



Ming Hsieh

Department of Electrical Engineering

Sonny Astani

Department of Civil and Environmental Engineering

A PROJECT REPORT ON

Feature Extraction, Classification and Multi-class Error Analysis of Sewer Pipeline Images

Under the Supervision of
Professor Keith Jenkins

Ming Hsieh Department of Electrical Engineering
University of Southern California

Author:

Preetham AGHALAYA
Manjunatha
USC, Dept. of Civil

Author:

Shravan RAVI
USC, Dept. of Electrical

Submitted On

December 08, 2014

Table of Contents

| | |
|---|------------|
| List of Tables | ii |
| List of Figures | iii |
| Abstract | iv |
| 1 Introduction | 1 |
| 1.1 Related work | 1 |
| 1.2 Contribution | 2 |
| 1.3 Scope | 2 |
| 2 Project Formulation and Setup | 2 |
| 2.1 Support Vector Machine (SVM) | 3 |
| 2.2 Neural Networks | 4 |
| 2.3 Error Analysis | 5 |
| 3 Methodology | 6 |
| 4 Implementation | 8 |
| 4.1 Feature Space | 8 |
| 4.2 Pre-processing and Feature Extraction | 10 |
| 4.3 Training Process | 13 |
| 4.4 Testing, Validation and Model Selection | 14 |
| 5 Final Results | 14 |
| 5.1 SVM | 15 |
| 5.2 Neural network | 15 |
| 6 Interpretation | 17 |
| 7 Summary, Conclusions and Future Work | 17 |
| 8 Acknowledgments | 18 |
| 9 Bibliography | 20 |
| Appendices | 22 |
| A Main Code | 22 |
| B Sub-routines | 32 |

List of Tables

| | | |
|---|---|----|
| 1 | Classifier performance for SVM | 16 |
| 2 | Classifier performance for FFNN | 17 |

List of Figures

| | | |
|----|---|----|
| 1 | A typical classification model of SVM | 4 |
| 2 | A typical NN model (source: Wikipedia) | 4 |
| 3 | Various phases for the project | 7 |
| 4 | Various classes of images | 9 |
| 5 | An inpainted image | 11 |
| 6 | A SURF detector points | 11 |
| 7 | A scree plot of the PCA latency (Eigen values) | 12 |
| 8 | Decision boundary of a SVM classifier | 14 |
| 9 | Distribution of the Dataset | 15 |
| 10 | Observations SVM | 16 |
| 11 | Misclassification error for NN with 1-2-3 hidden layers and various neurons | 19 |

Abstract

Object classification is an active research in the field of computer vision and image processing. In this project, a medium-scale data is manually labeled and used to train classifiers. Non-linear classifiers such as Support Vector Machine (SVM) and Feed-forward neural Networks (FFNN) are used to predict the testing set labels. The results look surprising and very promising for the large-scale object recognition. Also, it was found that large training set reduces the overfitting and improves the classification accuracy. Lastly, an attempt has been made to understand the generalization bound for multiple classes.

1 Introduction

Fatigue and failure are common in civil infrastructures, mechanical and aerospace systems and other applications. It is important to perform condition assessment regularly in order to minimize serious structural failures. In Structural Health Monitoring (SHM), the focus is on vibration-based methods that have been practiced for more than four decades. However, due to the development of accurate, robust and inexpensive visual sensors, vision-based autonomous SHM techniques have gained more attention. Thus, visual inspection is the most commonly used technique to assess the condition of almost all infrastructure systems ([Jahanshahi et al., 2011](#)).

Civil infrastructures play an important role in the development of a country's economy. The infrastructure cost in the US alone is more than \$20 trillion [Iyer and Sinha \(2013\)](#). According to the American Society of Civil Engineers' (ASCE) report, there are an estimated 0.7 to 0.8 million miles of public sewer mains in the United States [ASCE \(2013\)](#). Many of these pipelines were installed after World War II and are aging down. Due to the inadequate human resources, not every section of the pipe gets the required attention. Operational and maintenance cost of these pipelines itself requires billions of dollars every year. An irregular and non-periodic trend of sewer-pipe inspection jeopardizes the structural integrity of the waste water mains, thus increasing the possibility of untreated sewage discharge into the surroundings. A novel and innovative approach to classify healthy and unhealthy water mains is required to reduce human intervention. This system can improve the decision making ability based on objectivity, rather than on a subjective human perspective. This project includes feature extraction, classification and error analysis of sewage mains.

A vast number of videos are recorded by sending an unmanned vehicle through sewer pipelines. Analyzing these videos requires experienced human inspectors and hours of laborious visual inspection. In general, visual inspection by humans are very subjective. A damaged sewer pipe can induce a threat to the ground water sources and can be responsible for environmental issues and spread of epidemics. These factors motivate the need for autonomous vision-based techniques that eliminate the subjectivity to which humans are prone with minimal human intervention.

1.1 Related work

In a Closed-Circuit-Television (CCTV)-based monitoring system, [Fieguth and Sinha \(1999\)](#) used digital image processing to detect the cracks in underground pipes by means of statistical parameters such as samples, mean, variance and cross-correlation with the crack pixels and their surrounding window. Although this method is able to extract features like cracks and, to some extent, joint openings of the buried pipe, it failed to minimize false positives. Again, using a warped tunnel vision by a tool called Sewer Scanner and Evaluation Technology (SSET), [Iseley \(1999\)](#), [Sinha and Fieguth \(2006a\)](#) and [Iyer and Sinha \(2013\)](#) exhibited a method that extracts the crack (eliminating joints, holes and laterals) from the buried pipeline images. This is done by using the statistical filters for crack detection followed by cleaning and linking operations. Similarly, [Iyer and Sinha \(2006\)](#) worked on multiple crack detection algorithms based on contrast enhancement, morphology, non-linear filtering and curvature evaluation of crack patterns. It is shown that morphological characters such as branches, thickness, and orientation of cracks can be exploited well for their autonomous recognition. Further, with the concept of Bayesian classification, mathematical morphological processes, and segmentation by thresholding, [Sinha and Fieguth \(2006b\)](#) studied the

feature extraction methods for multiple cracks and joints in the sewer pipeline.

All of the prior works in the field of civil engineering are limited in defect detection from color or grayscale images using edge detection, mathematical morphology or other matched filter related algorithms. Edge detection algorithms are capable of detecting sewer pipeline features, but at the same time it is practically impossible to exclude noisy pixels or edges. Mathematical morphology works well for uniform textural images, but when the image is random textured, choosing a valid structuring element is difficult. In practice, pipeline infrastructures are comprised of complex shapes, and textures and non-uniform intensity are spread around the crack regions. In addition, only a small-scale image dataset (<500) are analyzed in the published work.

1.2 Contribution

Currently, there hardly exists any public benchmark datasets for pipeline in general, that are labeled and consisting of respectable amount of images/videos. In this work, a special consideration is taken in order to prepare a labeled database consists of few thousands of images from recorded sewer-pipe videos. As the first task, hundreds of these videos have to be converted to images, followed by the preparation of a sample dataset. A sample dataset consists of training, validation and testing images. After obtaining the image dataset, a few pre-processing algorithms such as image in-painting (text removal), super-resolution and denoising are to be performed. Image features are extracted automatically by using the bags-of-features method proposed by [Csurka et al. \(2004\)](#). Once the features are obtained, a nine-class object classification problem is solved by using two non-linear classifiers. Furthermore, inspired by the VC generalization bound for binary class problem, an attempt had been made to understand and deduce a generalization bound for multi-class problems involving support vector machine (SVM) and Feed-forward Neural Network (FFNN).

1.3 Scope

[Section 2](#) introduces the problem formulation and setup. In [Section 3](#) various methods involved in this work are presented. The implementation of the proposed method is discussed in [Section 4](#). Results and interpretation are provided in [Section 5](#) and [Section 6](#) respectively. Lastly, this work is concluded by summarizing the findings and suggesting few recommends for future work in [Section 7](#).

2 Project Formulation and Setup

Proposed method deals with classifying medium-scale image datasets into multiple classes. Among the several multiclass classifiers available, few can be generalized as direct extensions of the binary clasifers.

A classification task usually involves separating data into training and testing sets. Each instance in the training set contains one “target value” (i.e. the class labels) and several “attributes” (i.e. the features or observed variables). The goal of the learning algorithm is to produce a model (based on the training data) which predicts the target values of the test data given only the test data attributes.

Due to the high complexity of the problem, we decided to use a few robust multiclass classifiers, namely Support Vector Machines (SVMs) and Neural Networks(NN)as both these classifiers are

known to perform well for large multiclass classification (Csurka et al., 2004; Jahanshahi et al., 2013). Both these models have been explained briefly below.

2.1 Support Vector Machine (SVM)

Given a training datasets and its labels $(x_i, y_i) \ i = 1, \dots, l$ where $x_i \in \mathbb{R}^N$ and $y \in -1, 1^l$, SVMs satisfy the unconstrained optimization problem:

$$\begin{aligned} \min_{w, b, \xi} \quad & \frac{1}{2} W^T W + C \sum_{i=1}^l \xi_i \\ \text{subject to} \quad & y_i (W^T \phi(x_i) + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{aligned} \tag{1}$$

Here training vectors x_i are mapped into a higher (maybe infinite) dimensional space by the function ϕ . SVM finds a linear separating hyperplane with the maximal margin in this higher dimensional space. $C > 0$ is the penalty parameter of the error term. $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ is called the kernel function. In this work radial basis function is used as the kernel:

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|), \gamma > 0 \tag{2}$$

SVM requires that each data instance is represented as a vector of real numbers. Hence, if there are categorical attributes, we first have to convert them into numeric data. Scaling before applying SVM is very important. The main advantage of scaling is to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges. Another advantage is to avoid numerical difficulties during the calculation. Because kernel values usually depend on the inner products of feature vectors, e.g. the linear kernel and the polynomial kernel, large attribute values might cause numerical problems. We thus linearly scaled each attribute to the range $[0, 1]$.

In general, the RBF kernel is a reasonable first choice. This kernel nonlinearly maps samples into a higher dimensional space so it, unlike the linear kernel, can handle the case when the relation between class labels and attributes is nonlinear. Furthermore, the linear kernel is a special case of RBF. since the linear kernel with a penalty parameter C has the same performance as the RBF kernel with some parameters (C, γ) . The SVM algorithms were implemented using LIBSVM is an integrated software for support vector classification, (C-SVC, nu-SVC) which supports multiclass classification.

A pictorial representation of the flow of the algorithm used to obtain the model and classify test images is shown in Fig. 1. This is an implementation of an already existing approach; basically training the classifier using optimal parameters obtained through some sort of validation procedure and then testing this formed model on unseen data. The gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. The C parameter trades off misclassification of training examples against simplicity of the decision surface. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly. These parameters had to be fine-tuned to get the optimal values for classification. This was done using cross-validation and will be explained in the subsequent section.

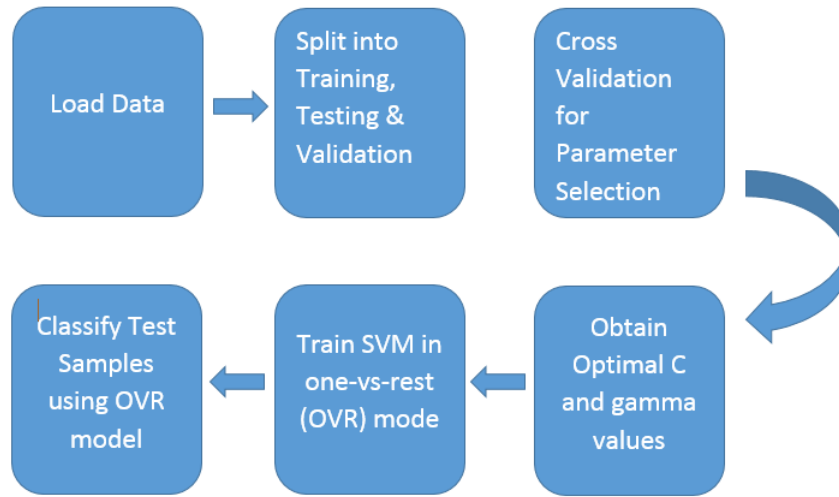


Figure 1: A typical classification model of SVM

2.2 Neural Networks

Artificial neural networks are the function approximators inspired by the biological process. If the system is dependent on the multiple parameters that are unknown, NN can be used to obtain the hidden parameters responsible for the known output. Fig. 2 shows a typical NN model. First

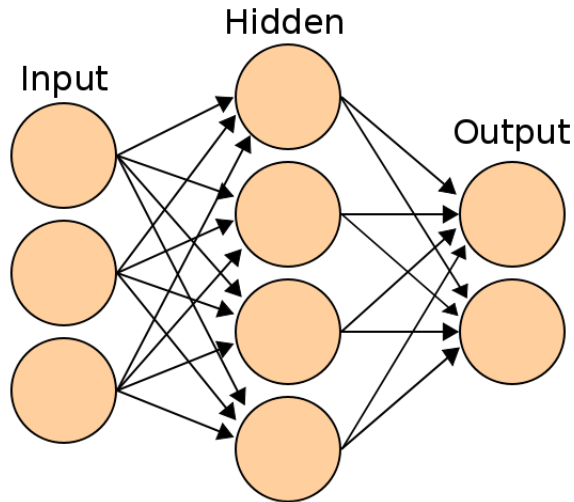


Figure 2: A typical NN model (source: Wikipedia)

layer is the input layer which depends on the size of the data. Last layer is called as the output layer, generally for a classification problem outputs are binary numbers at a given class position. Hidden layer, intermediate layers are used for the large complex dataset with high dimensional unknowns.

In this project, a linear searching of optimal number of hidden layers and the number of neurons is performed. The weights are initialized with the random entries. Similarly, biases also follows the same argument.

2.3 Error Analysis

Many theoretical and experimental studies have shown the influence the capacity of a learning machine has on its generalization ability. Small capacity learning machines might not need huge training sets to achieve the best result. High capacity machines though might need large amounts of data to reach acceptable performance. The behavior of the difference between the training and test error as a function of the training set size is characterized by a measure quantifying the machine's capacity. This measure is called the 'VC dimension' after Vladimir Vapnik and Alexey Chervonenkis. The VC dimension is an effective metric to measure the capacity of a learning machine for binary classification problems. For multiclass classification though, other concepts such as fat-shattering dimension, Natarajan dimension, Covering numbers and Rademacher complexity have been introduced. Due to constraints, the entire theory isn't explained. Rather, important results and expressions have been mentioned (Vapnik and Chervonenkis (1971); Haussler and Long (1995); Tewari and Bartlett (2014); Abu-Mostafa et al. (2012); Murphy (2012)).

In order to measure the predictive performance of a function $f : X \rightarrow Y$, we use a loss function. A loss function $l : Y \times Y \rightarrow R^+$ is a non-negative function that quantifies how bad the prediction $f(x)$ is if the true label is y . In the classification case, binary or otherwise, a natural loss function is the 0-1 loss. Given a loss function, we can define the risk of a function $f : X \rightarrow Y$ as its expected loss under the true underlying distribution: $R(f)$. Note that the risk of any function f is not directly accessible to the learner who only sees the samples. But the samples can be used to calculate the empirical risk of $f : R(f)$. Minimizing the empirical risk over a fixed class $F \subset Y^X$ of functions leads to a very important learning rule, namely empirical risk minimization (ERM). The excess risk of ERM relative to f^* can be decomposed as:

$$R(\hat{f}_n) - R(f^*) = (R(\hat{f}_n) - \hat{R}(\hat{f}_n)) + (\hat{R}(\hat{f}_n) - \hat{R}(\hat{f}_n^*)) + (\hat{R}(\hat{f}_n^*) - R(\hat{f}_n^*)) + (R(\hat{f}_n^*) - R(f^*)) \quad (3)$$

Covering Numbers : Balls of radius α placed at elements of T cover the set T entirely. The covering number (at scale α) of T is defined as the size of the smallest cover of T (at scale α). **Real Valued Functions: Fat Shattering Dimension** **Structural Risk Minimization:** Consider the classification setting with 0-1 loss (Eq. (4)).

$$\mathbb{E}[R(\hat{f}_n)] - R(f^*) \leq \min_k \{ \mathbb{E}[pen(k)] + (R(f_{\mathcal{F}(k)}^*) - R(f^*)) \} + \sqrt{\frac{\log(ce)}{2m}} \quad (4)$$

Eq. (5) Data Driven Penalties Using Rademacher Averages We can use the data dependent penalty to get the bound:

$$\mathbb{E}[R(\hat{f}_n)] - R(f^*) \leq \min_k \left\{ 2\mathcal{R}_n(\mathcal{F}^{(k)}) + (R(f_{\mathcal{F}(k)}^*) - R(f^*)) * \sqrt{\frac{\log k}{Cn}} \right\} \sqrt{\frac{\log(ce)}{2Cn}} \quad (5)$$

Interpretation for SVM

The Radial Basis Function kernel that has been used for the model is capable of mapping onto the entire feature space. The dimension of its feature space is therefore infinite. For static learning, we have the following theorem, 'If a concept class C has infinite VC dimension, then C is not learnable by any static learning algorithm'. This isn't the case for multiclass learning, where

we make use of other notions like the ‘effective VC-dimension’ and ‘covering numbers’ which do not necessarily impose such hard constraints. For SVM, the original VC dimension doesn’t work well. Vapnik noticed that the VC-dimension of a SVM is $h = \dim(F) + 1$. SVMs generally have very high VC dimensions. So, the value of the VC bound of the risk is also very high, possibly infinite. Despite having an infinite error bound, the actual performance of the SVM is pretty good. Though the SVM is distribution free, for a given training task, the distribution is already known. Hence, the SVM might not reach its full capacity while training. It was for this reason that Vapnik introduced the concept of effective VC dimension, which is a measurement of the capacity of not only the SVM, but also the sample complexity.

3 Methodology

This section provides a brief overview of the entire procedure, beginning right from hypothesis and class label selection, all the way to performance measures and error analysis.

In [Fig. 3](#) a flowchart is provided to visualize the framework. Each stage in this flowchart represents a class (or) group of operations. These classes incorporate the several intermediate steps in the procedure. An explanation of each step is provided after the flowchart.

The list of the methods used in this work are enumerated as below:

Model

1. Hypothesis set: In this work, hypothesis sets includes all the space consists of the SVM and NN models.
2. Class labels selection: Given, dataset totally consists of 16 to 17 classes. Out of which, nine classes are selected based on the importance of the classes and available computational resources.
3. Dataset: Dataset is obtained from converting 100 sewer pipeline videos to images. Images are manually labeled and outliers are filtered. Subsequently, training, validation and testing datasets are randomly selected.

Pre-processing

4. Crop: All the images are overlaid with text. This extra text and just behaves as noise in the image. Owing to the low quality of the image and the relatively huge size of the mask, inpainting algorithms too failed to provide us with a reasonable approximation. Thus, we adopted the option of cropping the images so that the part overlaid with text is cut and is no longer considered.
5. Inpaint: The patches at the bottom of the image which display the date, time and location are comparatively smaller than those at the top of the image. Thus, it was possible to apply the inpainting algorithm on them. This algorithm, which is based on a weighted minimum norm-based implementation will be discussed in the forthcoming section.
6. Feature Extraction: In order to automatically extract the features from the given images bag-of-features ([Csurka et al., 2004](#)) is used obtain a very high dimensional feature vectors.

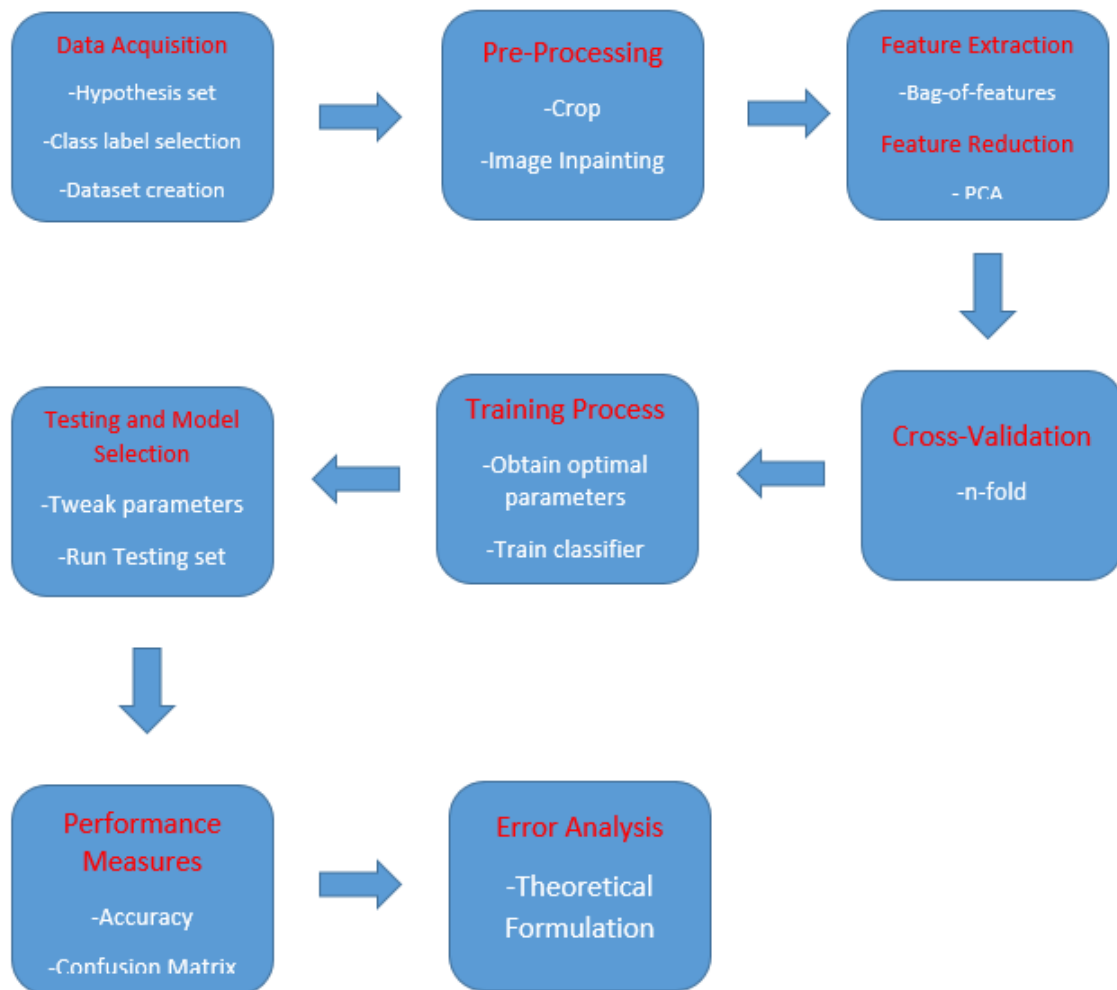


Figure 3: Various phases for the project

7. Split into training, testing and validation: Dataset is been divided into three subsets which has a ration 0.6:0.1:0.3, training, validation and testing respectively.
8. Feature Reduction : Since the feature matrix obtained for each of the sets is huge, it justifies considering some sort of feature reduction method to reduce the dimensionality of the data. The method we've used is Principal Component Analysis (PCA) and it'll be summarized shortly.

Training Process

9. Cross validation: Cross validation is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set.
10. Obtain optimal parameters: The cross validation step gives us the optimal value/s of the parameter/s needed for the model. These could be the penalty term (C) or gamma in case of SVMs or could be the number of hidden layers, number of nodes, etc.
11. Train Classifier: Once the optimal parameters are obtained, the classifier can be trained.

Testing, Validation and model Selection

12. Test on validation set: Once a model is created after learning the training data, the model is tested on the validation set. **Even though we carried out This could be due to a local minima.** The parameters of the model can be tweaked if the results of the validation aren't satisfactory. These trial are repeated till we are satisfied with the model.
13. Obtain robust model: The continuous improvement of the model by repetitive trials of the validation data ensures that the final model we obtain is very robust and resilient to overfitting.
14. Test on testing set: The model is then tested on the testing set.
15. Performance measures: Several performance measures such as accuracy, misclassification rate, confusion matrix, etc are used to judge the performance of the classifier on the testing data.

Error Analysis

16. Error measure: Finally, check whether the out-of-sample error satisfies the expression provided for the error bound.

4 Implementation

4.1 Feature Space

Images of the sewer pipeline are extremely diverse and challenging as shown in Fig. 4. In this figure different class images are shown. As in the image it is evident how diverse the images are in colorspace and also in shape characteristics. In this work, a histogram of a quantized vector is considered to represent repetitions of the image patches (explained in feature extraction section).



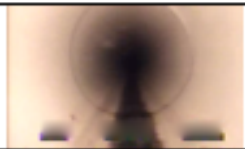

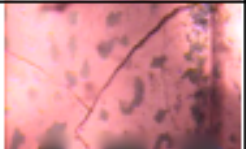





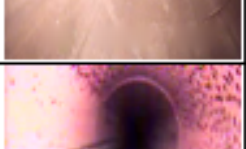
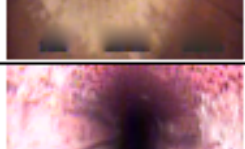
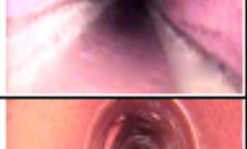

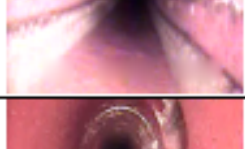






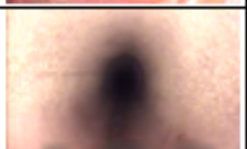



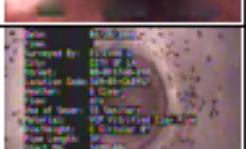
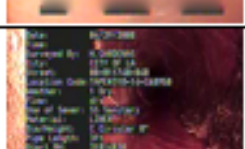
| | | | |
|---------------------------|---|--|---|
| Pipe Joint Without Defect |  |  |  |
| Pipe crack |  |  |  |
| Manhole |  |  |  |
| Corrosion |  |  |  |
| Pipe joint + side pipe |  |  |  |
| Side pipe zoomed |  |  |  |
| Calcinated |  |  |  |
| Text images |  |  |  |
| Debris |  |  |  |

Figure 4: Various classes of images

4.2 Pre-processing and Feature Extraction

Once the class labels were decided, then began the pre-processing part. Each image underwent the following stages of pre-processing and feature extraction:

Cropping the image

All the images are overlaid with text. This extra text and just behaves as noise in the image. Owing to the low quality of the image and the relatively huge size of the mask, inpainting algorithms too failed to provide us with a reasonable approximation. Thus, we adopted the option of cropping the images so that the part overlaid with text is cut and is no longer considered.

Image inpainting

Inpainting is the process of reconstructing lost or deteriorated parts of images and videos. The notion of digital inpainting was first introduced in the paper by Bertalmio et al. (2000). Smart digital inpainting models, techniques, and algorithms have broad applications in image interpolation, photo restoration, zooming and super-resolution, primal-sketch based perceptual image compression and coding, and the error concealment of (wireless) image transmission, etc. Inpainting is rooted in the restoration of images. The methodology followed is :

- The global picture determines how to fill in the gap. The purpose of inpainting is to restore the unity of the work.
- The structure of the gap surroundings is supposed to be continued into the gap. Contour lines that arrive at the gap boundary are prolonged into the gap.
- The different regions inside a gap, as defined by the contour lines, are filled with colors matching for those of its boundary.
- The small details are painted, i.e. “texture” is added.

Inpainting problem: Given an image and a region Ω inside it, the inpainting problem consists in modifying the image values of the pixels in Ω so that this region does not stand out with respect to its surroundings.

Our Method : Our method of image inpainting was based off a weighted minimum norm implementation. Given the image, we generate a mask of the image indicating the locations at which text is present. The algorithm makes use of the mask to determine the region in which inpainting is to be done. Once this region has been identified, all the pixels from this region are made zero. This can be considered equivalent to removing that region from the image. The missing pixels have to be replaced with values in such a way that the L2-norm of the solution is the least. The image (solution) is multiplied with a weighting matrix and at every iteration, the goal is to choose an x that minimizes the l2-norm of the solution. Different constraints are then applied to the image. For example, the image does not have continuous 2D edges, but instead has “jagged” edges in locations where the text occluded an image edge. This type of information could have been restored if we had used a 2D version of the *WMN* constrain that imposed both horizontal and vertical smoothness. Alternatively, we could have used the prior information that this type of painting should have a fairly consistent spatial texture, or the constraint that the painting should be

compressible in a wavelet transform, or the constraint that the painting is possibly compressible under a low-rank matrix representation of the image.

$$\hat{X}_{WMN} = (T^H T)^{-1} A^H (A (T^H T)^{-1} A^H)^{-1} b \quad (6)$$

Here, \hat{X}_{WMN} is the final solution we are interested in.

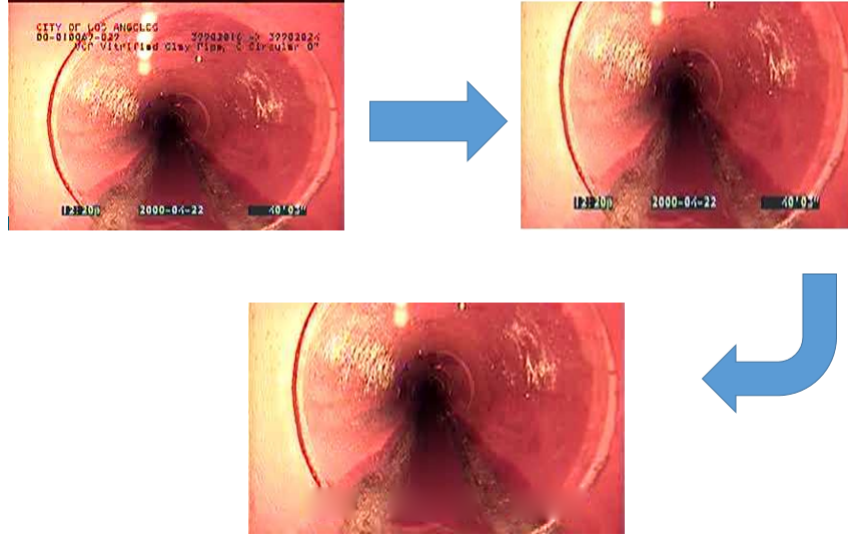


Figure 5: An inpainted image

Feature extraction

In this work, automatic feature extraction such as bag-of-features is been used to extract the feature vector from the given image samples. Firstly, image descriptors such as Speeded Up Robust Features (SURF) is used to obtain the key-points in images (refer Fig. 6). Secondly, descriptor vector that contains the visual information around the key-point is formed for each images. Thirdly,

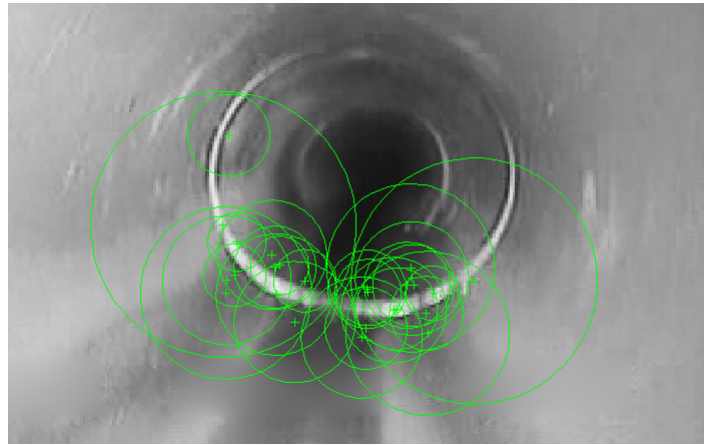


Figure 6: A SURF detector points

key-point descriptor are obtained for all the images in the dataset. Fourthly, these feature descriptors

are clustered to produce a visual vocabulary of k words. Fifthly, in each image the count of the visual words are considered and an histogram is obtained. This vector dimension is equal to the dimension of the feature vector. Lastly, each vector is normalized to a unit vector and to have a consistency. For more details please refer to [Csurka et al. \(2004\)](#).

Feature reduction

Since the feature matrix obtained for each of the sets is huge, it justifies considering some sort of feature reduction method to reduce the dimensionality of the data. The method we've used is Principal Component Analysis (PCA). Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. PCA was invented in 1901 by Karl Pearson ([Jolliffe, 2005](#)). The number of principal components is less than or equal to the number of original variables. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to (i.e., uncorrelated with) the preceding components. The principal components are orthogonal because they are the eigenvectors of the covariance matrix, which is symmetric. The principal components as a whole form an orthogonal basis for the space of the data. The first principal component is a single axis in space. When you project each observation on that axis, the resulting values form a new variable. And the variance of this variable is the maximum among all possible choices of the first axis. The second principal component is another axis in space, perpendicular to the first. Projecting the observations on this axis generates another new variable. The variance of this variable is the maximum among all possible choices of this second axis. The choice of how many dimensions to reduce to is made through visual inspection of the plot of the eigenvalues, also called the 'scree-plot'. The plot for our feature dimension is provided below. From the [Fig. 7](#), original number of dimensions, $D = 500$ and

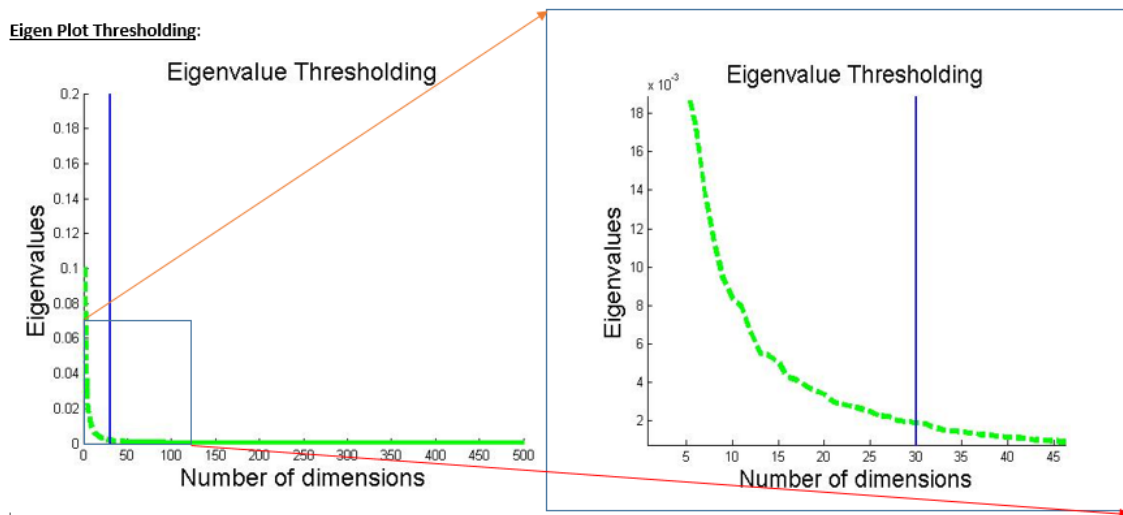


Figure 7: A scree plot of the PCA latency (Eigen values)

reduced number of dimensions, $D' = 30$.

4.3 Training Process

Once filtered and processed feature matrix is ready, we can begin the training process. The first step is Cross Validation. Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. The goal of cross validation is to define a dataset to “test” the model in the training phase (i.e., the validation dataset), in order to limit problems like overfitting, give an insight on how the model will generalize to an independent data set (i.e., an unknown dataset, for instance from a real problem), etc. One round of cross validation involves.

Partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds.

Cross-validation is important in guarding against testing hypotheses suggested by the data (called type III errors) especially where further samples are hazardous, costly or impossible to collect.

1. Obtain optimal parameters: The cross validation step gives us the optimal value/s of the parameter/s needed for the model. These could be the penalty term (C) or gamma in case of SVMs or could be the number of hidden layers, number of nodes, etc.

The optimal parameters obtained for the SVM with RBF Kernel are : $C = 2$ and $\gamma = 16$.

2. Train Classifier in OVR (one-vs-rest mode): Once the optimal parameters are obtained, the classifier can be trained. In machine learning, multiclass or multinomial classification is the problem of classifying instances into more than two classes. While some classification algorithms naturally permit the use of more than two classes, others are by nature binary algorithms; these can, however, be turned into multinomial classifiers by a variety of strategies.

One-vs.-rest

The one-vs.-rest (or one-vs.-all, OvA or OvR) strategy involves training a single classifier per class, with the samples of that class as positive samples and all other samples as negatives. This strategy requires the base classifiers to produce a real-valued confidence score for its decision, rather than just a class label; discrete class labels alone can lead to ambiguities, where multiple classes are predicted for a single sample.

One-vs.-one

In the one-vs.-one (OvO) reduction, one trains $K(K - 1) / 2$ binary classifiers for a K-way multiclass problem; each receives the samples of a pair of classes from the original training set, and must learn to distinguish these two classes. At prediction time, a voting scheme is applied: all $K(K - 1) / 2$ classifiers are applied to an unseen sample and the class that got the highest number of "+1" predictions gets predicted by the combined classifier. Like OvR, OvO suffers from ambiguities in that some regions of its input space may receive the same number of votes.

Number of training samples = 8,617

Original Dimension (D) = 500

New Dimension (D') = 30.

The model is tested repeatedly on the validation set until suitable parameters are obtained. This reduces the extent of overfitting or underfitting.

4.4 Testing, Validation and Model Selection

1. Test on validation set: Once a model is created after learning the training data, the model is tested on the validation set. Even though we carried out This could be due to a local minima. The parameters of the model can be tweaked if the results of the validation aren't satisfactory. These trial are repeated till we are satisfied with the model.
2. Obtain robust model: The continuous improvement of the model by repetitive trials of the validation data ensures that the final model we obtain is very robust and resilient to overfitting.
3. Test on Testing Set: The model is then tested on the testing set.
4. Performance Measures: Several performance measures such as accuracy, misclassification rate, confusion matrix, etc are used to judge the performance of the classifier on the testing data.

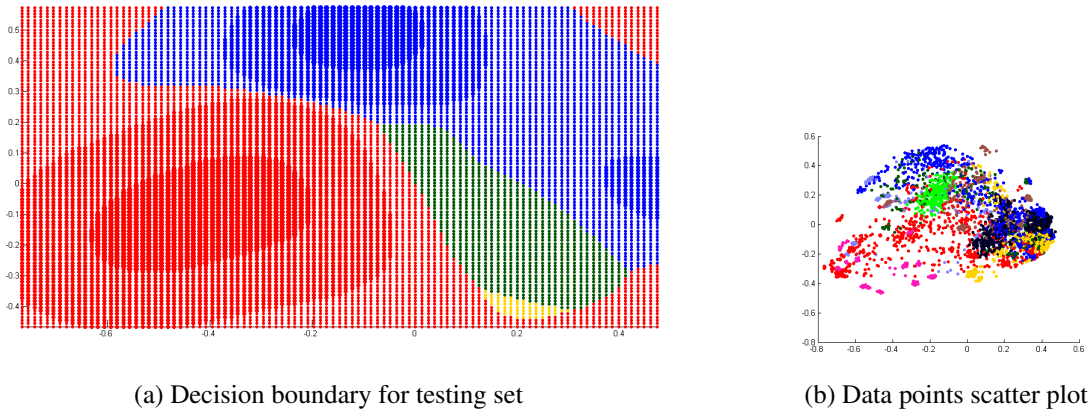


Figure 8: Decision boundary of a SVM classifier

5 Final Results

A new labeled dataset of a magnitude 14362 images is created for this project. This dataset consists of nine different classes. Namely: 1. Pipe joint without defect, 2. debris, 3. corrosion, 4. calcinated, 5. text images, 6. pipe crack, 7. pipe joint + side pipe, 8. side piped zoomed, and 9. manholes. The distribution of these classes is shown in [Fig. 9](#).

The dataset is randomly shuffled and divided into three subsets like: training, testing and validation (60, 30 and 10 %) respectively. Each of these sets are separately encoded in order to produce the feature matrices. Special care is taken such that their labels are still preserved while shuffling.

In this work, two non-linear classifiers are used to perform the object classification. The results of our method are shown in the below sections.

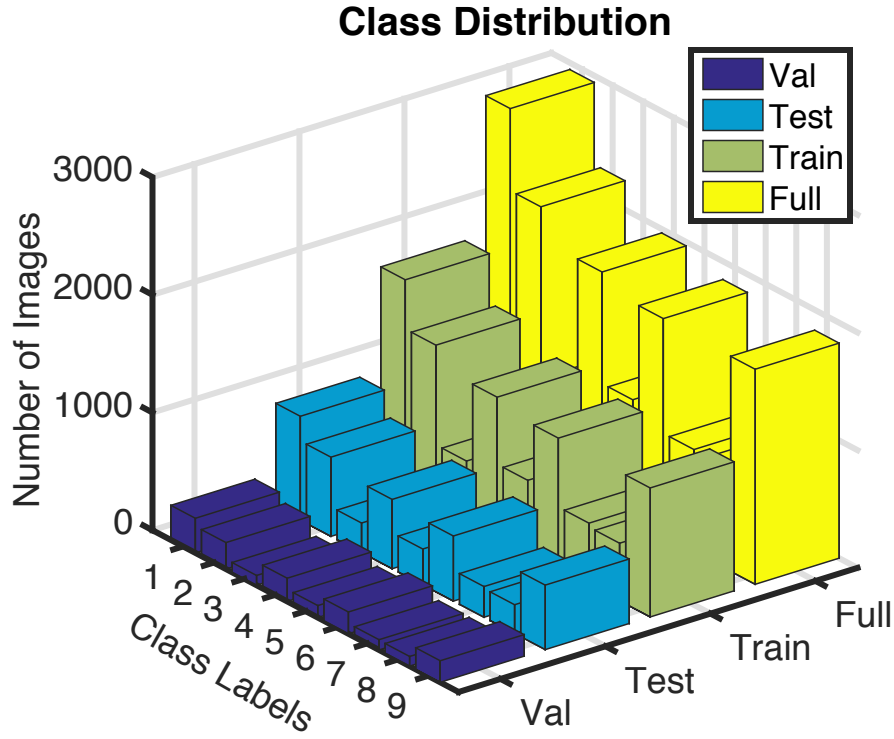


Figure 9: Distribution of the Dataset

5.1 SVM

We made use of an RBF kernel in order to implement the multiclass classification using SVMs. The parameters affecting the performance of the RBF kernel are: C (penalty parameter) and γ . These parameters were selected on the basis of the performance of the model on the validation set. We performed 3-fold cross validation. The default values used by LIBSVM are: $C = 1$, $\gamma = 1/(\text{number of features})$. The values we obtained after $k = 3$ fold cross validation are: $C = 2$, $\gamma = 16$.

Fig. 10 shows the observation made on validation sets. The figure on the left displays the decision values on the validation set. For a given class k , the greenish-blue values denote the samples belonging to the same class k and values that are brighter, towards the red spectrum belong to other classes. These indicate the misclassified samples. The figure on the right provides us with the output labels for class k . Since SVMs use the one-vs-rest methodology, each class is compared to the rest as a whole and labels are set. Similar to the previous figure, the red values indicate that the output labels of those samples are different from the labels assigned to samples belonging to class k .

5.2 Neural network

In this work, multi-layer neural networks are considered to learn the data and to predict the test class labels. Fig. 11 shows the misclassification error rate of neural network with one, two and three hidden layers. Execution time was more than 15 hours to train, validate and test all the dataset for various number of neuron units. It is noted that the best test set misclassification rate are: one

| Class Label | Validation Accuracy (%) | Testing Accuracy (%) |
|---------------------------|-------------------------|----------------------|
| Pipe Joint Without Defect | 79.2768 | 77.4553 |
| Pipe Crack | 84.5619 | 85.1405 |
| Pipe Joint + Side pipe | 94.2281 | 94.1723 |
| Side pipe zoomed | 86.3004 | 86.1853 |
| Manhole | 92.9068 | 92.9185 |
| Debris | 87.1349 | 87.1604 |
| Corrosion | 93.9499 | 93.9401 |
| Calcinated | 94.2976 | 94.3348 |
| Text Images | 87.274 | 87.2765 |
| Mean accuracy | 88.88 | 87.73 |

Table 1: Classifier performance for SVM

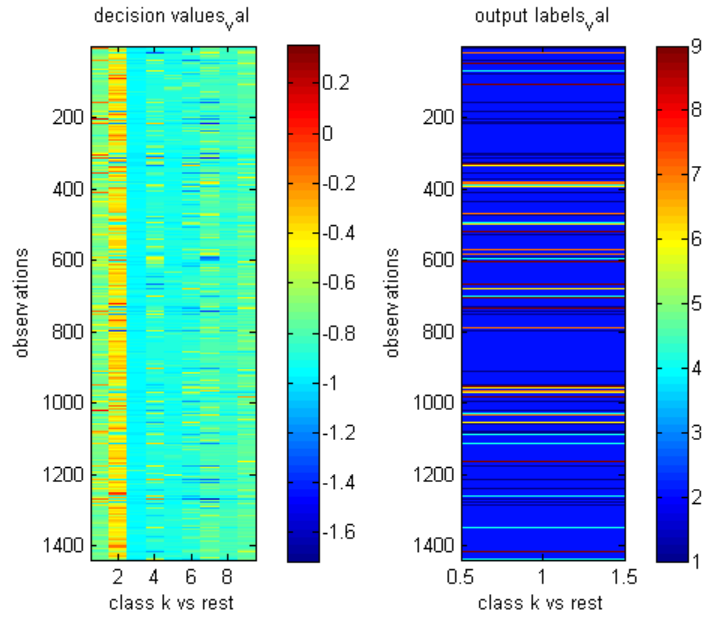


Figure 10: Observations SVM

hidden layer with 65 neurons gave 6.96282e-05, two layers with [570,210] neurons provided 0 and 6.9628e-05 for [420,120,20] neuron units (refer [Table 2](#)).

| Classifier type | Number of Neurons | Best Val. Accuracy (%) | Best Testing Accuracy (%) |
|-----------------|-------------------|------------------------|---------------------------|
| NN 1 HL | 65 | 97.2768 | 99.99 |
| NN 2 HL | 570, 210 | 95.6712 | 100 |
| NN 3 HL | 420, 120, 20 | 96.5619 | 99.99 |

Table 2: Classifier performance for FFNN

6 Interpretation

As can be observed from the results above, Neural Networks perform extremely well on the given data. They are capable of accurately classifying the images into one amongst 9 classes. SVM though, doesn't enjoy this performance. It gives slightly poorer performance than NN does. A possible reason could be the use of Radial Basis Function (RBF) as a kernel instead of a linear kernel. Additionally, multiclass SVMs follow the principle of one-vs-rest (OVR). Thus, even though the individual classification results are good, the overall performance could be quite poor. In OVR, the 'rest' is huge and can get separated easily from the main class. Combining all the single classifications could lead to some competitive effect between winners. This could be the reason behind the low accuracy.

In neural network model, [Fig. 11](#) it was noted that increasing the number of neuron units overfit the data and worked poorly for testing data for one hidden layer NN. In contrast, three hidden layer system performed well for increase in the neurons. But, two layer system on an average under performed. Three layer network for lower neurons worked poorly, as the non-linearity increased with number of layers. Even, one hidden layer with five neurons is able to provide a classifier that has the misclassification error rate of 0.003411. Thus, it is evident that an optimal number of neurons with one layer is sufficient for good classification rate. On the other hand, neural network with two or three hidden layers (permutations of neurons) is computationally heavy.

It is worthwhile to note, that both SVM and FFNN worked well on an average owing to the fact that the overfitting is relaxed due to the large number of training samples 8617. For validation and testing, 1438 and 4307 images are used respectively. As part of a separate experiment, the training set size was decreased, keeping the validation and testing set sizes the same. It was observed that the accuracy drastically decreased in this case, decreasing by almost 50%, which was to be expected.

7 Summary, Conclusions and Future Work

In this work, sewer pipeline defect and non-defect image classification is considered. A new dataset has been manually labeled and partitioned into training, testing and validation sets. A total of 14362 images under nine classes are formed by using 100 randomly selected sewer pipe videos. Furthermore, two non-linear classifiers such as SVM and FFNN are used to predict the class labels. These classifiers are selected based on the number of samples and prior observations made in the

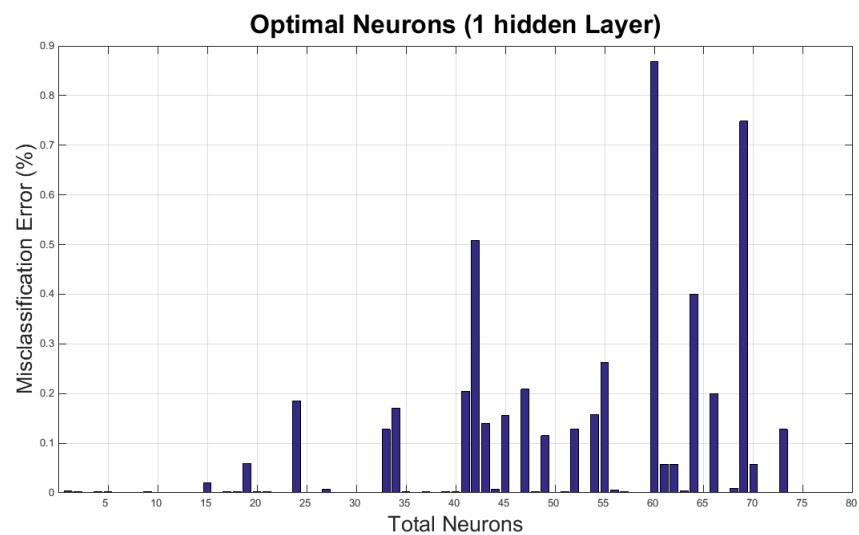
research community. Lastly, an attempt has been made to understand the generalization bound for multiple classes as an extension to the coursework.

Results obtained from both the classifiers were surprising and satisfactory. It is observed that non-uniform distribution of the images in each class might have biased the accuracy of the SVM classifier. This is due to the fact that current classification method is one-vs.-rest. Neural network even with one layer and five neurons performed tremendously well with a classification accuracy more than 99.5%. Lastly, higher training samples relaxed the overfitting and helped in the good performance of the classifiers.

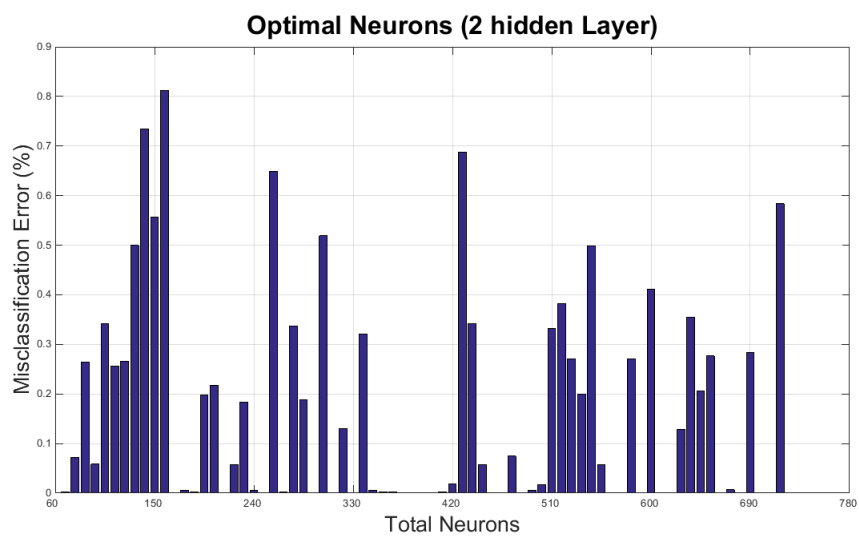
As a continuation, in the near future some of these suggestions will be implemented. Firstly, instead of using image inpainting, a video algorithm will be used to remove the text autonomously. Secondly, each images are obtained by frame-extraction of a video. Thus, augmenting temporal scale might be an interesting problem to solve. **Thirdly, due to time constraints, an hybrid parallel algorithm will be developed.** Fourthly, bag-of-feature, although performed very well, it is worth to consider the explicit shape characteristics to localize the objects of interest (includes intra-class items too) within each image. Lastly, a closed form solution of generalization bound that is pertinent for SVM and FFNN will be improved.

8 Acknowledgments

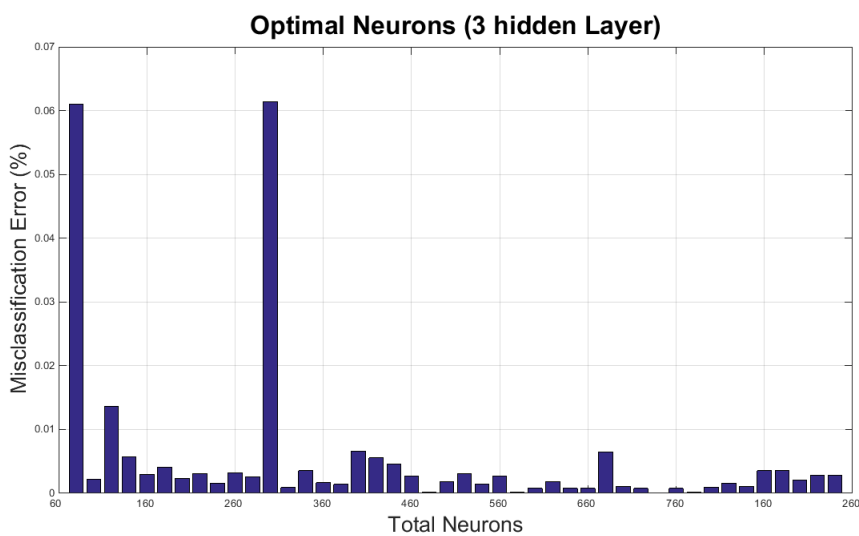
Firstly, the authors are highly thankful to Professor Keith Jenkins for his teaching and guiding throughout the semester. Secondly, the authors would also like to thank Shuyang Sheng, TA for EE 660 for his immense support in explaining homework assignments and project guidelines. Lastly, the authors are thankful to graduate students Vinay Hegde and Tanmay Patil working in USC, SHM lab for their conscientious efforts in creating the world's largest sewer pipeline ground-truth image database!



(a) One hidden layer



(b) Two hidden layer



9 Bibliography

- Abu-Mostafa, Y. S., M. Magdon-Ismael, and H.-T. Lin (2012). *Learning from data*. AMLBook.
- ASCE (2013). ASCE report card for america's infrastructure (report card). <http://www.infrastructurereportcard.org/>.
- Bertalmio, M., G. Sapiro, V. Caselles, and C. Ballester (2000). Image inpainting. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 417–424. ACM Press/Addison-Wesley Publishing Co.
- Csurka, G., C. Bray, C. Dance, and L. Fan (2004). Visual categorization with bags of keypoints. *Workshop on Statistical Learning in Computer Vision, ECCV*, 1–22.
- Fieguth, P. W. and S. K. Sinha (1999). Automated analysis and detection of cracks in underground scanned pipes. In *ICIP (4)*, pp. 395–399.
- Hausser, D. and P. M. Long (1995). A generalization of sauer's lemma. *Journal of Combinatorial Theory, Series A* 71(2), 219–240.
- Iseley, T. (1999). Pipeline condition assessment: Achieving uniform defect ratings. Volume 1, pp. 121–132. *Infra 99 Intl. Conf.*
- Iyer, S. and S. K. Sinha (2006). Segmentation of pipe images for crack detection in buried sewers. *Computer-Aided Civil and Infrastructure Engineering* 21(6), 395–410.
- Iyer, S. and S. K. Sinha (2013). Automated condition assessment of buried sewer pipes based on digital imaging techniques. *Journal of the Indian Institute of Science* 85(5), 235.
- Jahanshahi, M. R., S. F. Masri, C. W. Padgett, and G. S. Sukhatme (2013). An innovative methodology for detection and quantification of cracks through incorporation of depth perception. *Machine vision and applications* 24(2), 227–241.
- Jahanshahi, M. R., S. F. Masri, and G. S. Sukhatme (2011). Multi-image stitching and scene reconstruction for evaluating defect evolution in structures. *Structural Health Monitoring* 10(6), 643–657.
- Jolliffe, I. (2005). *Principal component analysis*.
- Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.
- Sinha, S. K. and P. W. Fieguth (2006a). Automated detection of cracks in buried concrete pipe images. *Automation in Construction* 15(1), 58 – 72.
- Sinha, S. K. and P. W. Fieguth (2006b). Segmentation of buried concrete pipe images. *Automation in Construction* 15(1), 47 – 57.
- Tewari, A. and P. L. Bartlett (2014). Learning theory. In P. S. Diniz, J. A. Suykens, R. Chellappa, and S. Theodoridis (Eds.), *Academic Press Library in Signal Processing: Volume 1 Signal Processing Theory and Machine Learning*, Volume 1 of *Academic Press Library in Signal Processing*, Chapter 14, pp. 775–816. Elsevier.

Vapnik, V. N. and A. Y. Chervonenkis (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications* 16(2), 264–280.

Appendices

A Main Code

```
%//%*****%
%//%*                               Ph.D                               *%
%//%*                               Project Sewer Pipe                 *%
%//%*                               *%                                *%
%//%*                               Name: Preetham Aghalaya Manjunatha   *%
%//%*                               Sharavan Ravi                       *%
%//%*                               USC ID Number: 7356627445           *%
%//%*                               USC Email: aghalaya@usc.edu         *%
%//%*                               Submission Date: 12/08/2012        *%
%//%*****%
%//%*                               Viterbi School of Engineering,      *%
%//%*                               Sonny Astani Dept. of Civil Engineering,*%
%//%*                               University of Southern california,   *%
%//%*                               Los Angeles, California.            *%
%//%*****%

%% Start parameters
%-----
clear all; close all; clc; %#ok<CLSCR>
Start = tic;
clcwaitbarz = findall(0,'type','figure','tag','TMWWaitbar');
delete(clcwaitbarz);

%% Inputs
%-----
MainInputs;

%% Pre-processing steps
%-----
% Create image sets
% if (shuffleNpartfiles_inpstruct.flagswitch && ~ exist('ZZZ_imageSetsFull.mat','file'))
%     imgPartitionStruct = makePartitions( shuffleNpartfiles_inpstruct );
% else
%     load ZZZ_imageSetsFull.mat
% end

% Image inpainting
% if (cropNinpaint_inpstruct.flagswitch)
%     largescaleImInpaint( imgPartitionStruct.imSetImgLocFull );
% end

%% Processing step
%-----
% Extract feature matrix and class labels
%-----

% A hybrid algorithm (RoboCRACK)
```

```

% if (largefeatmatlabel_inpstruct.flagswitch)
%     featlab_traintestset = largefeaturematrixNlabels ...
%         (hybrid_inpstruct, imgPartitionStruct);
% end

% Bag-of-Features
% bag_Tr_BoFOri = bagOfFeatures(imgPartitionStruct.trainingSets);
% featureMatrixTr_BoFOri = encode(bag_Tr_BoFOri, imgPartitionStruct.trainingSets);
% labelArrayTr_BoFOri = makeLabelArray(imgPartitionStruct.trainingSets);
%
% bag_Vl_BoFOri = bagOfFeatures(imgPartitionStruct.validationSets);
% featureMatrixVl_BoFOri = encode(bag_Vl_BoFOri, imgPartitionStruct.validationSets);
% labelArrayVl_BoFOri = makeLabelArray(imgPartitionStruct.validationSets);
%
% bag_Ts_BoFOri = bagOfFeatures(imgPartitionStruct.trainingSets);
% featureMatrixTs_BoFOri = encode(bag_Ts_BoFOri, imgPartitionStruct.testingSets);
% labelArrayTs_BoFOri = makeLabelArray(imgPartitionStruct.testingSets);

%-----
% Training, validation and testing
%-----

% A hybrid algorithm (RoboCRACK)

% Bag-of-Features
load ZZZ_imageSetsFull_BoF_Original.mat
categoryClassifier = trainImageCategoryClassifier(imgPartitionStruct.trainingSets, bag_Tr_BoFOri, labelArrayTr_BoFOri);
[confMatTr,knownLabelIdxTr,predictedLabelIdxTr,scoreTr] = evaluate(categoryClassifier, ...
    imgPartitionStruct.trainingSets);

[confMatVl,knownLabelIdxVl,predictedLabelIdxVl,scoreVl] = evaluate(categoryClassifier, ...
    imgPartitionStruct.validationSets);

[confMatTs,knownLabelIdxTs,predictedLabelIdxTs,scoreTs] = evaluate(categoryClassifier, ...
    imgPartitionStruct.testingSets);

%% Post-processing steps
%-----

%% End parameters
%-----

clcwaitbarz = findall(0,'type','figure','tag','TMWWaitbar');
delete(clcwaitbarz);
Runtime = toc(Start);

%%
clc; clear all; close all;
tic;
% addpath C:\Users\Shravan\Google Drive\Shared with Dell\...
%     Team SewerPipe\Programs\Matlab Functions\libsvm-3.20\...
%     matlab\multiclass;
% addpath C:\Users\Shravan\Google Drive\Shared with Dell\...
%     Team SewerPipe\Programs\Main\src;

```

```

load ('ZZZ_imageSetsFinalFeatMatLabels.mat');

%%
% Load Training, Validation & Testing Data
[test_Y]      = YTest;
[test_feature] = XTest;
[test_x]      = double(test_feature);
Ntest        = size (test_x , 1);
[val_Y]       = Yval;
[val_feature]  = Xval;
[val_x]       = double(val_feature);
Nval         = size (val_x , 1);
[train_Y]     = Ytrain;
[train_feature] = Xtrain;
[train_x]     = double(train_feature);
Ntrain       = size (train_x , 1);

% Shuffle
% [sortedTrainLabel, permIndex] = sortrows(rawTrainLabel);
% sortedTrainData = rawTrainData(permIndex,:);
% [sortedTestLabel, permIndex] = sortrows(rawTestLabel);
% sortedTestData = rawTestData(permIndex,:);
% (OR)
% [ train_X, train_Y, Target ] = shuffleFeatMatLabel( train_x, train_label );
% [ val_X, val_Y, Target ]    = shuffleFeatMatLabel( val_x, val_label );
%

%% Feature Reduction
% FOR TRAINING
Feature_Matrix = train_x;
% Remove inf and NAN
Feature_Matrix(isinf(Feature_Matrix) | isnan(Feature_Matrix)) = 0;

% Remove zero rows
Feature_Matrix( all(~Feature_Matrix,2), : ) = [];

% Remove zero columns
Feature_Matrix( :, all(~Feature_Matrix,1) ) = [];

%Remove duplicate columns
feat_mat_trans = Feature_Matrix';
unq_feat_mat   = unique (feat_mat_trans , 'rows' , 'stable');
feat_matrix    = unq_feat_mat';

% Use pca to obtain the eigenvalues
[coeff , score , eigenvalues] = pca (feat_matrix);
% plot (eigenvalues);
% By visual inspection, numFeatures to be reduced to : k = 30
k      = 30;
[coeff_1 , score_1 , eigenvalues_1] = pca (feat_matrix , 'NumComponents' , k);

train_X = score_1;

% FOR VALIDATION
Feature_Matrix_V = val_x;

```

```

% Remove inf and NAN
Feature_Matrix_V(isinf(Feature_Matrix_V) | isnan(Feature_Matrix_V)) = 0;

% Remove zero rows
Feature_Matrix_V( all(~Feature_Matrix_V,2), : ) = [];

% Remove zero columns
Feature_Matrix_V( :, all(~Feature_Matrix_V,1) ) = [];

%Remove duplicate columns
feat_mat_trans_V = Feature_Matrix_V';
unq_feat_mat_V    = unique (feat_mat_trans_V , 'rows' , 'stable');
feat_matrix_V     = unq_feat_mat_V';

% Use pca to obtain the eigenvalues
[coeff_V , score_V , eigenvalues_V] = pca (feat_matrix_V);
% plot (eigenvalues);
% By visual inspection, numFeatures to be reduced to : k = 30
k      = 30;
[coeff_2 , score_2 , eigenvalues_2] = pca (feat_matrix_V , 'NumComponents' , k);
val_X = score_2;

% FOR TESTING :
Feature_Matrix_T = test_x;
% Remove inf and NAN
Feature_Matrix_T(isinf(Feature_Matrix_T) | isnan(Feature_Matrix_T)) = 0;

% Remove zero rows
Feature_Matrix_T( all(~Feature_Matrix_T,2), : ) = [];

% Remove zero columns
Feature_Matrix_T( :, all(~Feature_Matrix_T,1) ) = [];

%Remove duplicate columns
feat_mat_trans_T = Feature_Matrix_T';
unq_feat_mat_T    = unique (feat_mat_trans_T , 'rows' , 'stable');
feat_matrix_T     = unq_feat_mat_T';

% Use pca to obtain the eigenvalues
[coeff_T , score_T , eigenvalues_T] = pca (feat_matrix_T);
% plot (eigenvalues);
% By visual inspection, numFeatures to be reduced to : k = 30
k      = 30;
[coeff_3 , score_3 , eigenvalues_3] = pca (feat_matrix_T , 'NumComponents' , k);
test_X = score_3;

totalData = [train_X; test_X];
totalLabel = [train_Y; test_Y];
NClass = 9;
%%
% % #####
% % Automatic Cross Validation
% % Parameter selection using n-fold cross validation
% % #####
% stepSize = 10;

```

```

% bestLog2c = 1;
% bestLog2g = -1;
% epsilon = 0.01;
% bestcv = 0;
% Ncv = 3; % Ncv-fold cross validation cross validation
% deltacv = 10^6;
% t = 0;
%
% while abs(deltacv) > epsilon
%     bestcv_prev = bestcv;
%     prevStepSize = stepSize;
%     stepSize = prevStepSize/2;
%     log2c_list = bestLog2c-prevStepSize:stepSize/2:bestLog2c+prevStepSize;
%     log2g_list = bestLog2g-prevStepSize:stepSize/2:bestLog2g+prevStepSize;
%
%     numLog2c = length(log2c_list);
%     numLog2g = length(log2g_list);
%     cvMatrix = zeros(numLog2c,numLog2g);
%
%     for i = 1:numLog2c
%         log2c = log2c_list(i);
%         for j = 1:numLog2g
%             log2g = log2g_list(j);
%             cmd = ['-q -c ', num2str(2^log2c), ' -g ', num2str(2^log2g)];
%             cv = get_cv_ac(train_Y, train_X, cmd, Ncv);
%             if (cv >= bestcv),
%                 bestcv = cv; bestc = 2^log2c; bestg = 2^log2g;
%             end
%         end
%     end
%     deltacv = bestcv - bestcv_prev;
%
% end
% disp(['The best parameters, yielding Accuracy=',num2str(bestcv*100),'%', are: C=',num2str

%
% #####
% Train the SVM in one-vs-rest (OVR) mode
% #####
bestc = 2;
bestg = 16;
bestParam = ['-q -c ', num2str(bestc), ' -g ', num2str(bestg)];
% bestParam = ['-q -c 8 -g 0.0625'];
model = ovrtrainBot(train_Y, train_X, bestParam);

% FOR VVALIDATION :
% #####
% Classify samples using OVR model
% #####
[predict_label_val, accuracy_val, decis_values_val] = ovrpredictBot(val_Y, val_X, model);
[decis_value_winner_val, label_out_val] = max(decis_values_val,[],2);
%
% % #####
% % Make confusion matrix
% % #####

```

```

% FOR VALIDATION :
[confusionMatrix_val ,order_val] = confusionmat(val_Y , label_out_val);
% % Note: For confusionMatrix
% % column: predicted class label
% % row: ground-truth class label
% % But we need the conventional confusion matrix which has
% % column: actual
% % row: predicted
% figure; imagesc(confusionMatrix_val);
% title ('Confusion Matrix for Validation Set' , 'FontSize' , 18);
% xlabel('actual class label' , 'FontSize' , 18);
% ylabel('predicted class label' , 'FontSize' , 18);
totalAccuracy_val = trace(confusionMatrix_val)/Nval;
disp(['Total accuracy from the SVM: ',num2str(totalAccuracy_val*100),'%']);

% FOR TESTING :
% #####
% Classify samples using OVR model
% #####
[predict_label_test, accuracy_test, decis_values_test] = ovrpredictBot(test_Y, test_X, model);
[decis_value_winner_test, label_out_test] = max(decis_values_test,[],2);

% % #####
% % Make confusion matrix
% % #####
[confusionMatrix_test ,order_test] = confusionmat(test_Y , label_out_test);
% figure; imagesc(confusionMatrix_val);
% title ('Confusion Matrix for Validation Set' , 'FontSize' , 18);
% xlabel('actual class label' , 'FontSize' , 18);
% ylabel('predicted class label' , 'FontSize' , 18);
totalAccuracy_val = trace(confusionMatrix_val)/Nval;
disp(['Total accuracy from the SVM: ',num2str(totalAccuracy_val*100),'%']);
%
%%
% #####
% Plot the results
% #####
% figure;
% % subplot(1,3,2); imagesc(predict_label_val); title('predicted labels_val'); xlabel('class k vs real label');
% subplot(1,2,1); imagesc(decis_values_val); title('decision values_val'); xlabel('class k vs real label');
% subplot(1,2,2); imagesc(label_out_val); title('output labels_val'); xlabel('class k vs real label');

% % plot the true label for the test set
% patchSize = 20*exp(decis_value_winner_test);
% % colorList = generateColorList(NClass);
% % colorPlot = colorList(test_Y,:);
% figure;
% scatter(test_X(:,1),test_X(:,2),patchSize , 'filled'); hold on;
%
% % plot the predicted labels for the test set
% patchSize = 10*exp(decis_value_winner_test);
% % colorPlot = colorList(label_out_test,:);
% scatter(test_X(:,1),test_X(:,2),patchSize , 'filled');

%%

```



```

%% #####
%% Plot the decision boundary
%% #####

% Generate data to cover the domain
minData = min(totalData,[],1);
maxData = max(totalData,[],1);
stepSizePlot = (maxData-minData)/50;
[xI yI] = meshgrid(minData(1):stepSizePlot(1):maxData(1),minData(2):stepSizePlot(2):maxData(2));
%% #####
%% Classify samples using OVR model
%% #####
[pdl, acc, dcsv] = ovrpredictBot(xI(:)*0, [xI(:) yI(:)], model);
%% Note: when the ground-truth labels of testData are unknown, simply put
%% any random number to the testLabel
[dcsv_winner, label_domain] = max(dcsv,[],2);
%
% plot the result
patchSize = 20*exp(dcsv_winner);
% colorList = generateColorList(NClass);
% colorPlot = colorList(label_domain,:);
figure;
scatter(xI(:),yI(:),patchSize , 'filled');

%%
Runtime = toc;
%% Cite:
% 1. libsvm for MATLAB?, Kittipat's Homepage
%   https://sites.google.com/site/kittipat/libsvm\_matlab/complete\_libsvm\_example

%%/%%*****%
%%/%%*                               Ph.D                               *%
%%/%%*                               Crack Package                       *%
%%/%%*                               *%
%%/%%*                               Name: Preetham Aghalaya Manjunatha    *%
%%/%%*                               USC ID Number: 7356627445             *%
%%/%%*                               USC Email: aghalaya@usc.edu           *%
%%/%%*                               Submission Date: --/--/2012          *%
%%/%%*****%
%%/%%*                               Viterbi School of Engineering,        *%
%%/%%*                               Sonny Astani Dept. of Civil Engineering,*%
%%/%%*                               University of Southern california,     *%
%%/%%*                               Los Angeles, California.              *%
%%/%%*****%
%% Notes:
% 70 - 15 - 15
% 1 HL = Min error = 0.0892 | no. of units (neurons) = 265             | Runtime = 358
secs.
% 2 HL = Min error = 0.0693 | no. of units (neurons) = 305,65          | Runtime = 4.89 hours
% 3 HL = Min error = 0.0812 | no. of units (neurons) = 290,215,395     | Runtime = 3.95 hours

% 50 - 5 - 45
% 1 HL = Min error = 0.1345 | no. of units (neurons) = 355             | Runtime = 280
secs.

```

```

% 2 HL = Min error = 0.1150 | no. of units (neurons) = 685,645      | Runtime = 4.04 hours
% 3 HL = Min error = 0.1239 | no. of units (neurons) = 525,615,675  | Runtime = 7.42 hours

%% Start parameters
%%/%*****%
tic;
clear all; close all; clc;
clcwaitbarz = findall(0,'type','figure','tag','TMWWaitbar');
delete(clcwaitbarz);

%% This script assumes these variables are defined:
%%/%*****%
%   Feature_matrix - input data
%   Labels - target data

% Change of variables
% [ 1 2 3 4 5 6 6 7 7 8 8 8 ..... 12]
% [ 1 2 3 4 5 6 6 7 7 8 8 8 ..... 12]
% [ 1 2 3 4 5 6 6 7 7 8 8 8 ..... 12]
% . . . . .
% [ 1 2 3 4 5 6 6 7 7 8 8 8 ..... 12]
% [ 1 2 3 4 5 6 6 7 7 8 8 8 ..... 12]
% rows x columns [n x m]
% n - samples; m - features (requires transpose). Similarly, to
% labels/targets
% If not the above format, then no need to transpose

% Feature matrix (transposed) and % Labels/targets (transposed)
load ZZZ_imageSetsFinalFeatMatLabels.mat
x = [Xtrain; Xval; XTest]';
t = [Ttrain; Tval; TTest]';

% x = (compute_mapping(x', 'PCA', 50))';
% Training, validation and testing size (1 to 100%)
% Caution: Total shall make 100%
% train_size = 50;
% valid_size = 5;
% test_size = 45;

% Number of hidden layers [maximum 3]
hidden_layers = 1;

% Hidden layer size
% hiddenLayerSize_Vec = 5:5:1000;    % 1HL
% hiddenLayerSize_Vec = 30:90:1000;  % 2HL
hiddenLayerSize_Vec = 20:100:800;    % 3 HL

% Window view / plotting options [on | off]
plotter = 'no';
viewfinalnet = 'off';

%% Create hidden layers (>1) neuron units combinations
%%/%*****%
switch hidden_layers

```

```

case 1
    pairs = hiddenLayerSize_Vec(:);

case 2
    [p,q] = meshgrid(hiddenLayerSize_Vec, hiddenLayerSize_Vec);
    pairs = [p(:) q(:)];

case 3
    [p,q,r] = meshgrid(hiddenLayerSize_Vec, hiddenLayerSize_Vec, hiddenLayerSize_Vec);
    pairs = [p(:) q(:) r(:)];
end

%% Train the network
%%//%*****

% Waitbar handler
h = waitbar(0, 'Initializing...', 'Name', 'Finding optimum number of neurons...!', ...
    'CreateCancelBtn', ...
    'setappdata(gcf, 'canceling', 1)');
setappdata(h, 'canceling', 0)

% Main loop
for step = 1 : 1 %length(pairs)

    % Check for Cancel button press
    if getappdata(h, 'canceling')
        break
    end

    % Create a Pattern Recognition Network
    net = patternnet(300);

    % Network options
    %%//%*****
    % Choose Input and Output Pre/Post-Processing Functions
    % For a list of all processing functions type: help nnprocess
    net.input.processFcns = {'removeconstantrows', 'mapminmax'};
    net.output.processFcns = {'removeconstantrows', 'mapminmax'};

    % Setup Division of Data for Training, Validation, Testing
    % For a list of all data division functions type: help nndivide
    net.divideFcn = 'divideind'; % Divide data randomly
    net.divideMode = 'none'; % Divide up every sample
    net.divideParam.trainInd = 1 : size(Xtrain,1);

    net.divideParam.valInd = size(Xtrain,1) + 1 : ...
        size(Xtrain,1) + size(Xval,1);

    net.divideParam.testInd = size(Xtrain,1) + size(Xval,1) + 1 : ...
        size(Xtrain,1) + size(Xval,1) + ...
        size(XTest,1);

    % Goal
    net.trainParam.goal = 1e-3;

```

```

% Change the transfer function for all hidden layers [output layer in default 'softmax'
for i = 1:size(pairs,2)
    net.layers{i}.transferFcn = 'tansig';
end

% Turn on/off nntraintoll window
net.trainParam.showWindow = 0;

% Callback of neural network function
% Train the Network
[net,tr] = train(net,x,t,'useParallel','yes','useGPU','yes');

% Test the Network
y = net(x,'useParallel','yes','useGPU','yes');
e = gsubtract(t,y);
tind = vec2ind(t);
yind = vec2ind(y);
percentErrors = sum(tind ~= yind)/numel(tind);
performance = perform(net,t,y);

% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y);
valPerformance = perform(net,valTargets,y);
testPerformance = perform(net,testTargets,y);

% [netOutput, tr, test_Errors, overallErrors] = funct_NNtrain_test (net, x, t);
% netOutputTotal(step).network = net;
% trTotal(step).trainNN = tr;

% Store error outputs
NNerrDetails(step).testErrTotal = percentErrors;
NNerrDetails(step).neuronUnits = pairs(step,:);

% Report current estimate in the waitbar's message field
waitbar(step/length(pairs), h, sprintf('Hidden layer units itr. = %i | Err: %1.3f',...
    step, percentErrors))
end

% DELETE the waitbar; don't try to CLOSE it
delete(h)

%% View network
if (strcmp(viewfinalnet, 'on'))
    view(finalnet)
end

%% End parameters
% Close figures, waitbars and all
clcwaitbarz = findall(0,'type','figure','tag','TMWWaitbar');
delete(clcwaitbarz);

```

```
% Close nntraintool window
nntraintool('close');
```

```
% Runtime
Runtime = toc;
```

B Sub-routines

```
function [cropped_img , mask] = crop (input_img)
% Description
```

```
I = imread (input_img);
cropped_img = imcrop (I , [0 60 352 240]);
```

```
M = zeros (181,352);
M(145:170 , 51:101) = 255;
M(145:170 , 132:215) = 255;
M(145:170 , 246:318) = 255;
mask = M;
```

```
% imwrite (input_img , input_img '.jpg' , '.jpg');
```

```
% M_r = cropped_img(: , : , 1);
% M_g = cropped_img(: , : , 2);
% M_b = cropped_img(: , : , 3);
%
```

```
% M_r(145:170 , 51:101) = 255;
% M_r(145:170 , 132:215) = 255;
% M_r(145:170 , 246:318) = 255;
```

```
%
% M_g(145:170 , 51:101) = 255;
% M_g(145:170 , 132:215) = 255;
% M_g(145:170 , 246:318) = 255;
```

```
%
% M_b(145:170 , 51:101) = 255;
% M_b(145:170 , 132:215) = 255;
% M_b(145:170 , 246:318) = 255;
```

```
% % M_b(~(145:170 , 51:101)) && M_b(~(145:170 , 132:215)) && M_b(~(145:170 , 246:318)) = 0;
%
```

```
% mask_r = M_r;
% mask_g = M_g;
% mask_b = M_b;
```

```
end
```

```
function [cropped_img , mask] = cropnMask (input_img)
% Description
```

```
I = imread (input_img);
cropped_img = imcrop (I , [0 60 352 240]);
```

```
end
```

```

function [cs,index] = filenamesort(c,mode)
%sort_nat: Natural order sort of cell array of strings.
% usage:  [S,INDEX] = sort_nat(C)
%
% where,
%   C is a cell array (vector) of strings to be sorted.
%   S is C, sorted in natural order.
%   INDEX is the sort order such that S = C(INDEX);
%
% Natural order sorting sorts strings containing digits in a way such that
% the numerical value of the digits is taken into account.  It is
% especially useful for sorting file names containing index numbers with
% different numbers of digits.  Often, people will use leading zeros to get
% the right sort order, but with this function you don't have to do that.
% For example, if C = {'file1.txt','file2.txt','file10.txt'}, a normal sort
% will give you
%
%       {'file1.txt'  'file10.txt'  'file2.txt'}
%
% whereas, sort_nat will give you
%
%       {'file1.txt'  'file2.txt'  'file10.txt'}
%
% See also: sort

% Version: 1.4, 22 January 2011
% Author:  Douglas M. Schwarz
% Email:    dmschwarz=ieee*org, dmschwarz=urgrad*rochester*edu
% Real_email = regexp(Email,{'=','*'},{'@','.'})

% Set default value for mode if necessary.
if nargin < 2
    mode = 'ascend';
end

% Make sure mode is either 'ascend' or 'descend'.
modes = strcmpi(mode,{'ascend','descend'});
is_descend = modes(2);
if ~any(modes)
    error('sort_nat:sortDirection',...
        'sorting direction must be ''ascend'' or ''descend''.')
end

% Replace runs of digits with '0'.
c2 = regexp(c,'\d+','0');

% Compute char version of c2 and locations of zeros.
s1 = char(c2);
z = s1 == '0';

% Extract the runs of digits and their start and end indices.
[digruns,first,last] = regexp(c,'\d+','match','start','end');

% Create matrix of numerical values of runs of digits and a matrix of the

```

```

% number of digits in each run.
num_str = length(c);
max_len = size(s1,2);
num_val = NaN(num_str,max_len);
num_dig = NaN(num_str,max_len);
for i = 1:num_str
    num_val(i,z(i,:)) = sscanf(sprintf('%s ',digruns{i}{:}),'%f');
    num_dig(i,z(i,:)) = last{i} - first{i} + 1;
end

% Find columns that have at least one non-NaN. Make sure activecols is a
% 1-by-n vector even if n = 0.
activecols = reshape(find(~all(isnan(num_val))),1,[]);
n = length(activecols);

% Compute which columns in the composite matrix get the numbers.
numcols = activecols + (1:2:2*n);

% Compute which columns in the composite matrix get the number of digits.
ndigcols = numcols + 1;

% Compute which columns in the composite matrix get chars.
charcols = true(1,max_len + 2*n);
charcols(numcols) = false;
charcols(ndigcols) = false;

% Create and fill composite matrix, comp.
comp = zeros(num_str,max_len + 2*n);
comp(:,charcols) = double(s1);
comp(:,numcols) = num_val(:,activecols);
comp(:,ndigcols) = num_dig(:,activecols);

% Sort rows of composite matrix and use index to sort c in ascending or
% descending order, depending on mode.
[unused,index] = sortrows(comp);
if is_descend
    index = index(end:-1:1);
end
index = reshape(index,size(c));
cs = c(index);

function Ai = generate_Ai(m_i);
    I = eye(size(m_i , 1));
    Ai = I(find(not(m_i)),:);
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Shuffle and partition params

%-----
% Shuffle flag
%-----

```

```

shuffleNpartfiles_inpstruct.flagswitch          = 1;    %[0-off | 1-on]

%-----
% Training percentage
% [train val test] - need first two values
%-----
shuffleNpartfiles_inpstruct.splitRatio    = [0.6 0.1];

%-----
% File renaming
% seqnum | actual -- string datatype
%-----
shuffleNpartfiles_inpstruct.splitShuffleType    = 'randomized';

%-----
% Folder paths for ground-truth, original. Train, test corresponds to the
% new repository folder to store
%-----

% Ground-truth
shuffleNpartfiles_inpstruct.imgFolder        = 'E:\Google Drive\Team SewerPipe\Progr

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Texture feature vector params

%-----
% Shuffle flag
%-----
texturefeature_inpstruct.flag_texturefeatures    = 0;    %[0-off | 1-on]

%-----
% Training images (data samples) to be considered
%-----
% full - uses all images in train folders (string)
% half - uses all images in train folders (string)
% n      - n number of images (integer)
texturefeature_inpstruct.imagenumbers          = 'full';

%-----
% Texture filter
%-----
% laws - Laws multi-channel filter (level, edge, spot, wave and ripple)
% sfta - Segmentation-based Fractal Texture Analysis
texturefeature_inpstruct.texturefilter_type    = 'laws';

%-----
% GPU array
%-----
% yes - creates GPU array (note: works for certain Matlab functions)
% no  - non GPU array
texturefeature_inpstruct.gpuarray              = 'no';

```



```

%-----
% Norm type
%-----
% L1          - L1 norm
% L2          - l2 norm
% infinity    - infinity norm
% frobenius   - frobenius norm
texturefeature_inpstruct.normtype           = 'frobenius';

%-----
% Window size (energy)
%-----
% n          - odd integer value (such as 3, 5, 7, 11, 13, 15, 17, ...)
texturefeature_inpstruct.windowsize        = 15;

%-----
% Training data folder path
%-----
texturefeature_inpstruct.folderpath_texture = [];

%-----
% Provide names of original data folders (include name of the folder even its folder
% is empty)
%-----
texturefeature_inpstruct.originaldata_folders = {'', ''};
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Crop and image inpaint params

%-----
% On/off flag
%-----
cropNinpaint_inpstruct.flagswitch          = 0;    %[0-off | 1-on]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Training and testing feature matrix and labels parameters

%-----
% Shuffle flag
%-----
% For train and test data
hybrid_inpstruct.flagswitch                = 1;    %[0-off | 1-on]

% Save feature matrix and labels in image info structure (as a variable).
% Else store n samples in sequential .mat files
hybrid_inpstruct.savefeaturematrixNlabels ...
                                                = 1;    %[0-off | 1-on]

%-----
% Training images (data samples) to be considered

```

```

%-----
% full - uses all images in train folders (string)
% half - uses all images in train folders (string)
% n     - n number of images (integer)
hybrid_inpstruct.imagenumbers      = 'full';

% Turn on/off adaptive histogram
hybrid_inpstruct.adaphist          = 'off';

%-----
% Training data folder path
%-----
hybrid_inpstruct.folderpath        = 'E:\Google Drive\Team SewerPipe\Programs\Main\d

%-----
% Provide names of groundtruth folders (include pseudoname even if ground-truth doesn't
% exist for a original folder, e.g. nocrack --> ground_nocrack. Also if the folder
% is empty)
%-----
hybrid_inpstruct.groundtruth_folders = {};

%-----
% Provide names of original data folders (include name of the folder even its folder
% is empty)
%-----
hybrid_inpstruct.originaldata_folders = {};

%-----
% GPU array
%-----
% yes - creates GPU array (note: works for certain Matlab functions)
% no  - non GPU array
hybrid_inpstruct.gpuarray          = 'no';

%-----
% Colorspace segmentation options
%-----
% Type of colorspace to segment RGB ground-truths
% HSV (recommended) or RGB
hybrid_inpstruct.colorspace = 'hsv';  %[hsv | rgb]

% RGB startindex
% n - channel value (integer [0, 255])
hybrid_inpstruct.RGBstartindex     = 235;

%-----
% Anisotropic diffusion parameters
%-----
% Stage I and Stage II
hybrid_inpstruct.aniso.num_iter = [10, 10];
hybrid_inpstruct.aniso.delta_t  = [1/7, 1/7];
hybrid_inpstruct.aniso.kappa    = [5, 5];
hybrid_inpstruct.aniso.option   = [1, 1];

%-----

```

```

% Hessian matrix parameters
%-----
% Frangi filter options
hybrid_inpstruct.frangiopt.FrangiScaleRange = [1 15];
hybrid_inpstruct.frangiopt.FrangiScaleRatio = 1;
hybrid_inpstruct.frangiopt.FrangiBetaOne    = 0.5;
hybrid_inpstruct.frangiopt.FrangiBetaTwo   = 15;
hybrid_inpstruct.frangiopt.BlackWhite      = 1;
hybrid_inpstruct.frangiopt.verbose         = 0;

%-----
% Blob removal parameters
%-----
% Blob filter standard deviation scale
hybrid_inpstruct.blobfilter_sigma         = 0.3;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

function largescaleImInpaint( imSetImgLocation )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

```

```

mask = zeros (181,352);
mask(145:170 , 51:101)    = 255;
mask(145:170 , 132:215)   = 255;
mask(145:170 , 246:318)   = 255;

```

```

h = waitbar(0,'1','Name','Image Inpainting',...
            'CreateCancelBtn',...
            'setappdata(gcf,'canceling',1)');
setappdata(h,'canceling',0)

```

```

for i = 1:length(imSetImgLocation)

```

```

    % Check for Cancel button press
    if getappdata(h,'canceling')
        break
    end
    % Report current estimate in the waitbar's message field
    waitbar(i/length(imSetImgLocation),h,sprintf('Image no.: %i',i))

```

```

    I = imread (cell2mat(imSetImgLocation(i)));
    cropped_img = imcrop (I , [0 60 352 240]);
    inpaintImage = inpaint ( cropped_img , mask );
    imwrite(uint8(inpaintImage),cell2mat(imSetImgLocation(i)))

```

```

end
delete(h)          % DELETE the waitbar; don't try to CLOSE it.

```

```

end

```

```

function inpaintImage = inpaint ( original , mask )

```

```

% Takes an image (with overlaying data) and it's colored version as inputs and generates tl

```

```

% weighted minnum norm reconstruction of the image which is free of the
% overlaying data

% Get size of image
[N1, N2, bytesppix] = size (original);
inpaintImage = zeros([N1, N2, bytesppix]);
h = [1; -1];
L = numel(h);

for itr = 1:bytesppix
    Imsplit_original = original(:, :, itr);

    T1 = @(x) reshape(real(ifft(bsxfun(@times, ...
        fft(reshape(x, [N1, N2]), N1+L-1, 1), ...
        fft(h, N1+L-1, 1)), N1+L-1, 1)), (N1+L-1)*N2, 1);
    S1.type = '()';
    S1.subs = {1:N1, ':'};
    T1h = @(x) reshape(subsref(real(ifft(bsxfun(@times, ...
        fft(reshape(x, [N1+L-1, N2]), N1+L-1, 1), ...
        conj(fft(h, N1+L-1, 1))), N1+L-1, 1)), S1), N1*N2, 1);

    T2 = @(x) reshape(real(ifft(bsxfun(@times, ...
        fft(reshape(x, [N1, N2]), N2+L-1, 2), ...
        fft(h.', N2+L-1, 2)), N2+L-1, 2)), N1*(N2+L-1), 1);
    S2.type = '()';
    S2.subs = {':', 1:N2};
    T2h = @(x) reshape(subsref(real(ifft(bsxfun(@times, ...
        fft(reshape(x, [N1, N2+L-1]), N2+L-1, 2), ...
        conj(fft(h.', N2+L-1, 2))), N2+L-1, 2)), S2), N1*N2, 1);

    S.type = '()';
    S.subs = {find(not(mask))};
    H = @(x) subsref(x, S);
    Hh = @(x) subsasgn(zeros(N1*N2, 1), S, x);

    T = @(x) [T1(x); T2(x)];
    Th = @(x) (T1h(x(1:(N1+L-1)*N2)) + T2h(x((N1+L-1)*N2+1:(N1+L-1)*N2 + ...
        N1*(N2+L-1))));
    xp = Hh(H(Imsplit_original));

    Q.type = '()';
    Q.subs = {find(mask)};
    Z = @(x) subsasgn(zeros(N1*N2, 1), Q, x);
    Zh = @(x) subsref(x, Q);

    c = pcg(@(x) Zh(Th(T(Z(x)))), -Zh(Th(T(xp))), 1e-6, 1000);

    inpaintImage(:, :, itr) = reshape(Z(c)+xp, N1, N2);
end

end

function [ labelArray ] = makeLabelArray( imgSets )
%UNTITLED4 Summary of this function goes here

```

```

% Detailed explanation goes here

imSetTotalNum = sum(cat(1, imgSets.Count));
labelArray    = zeros (imSetTotalNum, 1);
count = 1;
for i = 1:length(imgSets)
    labelArray(count : count + imgSets(i).Count-1,:) = i;
    count = imgSets(i).Count + count;
end

end

function imgPartitionStruct = makePartitions( inpStruct )

%UNTITLED2 Summary of this function goes here
% Detailed explanation goes here

% Create image sets by using Matlab imageset (R2014b onwards)
imgPartitionStruct.trVlts_size      = length(inpStruct.splitRatio) + 1;
imgPartitionStruct.imgSets         = imageSet(inpStruct.imgFolder,'recursive');
imgPartitionStruct.imSetTotalNum    = sum(cat(1, imgPartitionStruct.imgSets.Count));
imgPartitionStruct.imSetImgLoc      = {imgPartitionStruct.imgSets.ImageLocation};
imgPartitionStruct.imSetImgLocFull  = ...
horzcat(imgPartitionStruct.imSetImgLoc{1}, imgPartitionStruct.imSetImgLoc{2}, ...
        imgPartitionStruct.imSetImgLoc{3}, imgPartitionStruct.imSetImgLoc{4}, ...
        imgPartitionStruct.imSetImgLoc{5}, imgPartitionStruct.imSetImgLoc{6}, ...
        imgPartitionStruct.imSetImgLoc{7}, imgPartitionStruct.imSetImgLoc{8}, ...
        imgPartitionStruct.imSetImgLoc{9});

% Make train, validation and test set
[imgPartitionStruct.trainingSets, imgPartitionStruct.validationSets, ...
 imgPartitionStruct.testingSets] = partition(imgPartitionStruct.imgSets, ...
                                             inpStruct.splitRatio, ...
                                             inpStruct.splitShuffleType);

% Save image sets
save ZZZ_imageSetsFull.mat  imgPartitionStruct
end

% function plot3Dbargraph ()
close all; clc

% Plot the 3D bar graph
% load 'ZZZ_imageSetsFull.mat'
imgFull = [imgPartitionStruct.imgSets.Count];
trnFull = [imgPartitionStruct.trainingSets.Count];
valFull = [imgPartitionStruct.validationSets.Count];
tstFull = [imgPartitionStruct.testingSets.Count];

Y = [valFull; tstFull; trnFull; imgFull]';

% Plot 3D graph
str = {'Val'; 'Test'; 'Train'; 'Full'};

```

```

figure
h = bar3(Y);
%     colorbar
%     for k = 1:length(h)
%         zdata = h(k).ZData;
%         h(k).CData = zdata;
%         h(k).FaceColor = 'interp';
%     end
title('Class Distribution')
set(gca, 'PlotBoxAspectRatioMode','auto')
set(gca, 'XTickLabel',str, 'XTick', 1:numel(str), ...
        'YTickLabel',(1:9), 'YTick', (1:9),...
        'FontSize', 18, 'LineWidth', 3)
ylabel ('Class Labels', 'FontSize', 18)
zlabel ('Number of Images', 'FontSize', 18)
legend ('Val', 'Test', 'Train', 'Full')
%     axis tight;

% resize the figure window
%     set(hFig, 'units','normalized','outerposition',[0 0 1 1])

% Export figure to some format
%     export_fig('fig_texture_counts_.pdf',...
%         '-pdf', '-transparent', gcf);

% end

[ Xtrain, Ytrain, Ttrain, ImageShuffIndexMap_train ] ...
    = shuffleFeatMatLabel(featureMatrixTr_BoFOri , labelArrayTr_BoFOri);

[ Xval, Yval, Tval, ImageShuffIndexMap_val ] ...
    = shuffleFeatMatLabel(featureMatrixVl_BoFOri , labelArrayVl_BoFOri);

[ XTest, YTest, TTest, ImageShuffIndexMap_test ] ...
    = shuffleFeatMatLabel(featureMatrixTs_BoFOri , labelArrayTs_BoFOri);

function shuffleNcopy(input)

% Get the ground truth file names
if (isempty(input.folderpath_ground))
    Allfilenames = dir (input.folderpath_original);
else
    Allfilenames = dir (input.folderpath_ground);
end

% Filtered file names
names = setdiff({Allfilenames.name},{'.','..', 'desktop.ini', 'thumbs.db'});

% Sort the files by name
names_sorted = filenamesort(names);

```

```

% Random permutation
ix = randperm(length(names_sorted));
Shuffled_filenames = names_sorted(ix);

% Training data
% From Shuffled_filenames (1) to Shuffled_filenames (N * input.train_percent/100)
train_maxIndex = ceil(length(Shuffled_filenames) * input.trainValTest_percent(1) / 100);

% Condition to check between ground truth or original folder only
if (isempty(input.folderpath_ground) | isempty(input.folderpath_gtrain) | isempty(input.folderpath_gtest))
    % Delete files from a folder(s)
    delete(fullfile(input.folderpath_train, '*.*'));
    delete(fullfile(input.folderpath_test, '*.*'));

    % Copyfile in a loop
    for i = 1:length(Shuffled_filenames)
        if (i <= train_maxIndex)

            % Train set images from original folder
            switch input.filenameRenaming
                case 'actual'
                    copyfile(fullfile(input.folderpath_original, Shuffled_filenames{i}),
                               fullfile(input.folderpath_train, Shuffled_filenames{i}), 'f');
                case 'seqnumb'
                    outputBaseFileName = sprintf('%3.3d.png', i);
                    copyfile(fullfile(input.folderpath_original, Shuffled_filenames{i}),
                               fullfile(input.folderpath_train, outputBaseFileName), 'f');
            end
        else

            % Test set images from original folder
            switch input.filenameRenaming
                case 'actual'
                    copyfile(fullfile(input.folderpath_original, Shuffled_filenames{i}),
                               fullfile(input.folderpath_test, Shuffled_filenames{i}), 'f');
                case 'seqnumb'
                    outputBaseFileName = sprintf('%3.3d.png', i);
                    copyfile(fullfile(input.folderpath_original, Shuffled_filenames{i}),
                               fullfile(input.folderpath_test, outputBaseFileName), 'f');
            end
        end
    end
else
    % Delete files from a folder(s)
    delete(fullfile(input.folderpath_train, '*.*'));
    delete(fullfile(input.folderpath_gtrain, '*.*'));
    delete(fullfile(input.folderpath_test, '*.*'));
    delete(fullfile(input.folderpath_gtest, '*.*'));

    % Copyfile in a loop
    for i = 1:length(Shuffled_filenames)
        if (i <= train_maxIndex)

```

```

% Train set images from original folder
switch input.filenameRenaming
    case 'actual'
        copyfile(fullfile(input.folderpath_original, Shuffled_filenames{i}), .
            fullfile(input.folderpath_train, Shuffled_filenames{i}), 'f');
    case 'seqnumb'
        outputBaseFileName = sprintf('%3.3d.png', i);
        copyfile(fullfile(input.folderpath_original, Shuffled_filenames{i}),
            fullfile(input.folderpath_train, outputBaseFileName), 'f');
end

% Train set images from ground-truth folder
switch input.filenameRenaming
    case 'actual'
        copyfile(fullfile(input.folderpath_ground, Shuffled_filenames{i}), ...
            fullfile(input.folderpath_gtrain, Shuffled_filenames{i}), 'f');
    case 'seqnumb'
        outputBaseFileName = sprintf('%3.3d.png', i);
        copyfile(fullfile(input.folderpath_ground, Shuffled_filenames{i}),
            fullfile(input.folderpath_gtrain, outputBaseFileName), 'f');
end

else

% Test set images from original folder
switch input.filenameRenaming
    case 'actual'
        copyfile(fullfile(input.folderpath_original, Shuffled_filenames{i}), .
            fullfile(input.folderpath_test, Shuffled_filenames{i}), 'f');
    case 'seqnumb'
        outputBaseFileName = sprintf('%3.3d.png', i);
        copyfile(fullfile(input.folderpath_original, Shuffled_filenames{i}),
            fullfile(input.folderpath_test, outputBaseFileName), 'f');
end

% Test set images from ground-truth folder
switch input.filenameRenaming
    case 'actual'
        copyfile(fullfile(input.folderpath_ground, Shuffled_filenames{i}), ...
            fullfile(input.folderpath_gtest, Shuffled_filenames{i}), 'f');
    case 'seqnumb'
        outputBaseFileName = sprintf('%3.3d.png', i);
        copyfile(fullfile(input.folderpath_ground, Shuffled_filenames{i}),
            fullfile(input.folderpath_gtest, outputBaseFileName), 'f');
end

end

end

end

end
end

```



```

function [ Feature_matrix, Labels, Target, indexMap ] = shuffleFeatMatLabel( featMat, labelArray)
%UNTITLED2 Summary of this function goes here
% Detailed explanation goes here

ix = randperm(numel(labelArray));

% Populate the matrices
Feature_matrix = featMat(ix,:);
Labels         = labelArray(ix,:);
Target = full(ind2vec(Labels'))';

% Shuffled index mapping
indexMap = [labelArray, ix'];
end

```