**Project 4 – ECE 5363**

**Yildirim Kocoglu**

**Aaron Carman**

**Project Goal**

Before any technical work is started for this project, the dataset is examined manually to get a sense of the problem at hand. The given data is 11,000 images with size 101x101 divided equally between classes (5500 images are labeled as worms and 5500 images are labeled as no worms). The task is to use SVM to classify these images into groups that do and do not contain worms. The labels of the images are “1” for worms and “0” for no worms. The main goal of the project is to train and deploy and SVM classifier that has the highest possible prediction accuracy.

**Visual Verification of Images**

To begin the technical work in this project, first, the images were visually verified. It was noticed that some of the images in the “0” (no worms) folder in fact contained partial worms (half of a worm, etc.) and some of the other images in the “1” (worms) folder contained worms right on top of the defect or near the edge of the defect (hard to notice with an average human eye) but, none of the images were removed or altered due to the fact that the proportion of these images compared to the total number of images were very low. We believe that it may have some minor impact on our final accuracy of the model however, it shouldn’t be significant.

**Image Pre-processing**

In the next step, the given images are pre-processed for our convenience. This pre-processing occurs in the “**read\_and\_save\_images.m**” MATLAB script. This script creates a folder called “**TrainingData**” inside the folder where the “**read\_and\_save\_images.m**” script is located and saves 2 MAT files inside the “**TrainingData**” folder called “**Training\_noworm.mat**” (containing “no worm” images) and “**Training\_worm.mat**” (containing “worm” images) by reading the images in each folder named “0” (no worm images) and “1” (worm images).

Each image is later loaded into the MATLAB workspace of the “**Proj4\_Kocoglu\_Carman.m**” MATLAB script by loading these saved MAT files. **This was specifically done to save time during our hyperparameter selection process for SVM.**

The “**read\_and\_save\_images.m**” script first **binarizes** each image using the **imbinarize()** function, which converts the grayscale images into either 1’s or 0’s if the pixel’s value is above or below a certain threshold. The script later **flattens each image** into a row vector which creates a matrix of all the images at the end with **dimensions (NxL)** where N is the number of images (5500 for each folder/class) and L is the number of features (101x101 = 10201).

To put it simply, each image is reshaped from a 101x101 matrix to a 1x10201 vector and stored as part of a MAT file for later use by using this script. **The size of the images were kept the same (101x101) and not reduced.**

**SVM Training and Evaluation**

To begin training the classifier, the images and labels are loaded from their respective MAT files that were generated previously. Once the images are loaded, they are shuffled and divided into training, validation, and testing datasets.

Below are the percentage of the images used for training, validation, and testing in Table-1:

**Table-1: Percentage of images used in training, validation, and testing sets**

|  |  |
| --- | --- |
| Training images | 60% |
| Validation images | 20% |
| Testing images | 20% |

MATLAB’s **fitcsvm()** function is used to train a **RBF Kernel SVM** classifier.

Below are the hyperparameters used for training and re-training the **RBF Kernel SVM** in Table-2 and Table-3 respectively:

**Table-2: SVM Parameters for Training (60% of the data)**

|  |  |
| --- | --- |
| KernelScale (sigma) | ‘auto’ (67.73) |
| BoxConstraint (C) | 10 |
| Solver | SMO |
| Standardization | True |

**Table-3: SVM Parameters for Re-training (100% of the data)**

|  |  |
| --- | --- |
| KernelScale (sigma) | 67.73 |
| BoxConstraint (C) | 10 |
| Solver | SMO |
| Standardization | True |

Below are the training and testing time using 60% of the data (training) and 20% of the data (testing) in Table-4:

**Table-4: Training (60% of the data) and testing time (20% of the data) in seconds**

|  |  |
| --- | --- |
| Training time | 82.49 seconds |
| Testing time (test data) | 26.17 seconds |

Below are the Training, validation, and testing accuracy for the split data (used during hyperparameter selection) in Table-5:

**Table-5: Accuracy for training, validation, and test data sets (60% - 20% - 20%)**

|  |  |
| --- | --- |
| Training Accuracy | 100 % |
| Validation Accuracy | 92.18 % |
| Testing Accuracy | 90.91 % |

Below are the training and testing time using 100% of the data (training) and 20% of the data (testing) in Table-6:

**Table-6: Re-training time using 100% of the data and testing time using 20% of the data**

|  |  |
| --- | --- |
| Re-training time | 201.30 seconds |
| Testing time (test data) with the re-trained model | 38.72 seconds |

Below are the training and testing accuracy using 100% and 20% of the data respectively after re-training with all of the data (100% of the data) in Table-7:

**Table-7: Training Accuracy (100% of the data) and Testing Accuracy (20% of the data) after Re-training with all of the data**

|  |  |
| --- | --- |
| Training Accuracy (After re-training) | 99.98 % |
| Testing Accuracy (After re-training) | 99.91 % |

As one would expect, the accuracy for the training set is quite high since the classifier is designed around this set, while the accuracy is lower for the validation and testing sets since this represents new, unseen data. Regardless, the classifier produces an accuracy >90%, which was considered acceptable by the project group.

A **confusion matrix** is generated for the test dataset using **RBF Kernel SVM** and is shown in Figure 1.

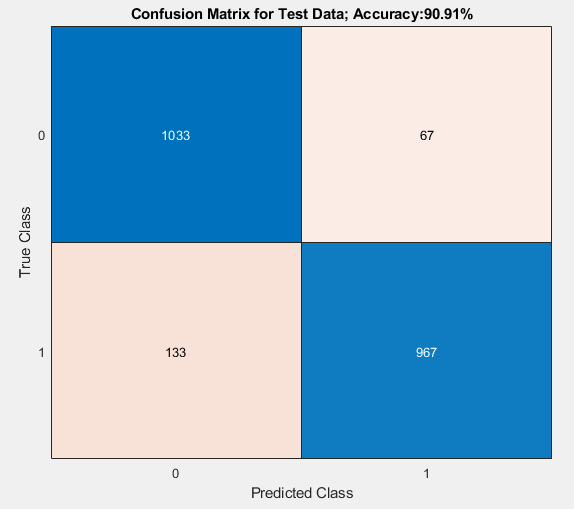


Figure 1: RBF Kernel SVM Confusion Matrix for Test Data

**Additional Notes Given to the User**

* If, the user wishes to re-create the process, the MATLAB scripts should be run in the order below:

1. **read\_and\_save\_images.m**
2. **Proj4\_Kocoglu\_Carman.m**
3. **Model.m**

* If the user does not wish to re-create the process and just use the **saved SVM model** (**SVM.mat**) to predict their own images, then just running “**Model.m**” MATLAB script is enough.
* “**Model.m**”
  + Asks for **user input of the location of the images** to be predicted and creates a “**Predictions\_list**” **cell array** containing the **names of the images** and the **predicted labels** (0 = no worm, 1 = worm). This cell array can be viewed by the user in the MATLAB workspace after executing the code.
  + It also prints out the number of worms and no worms in the command window.
  + **“Model.m**” **does not create** the “**TrainingData**” folder or the **MAT files** like the “**read\_and\_save\_images.m**” script (It saves them into the workspace directly).
* The main difference between “**Model.m”** and the other scripts is that this script has the same script “**read\_and\_save\_images.m**” **built-in as a function** but, it is made **flexible** enough to recognize the images, **regardless of the names of the images** whereas the other script (**read\_and\_save\_images.m**) is specifically built to read the images according to the format of the images given previously (image\_1, image\_2, etc.).
* It should be also noted that “**read\_and\_save\_images.m**” **does not ask for user input** and therefore, “**location1**” and “**location2**” variables in this script should be **manually modified by the user before running this script** in case the user wants to re-create the process.