features specifically suited to the task at hand. Additionally, it's possible that additional features could be extracted from integral images to boost classification precision. Other improvements could possibly be brought on by increasing the number of training images and experimenting with more NN architectures. Overall, this project does a good job of demonstrating the use of Haar-like features for image classification tasks involving K-Nearest Neighbor and Neural Network systems.

5 Member Contributions

Every member wrote code for this assignment. Before writing the paper/powerpoint, we came together to compare results and make any necessary changes. The submitted code is a combination of the code written by Connar and Sean.

Connar Hite: Wrote the NN section and edited paper/powerpoint
Ranjita Piratla: Wrote the KNN section and conclusion
Sean Saud: Wrote the introduction
Monika Somu: Wrote the Haar-like feature section
T.Shiva Harshith Varma: Wrote the issues during implementation