INTELLIGENT UPC-A BARCODE SCANNING USING MACHINE LEARNING

by

Justin H Hawks

B.S., Electrical Engineering, University of Pittsburgh, 2016

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2018

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This thesis was presented

by

Justin H Hawks

It was defended on

February 19, 2018

and approved by

Ervin Sejdić, Ph.D, Associate Professor

Department of Electrical and Computer Engineering

Murat Akcakaya, Ph.D, Assistant Professor

Department of Electrical and Computer Engineering

Samuel Dickerson, Ph.D, Assistant Professor

Department of Electrical and Computer Engineering

Thesis Advisor: Ervin Sejdić, Ph.D, Associate Professor

Department of Electrical and Computer Engineering

INTELLIGENT UPC-A BARCODE SCANNING USING MACHINE LEARNING

Justin H Hawks, M.S.

University of Pittsburgh, 2018

Encoding data in barcodes that can be decoded electronically with barcode scanners is useful in numerous industries and barcodes are used in various aspects of everyday life. This paper proposes a barcode scanner algorithm that unlike conventional barcode scanners, implements a machine learning algorithm in order to decode UPC-A barcodes. The machine learning algorithm makes use of the SURF algorithm for feature point extraction along with K-means clustering to create the visual vocabulary. The classifiers are trained using Support Vector Machines (SVMs) using a Sequential Minimal Optimization (SMO) solver. The proposed algorithm was tested and compared to the results obtained using a conventional barcode scanner algorithm developed in MATLAB, as well as to general distribution test data collected with an experimental setup using imaging scanners. Test results show that the barcode scanner algorithm with the trained image classifiers implemented outperform both the conventional barcode scanner algorithm tested as well as the barcode scanners used in the experimental test when it comes to accuracy. The results clearly depict that machine learning is a valuable tool for the future of barcode scanners. The improved read rates can drastically increase processing times in industrial settings such as delivery services.

Keywords: Barcodes, UPC-A, machine learning, image classifier, speeded-up robust features, support vector machine, sequential minimal optimization, K-means clustering, linear kernel, defects, contrast, decodability, accuracy.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

I would like to thank Dr. Robert Kerestes, Dr. Samuel Dickerson, Dr. Brandon Grainger, and Dr. Zhi-Hong Mao for convincing me to continue on with graduate school at the University of Pittsburgh after completing my undergraduate degree. I would also like to thank Dr. Kara Bocan, Dr. Michael Rothfuss, and Nicholas Franconi for all of their help and expertise throughout the completion of my research. I would like to express my deepest gratitude to my advisor, Dr. Ervin Sejdić, for affording me this incredible opportunity to further my education and his extremely useful guidance throughout the completion of my research. Finally, I'm extremely grateful for the rest of my colleagues, my friends, and especially my family, specifically Daniel and Beatrice Berti, for their support during my masters studies.

1.0 INTRODUCTION

1.1 HISTORY OF BARCODES

Barcodes make use of unambiguous numbers to identify goods, services, assets, and locations by electronically reading them wherever required in business processes [1]. Joe Woodland and Bernard Silver filed a patent in 1949 based on Woodland's idea for a barcode which was inspired by Morse Code [2, 3]. However, it was not until 1974 that Universal Product Codes (UPC's) were first implemented on products in grocery stores due to the fact that the technology needed to scan the barcodes wasn't available at the time the patent was filed [2, 4, 5].

Table 1: Common 1D and 2D symbologies recognized by GS1.

Linear Barcodes (1D)		Two Dimensional Barcodes (2D)
UPC-A	GS1 DataBar Omnidirectional	GS1 DataMatrix
UPC-E	GS1 DataBar Truncated	GS1 QR Code
EAN-8	GS1 DataBar Stacked	PDF417
EAN-13	GS1 DataBar Stacked Omnidirectional	
ITF-14 (Interleaved 2 of 5)	GS1 DataBar Limited	
GS1-128	GS1 DataBar Expanded	
GS1 DataBar	GS1 DataBar Expanded Stacked	

There are a number of different barcode symbologies currently used in the world today, each of which are used for specific applications. Barcodes are broken down into two general categories, linear (1D) and 2D. Amongst these categories there are three primary types of

symbologies which include linear, stacked, and matrix. Table 1 shows common symbologies currently recognized by GS1, which is the company that sets the international standards for barcodes.

Linear symbologies are decoded linearly (in a straight line) and contain no additional data in the vertical height of the barcode [1]. All linear codes use a combination of bars and spaces and the height of the barcode only exists to make the scanning of the barcode easier. The efficiency of linear barcodes is determined by how well a symbology can encode data in the minimal amount of space and some symbologies are more efficient than others [1]. They are able to be scanned by both laser or imaging scanners.

Stacked symbologies use the concept of linear symbologies and look like a series of linear barcodes stacked upon each other. This method allows the efficiency of the code to be increased because they are more space efficient [1]. They are read from top to bottom and can be scanned by both laser or imaging scanners.

Matrix symbologies encode data by turning the pixels (modules) in an image on or off [1]. The pixels are plotted in a matrix where a zero is represented by a white block (off) and a one is represented by a black block (on) [1]. This makes matrix symbologies the most efficient as they can contain a large amount of data in a very small area. However, they are required to be scanned with imaging scanners.

GS1 focuses on the symbologies that are able to be read by scanners that are commercially available and widely implemented throughout the world. Barcodes are scanned in the following three general environments:

- General Distribution
- Point-Of-Sale (POS)
- Point-Of-Care (POC)

The general distribution environment refers to distribution facilities such as UPS and FedEx warehouses. Items intended for general distribution scanning must use a UPC/EAN or GS1 DataBar symbologies [1]. In general, the POS environment usually refers to retail environments, such as grocery stores and department stores, and items in this environment also must use UPC/EAN or GS1 DataBar symbologies [1]. The POC environment typically

refers environments where items such as healthcare items are scanned for direct consumption, and because these products are never scanned at retail POS, 2D symbologies are permitted to be used [1]. Handheld scanners are typically used for POC scanning, however they are also often seen in retail environments. More information on what items fall under each of these environments and the restrictions for each can be found in [1].

Retail products, or any items that are scanned at the POS such as those in grocery stores or department stores, use either UPC-A, UPC-E, EAN-13, and EAN-8 barcodes and the the two and five digit add on symbols [1]. Although each of the mentioned symbologies are similar, UPC-A barcodes are most commonly used in the United States and are the simplest of the symbologies, making them the best candidate for use in this work.

1.2 UPC-A BARCODES

In order to train image classifiers using machine learning, it was imperative to understand in detail how data is encoded in UPC-A barcodes. In accordance with [1] UPC-A barcodes are encoded with 12 characters ranging from 0 to 9. The first 6 characters are from Number Set A and the last 6 characters are from Number Set C. An individual character in a UPC-A barcode is made up 4 elements, or bars, consisting of 2 white bars and 2 black bars. Each element is either 1, 2, 3, or 4 modules in width and a single character is comprised of 7 modules in total. Characters from Number Set A begin with a white element and end with a black element, whereas characters from Number Set C begin with a black element and end with a white element. This is better illustrated in Table 2.

UPC-A barcodes are read from left to right beginning with a left quiet zone, or blank margin, which is used to tell the barcode scanner where the symbol begins so that the scanner doesn't read in information that isn't related to the barcode specifically being read. The left quiet zone is followed by what is defined as a normal guard pattern comprised of three modules, where the first element of the guard bar pattern is a black bar one module wide followed by another white and black bar each also one module wide. Following the normal guard bar pattern are the first 6 characters of the UPC-A barcode from Number Set A as

Table 2: Number Sets A, B, and C

(S denotes a space(light bar), B denotes a bar (dark bar), and the element widths are in modules)

DigitValue	Set A Element Widths				Set B Element Widths				Set C Element Widths			
	S	**B**	**S**	**B**	**S**	**B**	**S**	**B**	**B**	**S**	**B**	**S**
0	3	2	1	1	1	1	2	3	3	2	1	1
1	2	2	2	1	1	2	2	2	2	2	2	1
2	2	1	2	2	2	2	1	2	2	1	2	2
3	1	4	1	1	1	1	4	1	1	4	1	1
4	1	1	3	2	2	3	1	1	1	1	3	2
5	1	2	3	1	1	3	2	1	1	2	3	1
6	1	1	1	4	4	1	1	1	1	1	1	4
7	1	3	1	2	2	1	3	1	1	3	1	2
8	1	2	1	3	3	1	2	1	1	2	1	3
9	3	1	1	2	2	1	1	3	3	1	1	2

previously described. The first six characters encoded from Number Set A are followed by what is defined as a centre guard bar pattern comprised of five modules, where the first element of the guard bar pattern is white bar one module wide, followed alternating black and white bars each one module wide until a total of five modules are reached. This center guard bar pattern serves two purposes. The first purpose being the splitting up the encoded data into two parts and the second purpose being allowing for the second six characters of encoded data which use Number Set C as previously described to begin with a black bar instead of a white bar like those from Number Set A. After the second six characters encoded from Number Set C, another normal guard bar pattern identical to the first normal guard bar pattern follows. Just as the left quiet zone was necessary in order to tell the barcode scanner where the symbol begins, a right quiet zone is required to tell the barcode scanner where the symbol ends and to again ensure the scanner doesn't read in information that isn't related to the specific barcode being scanned.

1.3 RESEARCH OBJECTIVES

Technology has since been the limiting factor in not only the types of barcodes that have been implemented over the years, but also with the performance of the scanners used to scan them [5]. Conventional barcode scanners, both imaging scanners and laser scanners, use analog edge detection in order to detect the dark and light (bars and spaces) areas of a barcode [6, 7, 8, 9]. If the edge detected by the scanner has an edge strength that is above a given threshold, the detected edge is considered to be a detected bar or space [6, 8, 9]. This method can often lead to errors due to many outside factors such as a poorly printed barcode or the distance of the scanner from the barcode based on the scanners limitations [6, 8, 9]. With the advancements in technology-specifically in the area of machine/deep learning-image detection, recognition, and classification have moved to the forefront of research for numerous applications [10, 11, 12, 13, 14].

In this work, two separate barcode scanner algorithms to decode barcodes of the UPC-A symbology were developed. We propose a barcode scanner program that incorporates image classifiers trained using a machine learning algorithm instead of using the edge detection algorithm used in conventional scanners. Two separate image classifiers corresponding to Number Sets A & C shown in Table 2 were trained to differentiate between any value from their respective number set regardless of defects in the images with accuracies above 95%. The classifiers were trained using Support Vector Machines (SVMs) and implemented into a barcode scanner program recognizing that this approach would lead to much longer simulation times. However, investigations into simulation times are beyond the scope of this paper as we focus on the overall accuracy of each scanner program in comparison to each other.

Chapter 2 provides a brief background of work somewhat related to this research as well as an overview of machine learning and the various algorithms that can be used. Chapter 3 outlines the experimental approach taken as well as the data acquisition procedures in detail. This section includes the approach used in the preprocessing of images for classifier training, as well as a detailed account of the training of the image classifiers and the generation of UPC-A barcodes for testing. Chapter 4 presents the results of the image classifier training

and the results acquired after passing the UPC-A barcodes generated for testing through each of the barcode scanner programs. Chapter 5 discusses the results and the implications they may have on the current and future technologies used in industry. Chapter 6 summarizes the work in this paper and discusses the conclusions reached as well as considerations for future work and is followed by a list of references.

2.0 BACKGROUND

2.1 RELATED WORK

Barcode scanner algorithms used by scanner manufacturers are extremely proprietary and due to this fact, writing a program to scan barcodes that accurately represents how an actual barcode scanner functions was difficult. To date, machine learning has been used in barcode detection and recognition applications, but has yet to be used in classification (decoding) applications [15, 16]. Generally, techniques for barcode detection are split up into the following four groups which are based on the specific image processing approaches used in each [15]:

- Morphological Operations
- Image Scanning
- Bottom-Hat Filtering
- Distance Transforms

All techniques listed above are discussed in detail by Katona et al. with the exception of the technique using distance tranforms [17]. Numerous other techniques have been researched including ones that exploit the regular spacing of parallel strips in order to detect both 1D and 2D barcodes as well research that takes advantage of the detection stability of the Maximal Stable Extremal Region (MSER) system [15, 18].

An algorithm for both detecting and decoding a 2D barcode of the Alphabetic symbology is presented in [16]. The proposed algorithm makes use of a two-step strategy which first localizes the word and recovers the grid structure of the symbol and then decodes the symbol using iterative structure based on statistical processing making use of the Dotem structure [16].

2.2 MACHINE LEARNING

Generally, machine learning can be defined as an approach to achieve artificial intelligence making use of systems that can learn from experience to find patterns in a set of data [19]. Instead of programming a computer with specific rules, machine learning teaches a computer to recognize patterns within data [19]. Algorithms (or a sets of rules) learn complex function (patterns) from the data, and based on the patterns learned, make predictions on it [19]. The art of machine learning is accomplished in three general steps [19]:

1. Data is inputed to be used in training the system
2. An algorithm uses this data to learn patterns
3. New data that has not been seen before is then classified for a best guess of what it probably is based on the knowledge gained from the previous step

Machine learning is a powerful tool because it can learn by itself from the data that is passed to it, meaning that a previously trained basic machine learning system can be re-used to learn new patterns without actually re-writing code.

2.2.1 Training a Machine Learning System

Machine learning techniques make use of features (attributes) which can be various properties of the things that the specific system is trying to learn about. It is important to choose helpful input features as they will determine the quality of the machine learning system [19]. However, figuring out the best features to use isn't always clear and trivial. The number of features used by the system determines how many dimensions the system is [19]. Although it is hard for humans to visualize anything higher than 3 dimensions, computers and machine learning systems can, and most machine learning problems have much higher dimensionality. Although adding more dimensions usually helps to separate out the data points which allows the machine learning system to better split the data up for classification, it is possible to use too many dimensions [19]. Using too many dimensions leads to the over-fitting of data, meaning that the system knows the training data very well but wont be able to generalize new information that it hasn't seen before in order to predict the newly inputted

information to the system correctly [19]. One of the biggest challenges is finding enough unbiased training data to use with all of the features selected to train on that is also in a format that can be inputted to a machine learning system depending on the type of machine learning algorithm being used [20]. Data used can be in many forms such as images, tables of data with multiple features, test data, sensor recordings of electrical signals, audio samples, and many others depending on what one specifically is trying to classify [19]. In general, there are three different approaches in training machine learning algorithms [19, 20]. These approaches include unsupervised learning, supervised learning, and reinforcement learning [19, 20]. Each of these is briefly discussed in detail in the following sub-sections.

2.2.1.1 Unsupervised Learning When unsupervised learning is used, the machine learning system must learn from a set of training data that isn't labeled [19, 21]. In other words, the system must be able to realize by itself the distinct differences it sees in the data and be able to categorize them by itself also. The number of features to use might not be known and the features detected might not be distinctly different from one another, making it difficult for the system to correctly classify the data [21]. Clustering techniques are the most common techniques used in unsupervised learning. Clustering algorithms can broken into two broad groups [21]:

- Hard clustering, where each data point only belongs to one cluster
- Soft clustering, where each data point can belong to more than one cluster

If possible data groupings are already known, hard or soft clustering techniques can be used [21]. However, if it's not known how the data may be grouped, self-organizing feature maps or hierarchical clustering can be used to look for possible structures within the data, or cluster evaluation can be used in order to look for the best number of groups for a given clustering algorithm [21]. Common clustering algorithms include k-Means, k-Medoids, Hierarchical Clustering, Self-Organizing Map, Fuzzy c-Means, and Gaussian Mixture Model [21]. While machine learning is an effective method for finding patterns hidden in big datasets, bigger datasets add to the complexity of the problem and reduction of the number of features or dimensionality of the data is often desired to improve the model. The three most commonly used dimensionality reduction techniques are [21]:

- Principle Component Analysis (PCA)
- Factor Analysis
- Nonnegative Matrix Factorization

PCA performs a linear transformation on the data and the majority of the variance or information in the high-dimensional dataset is captured by the first few principle components [21]. PCA uses the redundancy of information in a dataset and generates new variables using linear combinations of the original variables, resulting in a smaller number of new variables that capture the majority of the information [21]. There is no redundant information due to the fact that each principle component is a linear combination of the original variables and they are all orthogonal to each other [21].

Factor analysis provides a representation of the data in terms of a smaller number of unobserved common factors by identifying underlying correlations between variables in the dataset [21]. When datasets contain variables that overlap and depend on one another, factor analysis allows for the for the fitting of a model to the multivariate data in order to estimate the interdependence [21]. The measured variables depend on the small number of unobserved common factors where each is dependent on a linear combination of the common factors [21].

When model terms are required to represent nonnegative quantities, such as physical quantities, nonnegative matrix factorization is used [21]. This technique is based on a low-rank approximation of the feature space which along with reducing the number of features in a model, it also guarantees that all features are nonnegative for models that must represent physical quantities [21].

2.2.1.2 Supervised Learning Supervised learning is currently the most studied area in machine learning [19]. When a machine learning program is provided with training data that is already labeled supervised learning is being used [19, 22]. In other words, the machine learning program is being told how to categorize the training data that is being inputted to it. The machine learning program then uses this data to predict future data inputted that it hasn't seen before [22]. Classification and regression techniques are used to develop the predictive models when supervised learning is used. Classification techniques predict discrete

responses of the system by classifying the input data into categories, whereas regression techniques predict continuous responses by iteratively modeling the relationship between variables of the system [22].

Selecting the correct algorithm to use is a process of trial-and-error and is usually a decision that must be based on the trade-off between specific characteristics of the algorithm such as the training speed, memory usage, predictive accuracy of the trained model on new data, and the transparency or interpretability of the algorithm, which is how easily one can understand the reasons the chosen algorithm makes its predictions [22].

Classification problems can often be broken into binary or multiclass classifications [22]. Binary classification problems are used when a single training or test item can only be divided into two classes, whereas multiclass classification problems can be divided into more than two classes [22]. Multiclass classification problems are generally more challenging since they require a much more complex model [22]. Common classification algorithms include Logistic Regression, k Nearest Neighbor (kNN), Support Vector Machine (SVM), Neural Network, Naïve Bayes, Discriminant Analysis, and Decision Tree. Common regression algorithms include Linear Regression, Nonlinear Regression, Gaussian Process Regression (GPR), SVM Regression, Generalized Linear Model, and Regression Tree [22].

Trained models can often be improved to increase their predictive power and to prevent overfitting. This is accomplished by feature selection, feature transformation, or hyperparameter tuning [22]. Feature selection is the act of identifying the most relevant features or variables to provide the best predictive accuracy in model and is one of the most important tasks to consider when using machine learning [22]. It is especially useful when working with high dimensional data or when a dataset contains a large number of features and a limited number of observations [22]. Reducing the total number of features makes the results obtained easier to understand, saves on memory storage, and reduces computation times [22]. The four most common feature selection techniques are [22]:

- Stepwise Regression
- Sequential Feature Selection
- Regularization
- Neighborhood Component Analysis (NCA)

Stepwise regression sequentially adds or removes features until the highest prediction accuracy is achieved and there in no longer an improvement in it [22]. Sequential feature selection involves adding or removing predictor variables iteratively and evaluating the effect on the performance of the model at each iteration [22]. Regularization makes use of shrinkage estimators to remove any redundant features in the model by reducing their weights (coefficients) to zero [22]. NCA determines the weight that each feature has in the prediction of the output and allows for the features with the lowest weights to be discarded [22].

Feature transformation is the act of turning existing features into new features using techniques such as PCA, factor analysis, and nonnegative matrix factorization previously discussed [22]. Hyperparameter tuning is an iterative process of identifying the set of parameters that yields the best model and controls how a specific machine learning algorithm fits the selected model to the data [22]. The three most common parameter tuning methods are [22]:

- Bayesian Optimization
- Grid Search
- Gradient-Based Optimization

2.2.1.3 Reinforcement Learning When a machine learning system learns by trial-and-error through reward or punishment, reinforcement learning is being used [19]. The machine learning system learns by repeatedly training on the data a large amount of times [19]. When the system correctly classifies the data it is rewarded and the connections to make the decisions it did are strengthened [19]. When the system incorrectly classifies the data it receives no reward, or in other words is punished [19]. Over time, the system will learn to maximize the reward without the need for a human to explicitly tell the system the rules, and can often find ways to classify the data in ways that no human has ever thought of doing [19].

2.2.2 Machine Learning Algorithms

There are a number of supervised and unsupervised machine learning algorithms that each take a different approach to learning, making choosing the right algorithm often overwhelm-

ing. Selecting the correct algorithm largely depends on the size and the type of data being used and the specific application one is using it for [23]. However, even when one does know the type of data being used and its application, finding the best algorithm to use is still very much a trial and error process [23]. Various algorithms for both supervised and unsupervised learning are shown in Table 3 and some of the more common algorithms are briefly described in detail.

Table 3: Supervised and unsupervised learning algorithms used in classification, regression, and clustering.

Supervised Learning		Unsupervised Learning
Classification	Regression	Clustering
Support Vector Machines	Linear Regression	k-Means
Discriminant Analysis	Generalized Linear Model	k-Medoids
Naïve Bayes	Support Vector Regression	Fuzzy c-Means
Nearest Neighbor	Gaussian Process Regression	Hierarchical
	Ensemble Methods	Gaussian Mixture
	Decision Trees	Neural Networks
	Neural Networks	Hidden Markov Model

2.2.2.1 k-Means The k-Means algorithm partitions data into a k number of mutually exclusive clusters and how well a point fits into a cluster is determined by the distance from that point to the center of the cluster [21]. The algorithm is best used when the number of clusters is known and when the fast clustering of large data sets is desired [21].

2.2.2.2 k-Medoids The k-Medoids algorithm is very similar to the k-Means algorithm, but add the additional requirement that the centers of the clusters coincide with points in the data [21]. Similar to the k-Means algorithm, the algorithm is best used when the number of clusters is known, when the fast clustering of categorical data is desired, and when there is the need to scale large data sets [21].

2.2.2.3 Hierarchical Clustering

Hierarchical clustering algorithms produce sets of clusters by analyzing similarities between pairs of points within the data and grouping objects into a binary hierarchical tree [21]. The algorithm is best used when the number of clusters in the data isn't known in advance or if a visualization to guide one in the selection process is desired [21].

2.2.2.4 Self-Organizing Map

The self-organizing maps algorithm is a neural-network based clustering algorithm that is used to transform a dataset into a topology-preserving 2D map [21]. It is best used when there is the need to visualize high-dimensional data in 2D or 3D or when one needs to deduce the dimensionality of the data by preserving its topology [21].

2.2.2.5 Fuzzy c-Means

The Fuzzy c-Means algorithm is a partitioned based clustering algorithm that is most often used when data points may belong to more than one cluster [21]. The algorithm is best used when the number of clusters is known but they overlap and for pattern recognition applications [21].

2.2.2.6 Gaussian Mixture Model

Similar to the Fuzzy c-Means algorithm, the Gaussian Mixture Model algorithm is also a partition-based clustering algorithm, but differs due to data points coming from different multivariate and normal distributions with certain probabilities [21]. The algorithm is best used in similar circumstances as the Fuzzy c-Means algorithm such as when a data point might belong to more than one cluster, but differs such that it is also used when clusters have different sizes and correlation structures within them [21].

2.2.2.7 Logistic Regression

Logistic regression algorithms fit models that can predict the probability of a binary response belonging to one class or another and are very simple, making them a common starting point for binary classification problems [22]. The algorithm is best used when the data can be separated by a single linear boundary or when one needs a baseline for evaluating more complex classification methods [22].

2.2.2.8 k Nearest Neighbor

The kNN algorithm classifies objects in a dataset based on the classes of their nearest neighbors based on the assumption that objects near each other are similar [22]. Various distance metrics can be used to find the nearest neighbor such as Euclidean, city block, cosine, and Chebychev [22]. The algorithm is best used when there is a need for a simple algorithm that can establish benchmark learning rules [22]. However, due to the complexity of the algorithm, memory usage and prediction speed of the trained model have to be of lesser concern when considering using a kNN algorithm [22].

2.2.2.9 Support Vector Machine

SVM algorithms are used to classify data by finding the linear decision boundary (hyperplane) that separates all the data points from one class from those of another class [22]. When the data is linearly separable, the best hyperplane for an SVM is the one that yields the largest margin between the two classes of data [22]. If the data is not linearly separable, a loss function is used which penalizes data points that fall on the wrong side of the hyperplane [22]. Kernel transforms can also be used with SVMs to transform nonlinear separable data into higher dimensions in order to find a linear boundary to separate the data [22]. SVMs are best used for the classification of data that has exactly two classes, although they can be used for multiclass classification if error correcting techniques are used [22]. The main advantage to SVMs is that they are simple classifiers that are easy to interpret and accurate [22].

2.2.2.10 Neural Network

Neural networks are made up of highly connected networks of neurons that relate the inputs to the desired outputs and are inspired by the human brain [22]. They are trained by iteratively modifying the strengths of the connections so that given inputs map to the correct response [22]. They are best used when one must model a highly nonlinear system, when data is available incrementally and one wishes to constantly update the model, and when there could be unexpected changes in the input data [22]. Due to the complexity of these models, the interpretability of the model should not be a key concern if used [22].

2.2.2.11 Naïve Bayes The Naïve Bayes classifier algorithm is based on the Bayes algorithm and assumes that the presence of a particular feature in a class is unrelated to the presence of an other feature it sees in the same class [22]. New data is classified based on the highest probability of its belonging to a particular class [22]. The algorithm is best used on small datasets that contain many parameters, when a classifier that is extremely easy to interpret is desired, and when the model may encounter scenarios not seen in the training data [22].

2.2.2.12 Discriminant Analysis Discriminant analysis algorithms classify data by finding linear combinations of features in the data assuming that different classes generate data based on Gaussian distributions [22]. In order to train a discriminant analysis model, parameters for a Gaussian distribution for each class must be found and are then used to calculate boundaries to determine the class of new data which can be linear or quadratic functions [22]. The algorithm is best used when a simple and easy to interpret model is desired and when a model that is fast to predict and uses very little memory during training is also desired [22].

2.2.2.13 Decision Tree Decision trees predict responses to data following various decisions made in the tree from the root (beginning) down to a leaf node [22]. Branching conditions are used where the value of a predictor is compared to a trained weight [22]. The number of branches and values of the weights are determined in the training process [22]. The model can be further simplified by by pruning the model if desired [22]. The algorithm is best used when a fast fitting model that is very easy to interpret is desired, when one wishes to minimize memory usage, and when one isn't concerned with obtaining a very high predictive accuracy [22].

2.2.2.14 Linear Regression Linear regression models are both simple to interpret and to train, making them a great choice as a first model to fit to a new dataset [22]. Linear regression is a statistical modeling technique which is used to describe a continuous response variable as a linear function of one or more predictor variables [22]. The linear regression

algorithm is best used when a fast fitting model that is very easy to interpret is desired, as well as for use as a baseline model on a dataset when evaluating more complex models to aid in determining the best model to use [22].

2.2.2.15 Nonlinear Regression Similar to linear regression, nonlinear regression is also a statistical modeling technique but instead it is used to help describe nonlinear relationships within experimental data [22]. The algorithm is best used when the dataset has very strong nonlinear trends and can't easily be transformed into a linear space [22].

2.2.2.16 Gaussian Process Regression GPR models are used for predicting the value of a continuous response variable and are nonparametric [22]. These models are best used in the field of spatial analysis for interpolating spatial data in the presence of uncertainty or as a surrogate model in order to facilitate the optimization of complex designs [22].

2.2.2.17 SVM Regression SVM regression models work just like SVM classification models previously discussed, but are modified in order to be able to predict a continuous response [22]. Unlike the SVM classification models that find a hyperplane that separates the data, SVM regression algorithms find a model that deviates from the measured by a very small value [22]. These algorithms are best used for very high-dimensional datasets [22].

2.2.2.18 Generalized Linear Model A generalized linear model is a special case of a nonlinear model that makes uses of linear methods by fitting a linear combination of the inputs to a nonlinear function of the outputs [22]. The algorithm is best used when the variables in the dataset have a non-normal distribution [22].

2.2.2.19 Regression Tree Regression tree algorithms are very similar to the decision trees previously discussed for classification, but are modified to be able to predict continuous responses [22]. They are best used when predictors are categorical (discrete) or behave nonlinearly [22].

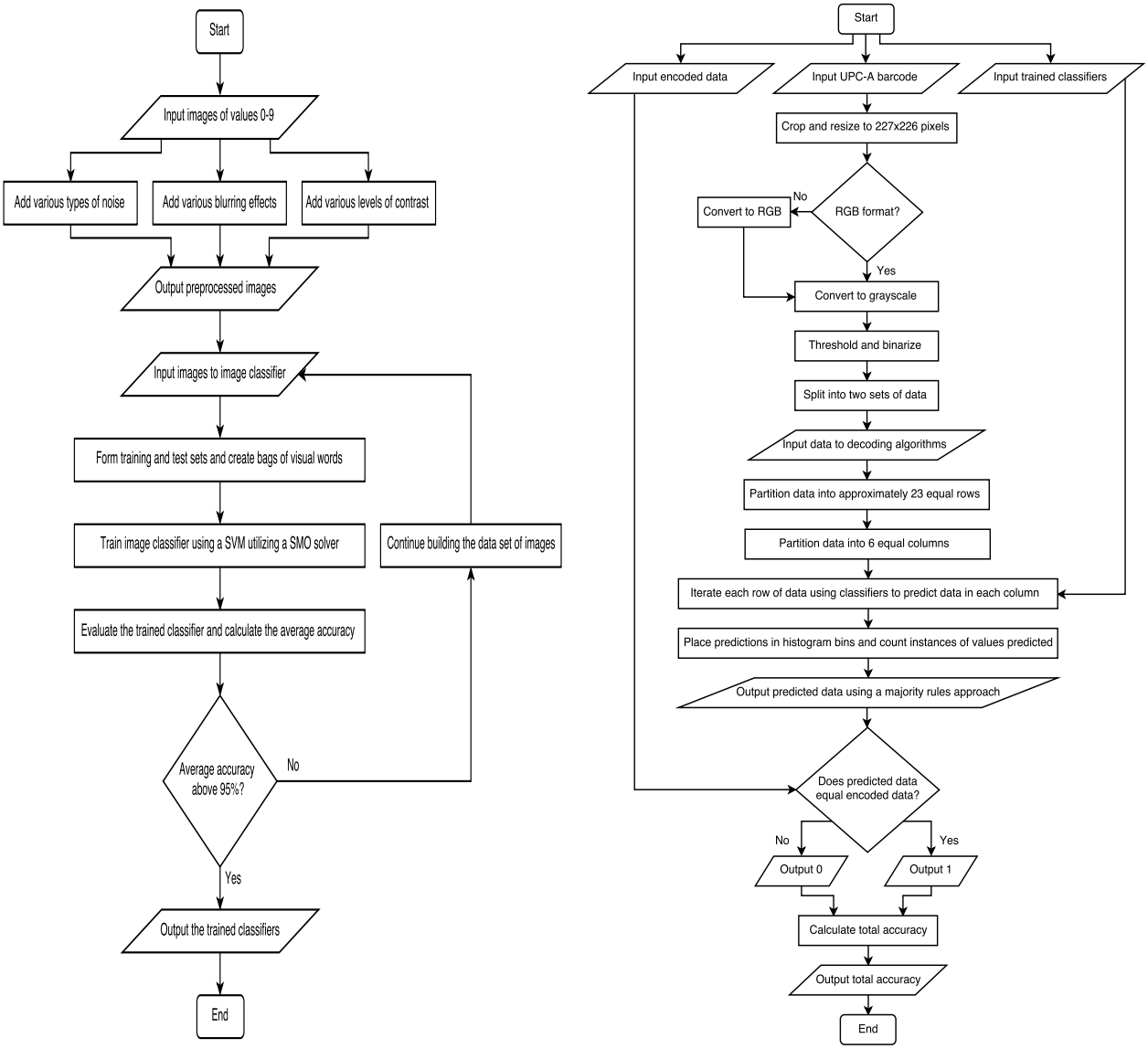
3.0 METHODOLOGY

The following sub-sections provide a detailed overview of how our barcode scanner algorithms were developed and implemented in a barcode scanner program. First, a detailed overview of the image preprocessing required is given, followed by detailed overviews of classifier training, test code generation, and the implementation of the algorithms into barcode scanner programs. Flow charts of the image classifier training process as well as the barcode scanner algorithm are provided in Fig. 1a and Fig. 1b respectively, for a high-level overview of the work presented in this paper.

Unless otherwise specified, all simulations and results in this paper were obtained using MATLAB R2017a on an academic license with access to all available tool boxes. A Dell Precision Tower 5810 XCTO Base workstation was used running the 64-bit Microsoft Windows 10 Professional OS with an Intel(R) Xeon(R) CPU E5-1620 v4 @ 3.5GHz x64-based processor, 16GB (4x4GB) 2400MHz DDR4 RDIMM ECC of installed RAM, a 2.5" 512GB SATA Class 20 Solid State Drive, an integrated Intel AHCI chipset SATA controller (6x6.0Gb/s)-SW RAID 0/1/5/10, and an NVIDIA Quadro K620 2GB (DP, DL-DVI-I) (1 DP to SL-DVI adapter) video card.

3.0.1 Preprocessing Images for Classifier Training

UPC-A barcodes were generated that were of perfect quality and contained no defects so that at least one instance of each number 0-9 from both Number Set A and Number Set C were present using Barcode Studio (TEC-IT Barcode Studio Version 15.5.1.21844) along with the values from Table 2. Paint.NET (dotPDN LLC and Rick Brewster Paint.NET Version 4.0.17) was used to crop an image of each value from each number set respectively. Numerous



(a) Classifier training process flow chart.

(b) Barcode scanner algorithm flow chart.

Figure 1: Algorithm flow charts.

image processing and photo effects were applied to the cropped images of each value such as various types of blurs, distortions, and noises, as well as adjustments to the brightness and contrast of each image. MATLAB was also used to add Gaussian, Poisson, Salt & Pepper, and Speckle noise to each of the original images with various noise parameters.

Assuming that noise in an image at each pixel is independent, the mean $\bar{I}$ and standard deviation σ for each pixel are given by Eqn. 3.1 & Eqn. 3.2 respectively for each $i, j = 0, ...N-1$ [24].

$$\bar{I}(i,j) = \frac{1}{N}\sum_{k=0}^{N-1} I_k(i,j) \tag{3.1}$$

$$\sigma(i,j) = \left(\frac{1}{N-1}\sum_{k=0}^{N-1}(I_k(i,j) - \bar{I}(i,j))^2\right)^{1/2} \tag{3.2}$$

Since noise in neighboring pixels of an image is not independent in reality, the auto-covariance of the noise in the image can be assumed to be the same everywhere in the image and we can compute the auto-covariance in Eqn. 3.3

$$\begin{aligned} C_{II}(i',j') = \frac{1}{N^2}\sum_{i=0}^{N_{i'}}\sum_{j=0}^{N_{j'}}(I_k(i,j) - \bar{I}(i,j))(I_k(i+i', j+j') \\ - \bar{I}_k(i+i', j+j')) \end{aligned} \tag{3.3}$$

letting $N_{i'} = N - i' - 1$ and $N_{j'} = N - j' - 1$ for each $i', j' = 0, ..., N-1$ [24].

Random noise $n(i,j)$ can be added to an image pixel value $I(i,j)$ using the additive noise model shown in Eqn. 3.4 [24].

$$\hat{I}(i,j) = I(i,j) + n(i,j) \tag{3.4}$$

Gaussian noise was added to each image with default mean and variance of zero and 0.1 respectively, as well as with a mean of 1 and variance of 0.5, mean of 0.5 and variance of 0.5, mean of 1 and variance of 1, and mean of 0.5 and variance of 1 using the distribution shown in Eqn. 3.5 [24]

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \tag{3.5}$$

where the variance is equal to the standard deviation squared (σ^2).

Poisson noise was added to each image using the input pixel values without scaling them The pixels generated from a Poisson distribution had a mean equal to the pixel value prior to applying the probability distribution function in Eqn. 3.6 [25].

$$p(I(i,j)) = \frac{\bar{I}(i,j)^{I(i,j)} e^{-\bar{I}(i,j)}}{I(i,j)!} \tag{3.6}$$

Salt & Pepper noise was added to each image with a default noise density of 0.05 based on Eqn. 3.7

$$I_{sp}(i,j) = \begin{cases} I(i,j) & x < l \\ I_{min} + y(I_{max} - I_{min}) & x \geq l \end{cases} \tag{3.7}$$

where $x, y \in [0,1]$ are two uniformly distributed random variables [24].

Speckle (multiplicative) noise was also added to each image based on Eqn. 3.8

$$J = I + n * I \tag{3.8}$$

where I is the original image, J is the output image with noise added, and n is uniformly distributed random noise with zero mean and variance of 0.05.

Random values of mean, variance, and noise density were generated to randomly add Gaussian, Salt & Pepper, and Speckle noise to the original images. The Image Batch Processor application available in the MATLAB Image Processing and Computer Vision Toolbox, was used to convert each of the images for each value from a color (RGB) image to a gray scale image as well as binary instances of each of the images.

Random instances of black and white color were added to the originally cropped images based on the original image size. The cropped images were 88 pixels in height and 14 pixels in width. Random integers from the first and second half of both the width and height of the image were calculated and used to randomly select the location of the area to be changed to either black or white by setting the value of the image in that area equal to 0, 1, or 255 depending on the variable type of the image. A total of 4,000 images of each value for each number set were created to be used for classifier training.

3.0.2 Classifier Training

Image category classifiers for each respective number set were trained as their own independent classifiers. A datastore, or a repository for collections of data that are too large to fit in memory, of the image data for each number set was created using the names of the sub-folders as the label names for the classifier. The data set was randomly split into training and test data using 50% of the images for the training data and 50% of the images for test data.

A bag of visual words object was created using the set of data randomly selected for training over all ten image categories in each number set for values 0 - 9 [26]. Feature point locations for the images were selected using the grid method, where the grid step was [8 8] and the block width was [32 64 96 128] [27]. Speeded-Up Robust Features (SURFs) were extracted from the selected feature point locations and 80% of the strongest features extracted were kept from each category in order to improve clustering as the SURF algorithm consists of both feature detection and representation aspects based on the Hessian matrix, and uses the concept of integral images to quickly compute box type convolution filters [26, 28, 29, 30, 31]. The sum of all pixels in the input image I within some rectangular region containing the origin and x can be represented by the entry of the integral image $I_\Sigma(x)$ at any location $x = (x, y)^T$ as shown in Eqn. 3.9 [32, 31].

$$I_\Sigma(x) = \sum_{i=0}^{i<x} \sum_{j=0}^{j<y} I(i, j) \tag{3.9}$$

The SURF algorithm and the use of integral images is very helpful because the calculation time is independent of the size of the image allowing for the use of large filters [32, 31]. The Hessian matrix is used for detection because of how accurately it performs. For any given point $x = (x, y)$ found in an image I, the Hessian matrix in x at a scale of σ is shown in Eqn. 3.10

$$\mathcal{H}(x, \sigma) = \begin{bmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{xy}(x, \sigma) & L_{yy}(x, \sigma) \end{bmatrix} \tag{3.10}$$

where $L_{xx}(x,\sigma)$, $L_{xy}(x,\sigma)$, and $L_{yy}(x,\sigma)$ are the convolution of the Gaussian second order derivitive $\frac{\partial^2}{\partial x^2}g(\sigma)$ with the image I in point x [32]. The algorithm searches for points where the determinant of the defined Hessian matrix has a local maxima based on Eqn. 3.11.

$$det(\mathcal{H}_approx) = D_{xx}D_{yy} - (wD_{xy})^2 \tag{3.11}$$

The relative weight of the filter responses w defined in Eqn. 3.12 is used in order to balance the determinant expression in Eqn. 3.11 due to the energy conservation between the Gaussian and approximated Gaussian kernels using the Frobenius norm $|x|_F$ and a n x n filter size [32].

$$w = \frac{|L_{xy}(\sigma)|_F|D_{yy}(n)|_F}{|L_{yy}(\sigma)|_F|D_{xy}(n)|_F} \tag{3.12}$$

The number of features from each category were automatically balanced by determining which image category had the least number of strongest features and only keeping the same number of the strongest features from each of the other image categories [26]. K-Means clustering was used to create a visual vocabulary using the extracted features due to the many advantages the algorithm provides such as that it is simple and fast, highly efficient and flexible, yields respectable results for convex clusters, and provides good geometrical and statistical meaning [26, 33, 34]. The following equations were used for the local-maximum-likelihood estimates $\hat{\boldsymbol{\mu}}_{\boldsymbol{i}}$, $\hat{\boldsymbol{\Sigma}}_{\boldsymbol{i}}$, and $\hat{P}(\omega_i)$ [35]:

$$\hat{P}(\omega_i) = \frac{1}{n}\sum_{k=1}^{n}\hat{P}(\omega_i|\mathbf{x}_k,\hat{\boldsymbol{\theta}}) \tag{3.13}$$

$$\hat{\boldsymbol{\mu}}_{\boldsymbol{i}} = \frac{\sum_{k=1}^{n}\hat{P}(\omega_i|\mathbf{x}_k,\hat{\boldsymbol{\theta}})\mathbf{x}_k}{\sum_{k=1}^{n}\hat{P}(\omega_i|\mathbf{x}_k,\hat{\boldsymbol{\theta}})} \tag{3.14}$$

$$\hat{\boldsymbol{\Sigma}}_{\boldsymbol{i}} = \frac{\sum_{k=1}^{n}\hat{P}(\omega_i|\mathbf{x}_k,\hat{\boldsymbol{\theta}})(\mathbf{x}_k-\boldsymbol{\mu}_{\boldsymbol{i}})(\mathbf{x}_k-\boldsymbol{\mu}_{\boldsymbol{i}})^t}{\sum_{k=1}^{n}\hat{P}(\omega_i|\mathbf{x}_k,\hat{\boldsymbol{\theta}})} \tag{3.15}$$

where

$$\hat{P}(\omega_i|\mathbf{x}_k,\hat{\boldsymbol{\theta}}) = \frac{|\hat{\boldsymbol{\Sigma}}_{\boldsymbol{i}}|^{-1/2}exp[-\frac{1}{2}(\mathbf{x}_k-\boldsymbol{\mu}_{\boldsymbol{i}})^t\hat{\boldsymbol{\Sigma}}_{\boldsymbol{i}}{}^{-1}(\mathbf{x}_k-\boldsymbol{\mu}_{\boldsymbol{i}})]\hat{P}(\omega_i)}{\sum_{j=1}^{c}|\hat{\boldsymbol{\Sigma}}_{\boldsymbol{j}}|^{-1/2}exp[-\frac{1}{2}(\mathbf{x}_k-\boldsymbol{\mu}_{\boldsymbol{j}})^t\hat{\boldsymbol{\Sigma}}_{\boldsymbol{j}}{}^{-1}(\mathbf{x}_k-\boldsymbol{\mu}_{\boldsymbol{j}})]\hat{P}(\omega_j)} \tag{3.16}$$

The algorithm computes the squared Euclidean distance $||\mathbf{x}_k - \hat{\boldsymbol{\mu}}_{\boldsymbol{i}}||^2$ and finds the mean $\hat{\boldsymbol{\mu}}_{\boldsymbol{m}}$ nearest to $\mathbf{x}_k$ and then approximates $\hat{P}(\omega_i|\mathbf{x}_k, \hat{\boldsymbol{\theta}})$ as shown in Eqn. 3.17 and uses this approximation and iterating Eqn. 3.14, the algorithm then finds $\hat{\boldsymbol{\mu}}_{\mathbf{1}}, ..., \hat{\boldsymbol{\mu}}_{\mathbf{c}}$ [35, 34].

$$\hat{P}(\omega_i|\mathbf{x}_k, \hat{\boldsymbol{\theta}}) \approx \begin{cases} 1 & \text{if } i = m \\ 0 & \text{otherwise.} \end{cases} \tag{3.17}$$

The image classifiers were trained using a Support Vector Machine (SVM). In general, the main task when training SVMs is to solve the quadratic optimization problem when given instances $\mathbf{x}_i$, $i = 1, ..., l$ with labels $y_i \in 1, -1$ shown in Eqn. 3.18

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\text{minimize}} \quad & f(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \mathbf{e}^T \boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C, i = 1, ..., l, \\ & \mathbf{y}^T \boldsymbol{\alpha} = 0 \end{aligned} \tag{3.18}$$

where $\mathbf{e}$ is the vector of all ones, C is the upper bound of all variables, Q is an l by l symmetric matrix with $Q_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, and $K(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel function [36].

The SVM classifier finds a hyperplane to separate the two-class data with maximal margin which is defined as the distance from the separating hyperplane to the closest training point [26, 10, 37, 38]. Thus, this hyperplane results in minimizing the risk of misclassification of test data [39, 40]. In our case specifically, the classification function is found for given observations $\mathbf{X}$ and corresponding labels $\mathbf{Y}$ shown in Eqn. 3.19

$$\mathbf{f}(\mathbf{x}) = \mathbf{sign}(\mathbf{w^T x} + \mathbf{b}) \tag{3.19}$$

where $\mathbf{w}$ and $\mathbf{b}$ are parameters of the hyperplane [26]. Due to the fact that data sets are not always linearly separable, a mapping function ϕ is made using the original data space of $\mathbf{X}$ that is used to map the data to another feature space which can have an infinite dimension and the kernel function is introduced as shown in Eqn. 3.20 which can be used to express the decision function shown in Eqn. 3.21

$$\mathbf{K}(\mathbf{u}, \mathbf{v}) = \boldsymbol{\phi}(\mathbf{u}) \cdot \boldsymbol{\phi}(\mathbf{v}) \tag{3.20}$$

$$\mathbf{f(x) = sign}(\sum_{\mathbf{i}} \mathbf{y_i}\alpha_i \mathbf{K(x, x_i)} + \mathbf{b}) \tag{3.21}$$

where $\mathbf{x_i}$ are the training features from the data space $\mathbf{X}$, $\mathbf{y_i}$ is the label of $\mathbf{x_i}$, and the parameter α_i are equal to zero for most i [26]. The number of occurrences of each keypoint $\mathbf{v_i}$ for the vocabulary $\mathbf{V}$ in each image $\mathbf{I_i}$ are used to form binned histograms which are used as the input features $\mathbf{x_i}$ to the SVM [26].

A Sequential Minimal Optimization (SMO) solver was used which is an extreme case and restricts the subset of the vector being optimized to only having two elements [36, 41, 42] using the following algorithm [36]:

1. Find $\boldsymbol{\alpha}^1$ as the initial feasible solution. Set $k = 1$.
2. If $\boldsymbol{\alpha}^k$ is an optimal solution of (1), stop. Otherwise, find a *two-element* working set $B = i, j \subset 1, ..., l$. Define $N \equiv 1, ..., l$ B and $\boldsymbol{\alpha}_B^k$ and $\boldsymbol{\alpha}_N^k$ to be sub-vectors of $\boldsymbol{\alpha}^k$ corresponding to B and N, respectively.
3. Solve the following sub-problem with the variable $\boldsymbol{\alpha}_B$:

$$\begin{aligned}
\underset{\boldsymbol{\alpha}_B}{\text{minimize}} \quad & \frac{1}{2}\begin{bmatrix}\boldsymbol{\alpha}_B^T & (\boldsymbol{\alpha}_N^k)^T\end{bmatrix}\begin{bmatrix}Q_{BB} & Q_{BN}\\ Q_{NB} & Q_{NN}\end{bmatrix}\begin{bmatrix}\boldsymbol{\alpha}_B\\ \boldsymbol{\alpha}_N^k\end{bmatrix} - \begin{bmatrix}\mathbf{e}_B^T & \mathbf{e}_N^T\end{bmatrix}\begin{bmatrix}\boldsymbol{\alpha}_B\\ \boldsymbol{\alpha}_N^k\end{bmatrix} \\
& = \frac{1}{2}\boldsymbol{\alpha}_B^T Q_{BB}\boldsymbol{\alpha}_B + (-\mathbf{e}_B + Q_{BN}\boldsymbol{\alpha}_N^k)^T\boldsymbol{\alpha}_B + \text{constant} \\
& = \frac{1}{2}\begin{bmatrix}\alpha_i & \alpha_j\end{bmatrix}\begin{bmatrix}Q_{ii} & Q_{ij}\\ Q_{ij} & Q_{jj}\end{bmatrix}\begin{bmatrix}\alpha_i\\ \alpha_j\end{bmatrix} + (-\mathbf{e}_B + Q_{BN}\boldsymbol{\alpha}_N^k)^T\begin{bmatrix}\alpha_i\\ \alpha_j\end{bmatrix} + \text{constant} \\
\text{subject to} \quad & 0 \leq \alpha_i, \alpha_j \leq C, \\
& y_i\alpha_i + y_j\alpha_j = -\mathbf{y}_N^T\boldsymbol{\alpha}_N^k
\end{aligned}$$

   where $\begin{bmatrix}Q_{BB} & Q_{BN}\\ Q_{NB} & Q_{NN}\end{bmatrix}$ is a permutation of the matrix Q.

4. Set $\boldsymbol{\alpha}_B^{k+1}$ to be the optimal solution of 3.0.2 and $\boldsymbol{\alpha}_N^{k+1} \equiv \boldsymbol{\alpha}_N^k$. Set $k \leftarrow k + 1$ and go to Step 2.

Due to the simple nature of the classifier being trained and the fact that it is a two-variable problem, only a few components were required to be updated at each iteration and

therefore there was no need to worry about slow convergence times making this the preferable optimization routine to use [36]. Using a linear kernel, the elements of the Gram matrix were computed using Eqn. 3.22

$$G(x_j, x_k) = x_j' x_k \tag{3.22}$$

where the Gram matrix of a set of n vectors $\{x_1, ..., x_n; x_j \in R^P\}$ is an n-by-n matrix with element (j, k) defined as $G(x_j, x_k) = \langle \phi(x_j), \phi(x_k) \rangle$ where $G(x_j, x_k)$ is an inner product of the transformed predictors using the kernel function ϕ. In an effort to prevent overfitting, a box constraint value of 1.1 was used based on Eqn. 3.23

$$C_j = nC_0 w_j^* \tag{3.23}$$

where C_j is the box constraint of observation j, n is the training sample size, C_0 is the initial box constraint, and w_j^* is the total weight of observation j in order to control the maximum penalty imposed on margin-violating observations. Increasing this value caused the SVM classifier to assign fewer support vectors and did not significantly increase the training time of the classifiers.

Upon completion of the classifier training, an image category classifier containing the number of categories and category labels for the inputs from the image data store previously created by encoding features for the total number of images in the training data from each image category was returned for each trained classifier [26]. The trained classifiers were evaluated using the test data previously defined to obtain the confusion matrix for each of the classifiers based on Eqn. 3.24

$$M_{ij} = \frac{|\{\mathbf{I_k} \in \mathbf{C_j} : \mathbf{h(I_k) = i}\}|}{|\mathbf{C_j}|} \tag{3.24}$$

where $\mathbf{i}, \mathbf{j} \in \mathbf{1}, ..., \mathbf{N_c}, \mathbf{C_j}$ is the set of test images from category $\mathbf{j}$ and $\mathbf{h(I_k)}$ is the category which obtained the highest classifier output for image I_k. The overall error rate was calculated using Eqn. 3.25.

$$R = \mathbf{1} - \frac{\sum_{\mathbf{j=1}}^{\mathbf{N_c}} |\mathbf{C_j}| M_{jj}}{\sum_{\mathbf{j=1}}^{\mathbf{N_c}} |\mathbf{C_j}|} \tag{3.25}$$

The row indices of the matrices correspond to the known labels and the columns correspond to the predicted labels [26]. The average accuracy of the trained classifiers was calculated by taking the mean of the diagonal elements of the obtained confusion matrices. The classifiers were tested by categorizing new images not contained in the test or training data by computing the K-step ahead output of the trained classifiers using the new images and 10-fold cross validation [26].

3.0.3 Test Code Generation, Adding Defects, Contrast, & Noise

A set of 10,000 UPC-A barcodes was generated for testing using the Barcode Studio program where each barcode had a module width of 2 pixels, was 120 pixels in width, 226 pixels in height, and had a resolution of 96 dpi. Black and white defects were added to the generated codes in order to represent smudges, covering, or tearing of a barcode label. Using the calculated size of each image, a section of each barcode was randomly chosen where all the pixels in the randomly chosen section were set to be either black or white.

Test data from an experimental general distribution test set up using imaging scanners used to simulate the scanning of barcodes in distribution facilities, such as UPS and FedEx warehouses, are used to determine what could cause even the best barcode scanners on the market to struggle with accurately decoding the encoded data. The test set up uses a motor mounted to a base that spins a wheel holding up to six symbols printed on 6" x 6" substrates at a time, at speeds of 1 ft/s, 5 ft/s, and 10 ft/s. Two imaging scanners are positioned underneath the wheel to read each of the symbols as they pass by at the various speeds. Scanner specifications can be seen in Table 4, where Scanners A and B are a Datalogic Matrix 300 and a Datalogic Matrix 400 respectively, and details about the barcodes used in the testing are shown in Table 5. Analyzing the test data conducted on a set of 16 UPC-A barcodes with both imaging scanners, it was determined that low contrast and decodability grades as per [43] were the largest factors affecting the read rates of each of the scanners. Sixteen barcodes containing the same encoded data as the barcodes used in the experimental general distribution testing were created using the Barcode Studio program. In order to achieve low contrast grades barcodes are printed on different colors substrates as

shown in Fig. 2. The colors shown in Figures 2a and 2b have RGB values of (175,212,181) and (118,176,177) respectively. These colors were added to each of the created barcodes corresponding to the colors used for each barcode in the experimental testing using the Paint.NET program.

Table 4: Scanner specifications used in the experimental general distribution testing.

Scanner	Scanner Type	Resolution	Sensor	Frame Rate
A	Imager	1280 x 1024 (1.3 Megapixels)	CMOS(complementary metal-oxide semiconductor)	60 frames/s
B	Imager	1600 x 1200 (1.9 Megapixels)	CCD (charge-coupled device)	15 frames/s

Table 5: Table of images generated to replicate the barcodes used in the experimental testing including the printer used, type and color of substrate used, and all parameters applied to the images, where the overall grade, contrast, and decodability are based on standards found in [43].

Image	Printer	Substrate	Encoded Data	Height (mm)	Width (mm)	Aspect Ratio	X-Dimension	Grade	Contrast	Decodability
1.png	Xerox WorkCentre 7535 @ 1200 dpi	Kraft 45 (Green)	061414156004	31.01	42.375	1	0.375	C	2.00	2.00
2.png	Xerox WorkCentre 7535 @ 1200 dpi	Kraft 45 (Green)	061414156011	31.01	42.375	1	0.375	F	2.00	0.00
3.png	Xerox WorkCentre 7535 @ 1200 dpi	Kraft -30 (Turquoise)	061414156028	31.01	42.375	1	0.375	D	1.00	2.00
4.png	Xerox WorkCentre 7535 @ 1200 dpi	Kraft -30 (Turquoise)	061414156035	31.01	42.375	1	0.375	F	1.00	0.00
5.png	Xerox WorkCentre 7535 @ 1200 dpi	Kraft -30 (Turquoise)	061414156042	34.73	56.5	1	0.500	C	1.00	2.00
6.png	Xerox WorkCentre 7535 @ 1200 dpi	Kraft 45 (Green)	061414156059	34.73	56.5	1	0.500	D	2.00	1.00
7.png	Xerox WorkCentre 7535 @ 1200 dpi	Kraft -30 (Turquoise)	061414156066	34.73	56.5	1	0.500	D	1.00	2.00
8.png	Xerox WorkCentre 7535 @ 1200 dpi	Kraft -30 (Turquoise)	061414156073	34.73	56.5	1	0.500	F	1.00	1.00
9.png	Xerox WorkCentre 7535 @ 1200 dpi	Kraft 45 (Green)	700090418319	31.01	42.375	1	0.375	D	2.00	1.00
10.png	Xerox WorkCentre 7535 @ 1200 dpi	Kraft 45 (Green)	701191469835	31.01	42.375	1	0.375	F	2.00	0.00
11.png	Xerox WorkCentre 7535 @ 1200 dpi	Kraft -30 (Turquoise)	703291603805	31.01	42.375	1	0.375	F	1.00	2.00
12.png	Xerox WorkCentre 7535 @ 1200 dpi	Kraft -30 (Turquoise)	705990602202	31.01	42.375	1	0.375	F	1.00	0.00
13.png	Xerox WorkCentre 7535 @ 1200 dpi	Kraft 45 (Green)	710699474388	34.73	56.5	1	0.500	D	2.00	2.00
14.png	Xerox WorkCentre 7535 @ 1200 dpi	Kraft 45 (Green)	714290689172	34.73	56.5	1	0.500	D	2.00	1.00
15.png	Xerox WorkCentre 7535 @ 1200 dpi	Kraft -30 (Turquoise)	717491614868	34.73	56.5	1	0.500	F	1.00	2.00
16.png	Xerox WorkCentre 7535 @ 1200 dpi	Kraft -30 (Turquoise)	721792654263	34.73	56.5	1	0.500	F	1.00	0.00

Decodability issues in barcodes have various causes. The main causes are the substrate the barcode is printed on, the printer and printer ink used, and the properties of the barcode when generated such as the x-dimension, resolution, width, height, and magnification level.

(a) Green substrate.

(b) Turquoise substrate.

Figure 2: UPC-A barcodes printed on green and turquoise substrates to decrease contrast grades.

These issues were simulated by contaminating each image with zero mean Gaussian noise at a specific SNR (Signal to Noise Ratio) as well as with Salt & Pepper noise at specific pixel contamination densities.

The SNR of an image can be calculated using Eqn. 3.26 [44].

$$SNR = 10log_{10}\left(\frac{var(image)}{var(noise)}\right) \tag{3.26}$$

Solving Eqn. 3.26 for variance of the noise in an image yielded Eqn. 3.27, which was used to add noise to each image.

$$var(noise) = \frac{var(image)}{10^{\frac{SNR}{10}}} \tag{3.27}$$

Salt & Pepper noise was added to each of the images based on the relationship shown in Eqn. 3.28

$$I_{noise} = d * \texttt{numel}(I) \tag{3.28}$$

multiplying d by the number of elements found in array I where I_{noise} is the resulting image after adding the Salt & Pepper noise, d is the noise density, and I is the original image the noise is added to. Zero mean Gaussian noise at SNR's of -5dB, -6dB, -7dB, -8dB, and -9dB as well as Salt & Pepper noise at densities ranging from 0.45 to 0.7 in increments of 0.05 was added to each of the 16 UPC-A barcodes generated to best simulate those used in the experimental testing.

3.0.4 UPC-A Barcode Scanner

A conventional barcode scanner was emulated by implementing the analog edge detection algorithm previously described on any barcode image read in. Images read in to the barcode scanner program were resized to 227x226 pixels for uniformity using nearest-neighbor interpolation, as well as anti-aliasing and no color dithering, in order to introduce the smallest amount of noise into the image while resizing it [45, 46]. The intensity and contrast of each image were adjusted by mapping the intensity values in in a grayscale image to new values so that 1% of the data in the resulting image was saturated at low and high intensities of the original image data, thus increasing the contrast of the original image. The global image threshold level was computed for the intensity image using Otsu's method, which chose the threshold in order to minimize the intraclass variance of both the black and white pixels and was used to convert the image to a binary image [47]. After some manipulation of the data contained in each image, the data was checked to meet specific conditions based on the edges detected. If the conditions were met, the data was passed to a function to be decoded based on the values from Table 2, otherwise the read in barcode was considered unreadable. The decoded data was then compared to the actual encoded data to determine if the program decoded the data accurately. Details on how this scanner was implemented are given in the following subsections.

In the case of a barcode scanner with the trained image classifiers implemented, the image data was broken up into two sets of data containing all rows of the original data. Two separate functions identical to each other were used to decode each set of data using the respective trained SVM image classifiers. The first separate set of data defined was where

the first six elements of encoded data were present and was defined to start at the column after the first normal guard bar pattern and end at the last column before the centre guard bar pattern began. The second separate set of data defined was where the last six elements of encoded data were present and was defined to start at the column after the centre guard bar pattern and end at the last column before the second normal guard bar pattern began.

The input sets of data were partitioned into cell arrays by rows into 23 approximately equal sections, with each section containing all columns of the data and labeled sequentially. The function iterated through the length of each previously partitioned section (number of columns in inputed data) and further partitioned each of the previously partitioned sections of data into the sections that contained each of the 6 elements of encoded data. Arrays of length 23 filled with zeros for each of the 6 elements were created and a generic 1x6 vector was created to hold the decoded data for each of the first 6 elements. The function iterated through each of the 23 sections of the previously partitioned sets of data for each element and predicted what value was encoded in each of the 23 sections using a discriminant analysis classification model for each element.

The function predicted the label index using the trained SVM classifiers at each iteration corresponding to the class with the lowest average binary loss of the classifier and output the answer obtained at each iteration in the form of a string of characters (labels) in a structure. The label was converted from a structure to a double at each iteration and the value was placed into the respective array of zeros previously created for each element at each iteration. Histograms were generated using the array of values for each element as the input, with the width of the histogram bins equal to 1 and the binning method set to integers yielding histogram bin counts from all 23 sections of each element. The label values and the bin edge values seen by the histogram after the values were placed in bins were used and a value of 0.5 was added to each edge value to ensure the edge values were all whole numbers. A majority rules method was used where the function detected where the most instances of a value were seen. If the function saw two or more equal number of instances of different values, the decoded value of the respective element was set to NaN (not a number). Otherwise, the decoded value of the respective element was set to the value where the most instances were detected from the histogram. The two sets of data were decoded using Table

2 and Number Set A and Number Set C respectively and when the function decoded values for each of the 6 elements, the vector containing the decoded values was output to the main function.

The vectors containing the decoded data from were concatenated to make a vector containing the fully decoded data. The function checked the concatenated vector for any instances of NaN and created a vector of logical ones and zeros to determine if a NaN value existed or not in the concatenated vector at each position. The vector was checked for any nonzero entries and if any were found, the read rate was automatically set to zero (no read). Otherwise, the read rate was determined by comparing the resulting vector from concatenating the output vectors from each set of data with the actual encoded data.Details on how each of the scanners were implemented are given in the following subsections.

3.0.5 Conventional UPC-A Barcode Scanner

3.0.5.1 Horizontal Function A function was created titled `horizontal` and was called in the program to determine the location of vertical black lines in the image based on whether there were black or white pixels at specific locations. Inputs to the function were a predetermined horizontal rule value, the calculated number of rows and columns of the image, and the image itself. A generic row vector was created that was the length of number of total columns (width) of the image. The function iterated through each column in the image and determined the pixel values at each row location calculated by dividing the total number of rows in the image by the horizontal rule value and ensured that a whole number row was obtained regardless of the size of the image read in. If the pixel value at the respective column being evaluated was equal to 0 (white), the value of the generic row vector for that respective column was set to 1, otherwise the value was set to a 0. Once the function finished iterating through each column, the generic row vector that was modified at each iteration was passed back to the main program.

3.0.5.2 Region Function A function `Region` was created and called into the program in order to separate and place the instances of black and white vertical bars (columns)

together and to determine the total number of regions for the decoding process. The input to the function was the output of the previously executed `horizontal` function which was the row vector containing instances of either black or white pixels. The output variable for the number of regions was initialized and set equal to zero and the inputed row vector was checked for any nonzero elements. The function iterated through the inputed row vector for the length of the vector itself minus one and the data contained in the inputed row vector was separated. If the value at the respective position in the inputed row vector was not equal to the next value in the inputed row vector, the output variable previously defined was set equal to the output variable plus one. Otherwise, if it was determined that there were no nonzero elements in the inputed row vector, meaning a vector of all white pixels was read in, the output variable was set equal to a value of 60 so that the program could continue even if the row vector selected contained no data. Once the function finished iterating through the inputed row vector, the calculated value was passed back to the main program.

3.0.5.3 Collect Function A function was created titled `Collect` which collected the black and white pixels in the data. Inputs to the function were the outputted row vector obtained from the `horizontal` function and the value obtained from the `Region` function of the total number of regions. A generic row vector was created as the output of the function that was the length of the total number of regions obtained from the `Region` function and the array obtained from the `horizontal` function was checked for nonzero entries. The output vector was set equal to a row vector of zeros of length 48. If all elements of the inputed vector were zero, the vector was passed back to the main program. Otherwise, the function iterated to the inputed value of total regions obtained with the `Region` function and collected the pixel data. A generic variable was set equal to zero and if the respective position of the read in row vector being iterated was equal to zero or one, the generic variable was set to it's previous value plus one as long as the position of the row vector being iterated was equal to zero or one. Otherwise, the generic variable was placed in the respective position of the generic vector created at each iteration. Once the function finished iterating through the total number of regions obtained from the `Region` function, the generic row vector that was modified at each iteration was passed back to the main program.

3.0.5.4 Check Function A function `Check` was created to complete calculations needed to determine if the read in barcode was in fact a UPC-A (or EAN-13) barcode. The input to the function was the row vector obtained from the `Collect` function. A generic output variable was created and set equal to one and the function iterated to the length of the inputed row vector minus 11 (UPC). A variable was set equal to the smallest element of a 1x12 array where the first element of the array was the respective element of the inputed row vector at each iteration, and each element after was the respective element plus one sequentially to plus 11. A 1x12 array was created where the first element of the array was the respective element of the inputed row vector at each iteration divided by the value smallest value obtained from the vector in the previous calculation, and each element after was the respective element plus one sequentially to plus 11 divided by the value rounded to the nearest integer. If the maximum value contained in the array previously created minus the minimum value contained in the array was greater than 4, the number of elements per character, the generic output variable previously defined was set equal to it's value at that iteration plus one and the variable was passed to the main function. Otherwise, if the maximum value contained in the array previously created minus the minimum value contained in the array was less than or equal to 4, the function discontinued iterating and passed the value of the output variable at the time of the break to the main function. Another generic output variable was created and set equal to one and the function iterated from the length of the inputed row vector in increments of -1 down to a value of 12. A variable was set equal to the smallest element of a 1x12 array where the first element of the array was the respective element of the inputed row vector at each iteration, and each element after was the respective element minus one sequentially to minus 11. A 1x12 array was created where the first element of the array was the respective element of the inputed row vector at each iteration divided by the value smallest value obtained from vector in the previous calculation, and each element after was the respective element minus one sequentially to minus 11 divided by the value rounded to the nearest integer. If the maximum value contained in the array previously created minus the minimum value contained in the array was greater than 4, the number of elements per character, the generic output variable previously defined was set equal to it's value at that iteration plus one and the variable was passed to the main

function. Otherwise, if the maximum value contained in the array previously created minus the minimum value contained in the array was less than or equal to 4, the function would discontinue iterating and pass the value of the output variable at the time of the break to the main function.

A variable *rec* was set equal to the value contained in the array outputted by the `Collect` function at the element in the array at a position determined by the value of the first output variable of the `Check` function. The `round` function was used to calculate the resulting array p when each element of the array outputted by the `Collect` function was divided by the previously defined variable *rec*. An `if` loop was used to determine if the barcode read in was in fact a UPC-A barcode by checking that each of the follow conditions were met:

- The total number of regions outputted by the `Region` function minus the summation of the two output variables from the `Check` function is equal to 54.
- The value contained in array p at the position equal to the first output variable outputted by the `Check` function is equal to 1.
- The value contained in array p at the position equal to the first output variable outputted by the `Check` function plus 1 is equal to 1.
- The value contained in array p at the position equal to the first output variable outputted by the `Check` function plus 2 is equal to 1.

If all four conditions were met, the array p, as well as the first output variable calculated by the `Check` function were passed into the function `UPCA` to decode the data. Otherwise, the decoded data was set to a row vector of length 12 containing all zeros using the `zeros` function.

3.0.5.5 UPCA Function The function `UPCA` was created and called in order to decode the encoded data if the previous conditions were met. A generic row vector was created that was six elements in length to represent the first six elements of encoded data. The function iterated from the value of the variable passed in outputted by the `Check` function plus three in order to skip over normal guard bar pattern and in increments of four, up to the value passed in outputted by the `Check` function plus 23, ending at where the centre guard bar

pattern previously defined began. The function checked the data in the row vector p at the specified iteration compared to the corresponding 4 digit values that represent numbers 0-9 based on Table 2 from Number Set A and if the data was equal to any of the 4 digit values, the corresponding value from Table 2 was placed in the respective position of the generic row vector created that represented the first six elements of data. A second generic row vector was created that was also six elements in length to represent the second six elements of encoded data. Similarly, the function iterated from the value of the variable passed in outputted by the `Check` function plus 32 in order to skip over centre guard bar pattern and increments of four up to the value of the variable passed in outputted by the `Check` function plus 52 ending at where the second normal guard bar pattern previously defined began. The function checked the data in the row vector p at the specified iteration compared to the corresponding 4 digit values that represent numbers 0-9 based on Table 2 from Number Set C and if the data was equal to any of the 4 digit values, the corresponding value from Table 2 was placed in the respective position of the generic row vector created that represented the second six elements of data. The two generic arrays containing the first and second six elements of encoded data were concatenated into one row vector of length 12 and outputted to the main function.

3.0.5.6 Rate Function A function `rate` was created to calculate whether or not the decoded data from the program is equal to the encoded data defined in the *Encoded.mat* file at the respective iteration, otherwise known as the read rate. Inputs to the function were the row vector outputted by the `UPCA` function or the row vector of zeros created if the four conditions weren't met, and the data from the *Encoded.mat* file at the respective iteration. If the data outputted by the `UPCA` function was equal to the data in the *Encoded.mat* file at the respective iteration the output variable was set equal to 1 and passed back to the main function, otherwise the output variable was set equal to 0 and passed back to the main function. A row vector was created that stored the value outputted by the `rate` function at each iteration. The total accuracy of the program was calculated by dividing the summation of the values contained in the row vector of all the read rates obtained using the `rate` function by the total number of iterations.

3.0.6 Machine Learning Implemented UPC-A Barcode Scanner

The previously trained image classifiers saved as *BarcodeNumberSet_A_Classifier.mat* and *BarcodeNumberSet_C_Classifier.mat* were loaded into the program. Each image read in was resized to 227x226 pixels for uniformity using nearest-neighbor interpolation, as well as anti-aliasing and no color dithering, in order to introduce the smallest amount of noise into the image while resizing it [45, 46]. The intensity and contrast of each image was adjusted by mapping the intensity values in in a grayscale image to new values so that 1% of the data in the resulting image was saturated at low and high intensities of the original image data, which increased the contrast of the original image. The image data was broken up into two sets of data containing all rows of the original data. The first separate set of data defined was where the first six elements of encoded data were present and was defined to start at the column after the first normal guard bar pattern and end at the last column before the centre guard bar pattern began. The second separate set of data defined was where the first six elements of encoded data were present and was defined to start at the column after the centre guard bar pattern and end at the last column before the second normal guard bar pattern began. The two separated sets of data were then sent to created functions `section1A` and `section2` to decode the data.

3.0.6.1 Section1A Function The function `section1A` was created to decoded the first set of separated data using Table 2 and the Number Set A classifier. Inputs to the function were the first set of previously separated data from the image and the image classifier trained for Number Set A. The inputed set of data was partitioned into cell arrays by rows into 23 approximately equal sections, with each section containing all columns of the data and labeled sequentially. The function iterated to the length of each previously partitioned section (number of columns in inputed data) and further partitioned each of the previously partitioned sections of data into the sections that contained each of the 6 elements of encoded data. Arrays of length 23 filled with zeros for each of the 6 elements were created and a generic 1x6 vector was created to hold the decoded data for each of the first 6 elements. The function iterated through each of the 23 sections of the previously partitioned sets of

data for each element and predicted what value was encoded in each of the 23 sections using a discriminant analysis classification model for each element using the trained classifier for Number Set A.

The function predicted the label index using the SVM classifier at each iteration corresponding to the class with the lowest average binary loss of the classifier and outputted the answer obtained at each iteration in the form of a string of characters (Labels) in a structure. The label was converted from a structure to a double at each iteration and the value was placed into the respective array of zeros previously created for each element at each iteration. A histogram was generated using the array of values for each element as the input, with the width of the histogram bins equal to 1 and the binning method set to integers yielding histogram bin counts from all 23 sections of each element. The label values and the bin edge values seen by the histogram after the values were placed in bins were used and a value of 0.5 was added to each edge value to ensure the edge values were all whole numbers. The function detected where the most instances of a value were seen and if the function saw 2 or more equal number of instances of different values, the decoded value of the respective element was set to NaN (not a number). Otherwise, the decoded value of the respective element was set to the value where the most instances were detected from the histogram. When the function decoded values for each of the 6 elements, the vector containing the decoded values was outputted to the main function.

3.0.6.2 Section2 Function The function `section2` was created to decoded the second set of separated data using Table 2 and Number Set C. Inputs to the function were the second set of previously separated data from the image and the image classifier trained for Number Set C. Other than the inputs, the function worked identical to that of the `section1A` function and when the function decoded values for each of the 6 elements, the vector containing the decoded values was outputted to the main function.

The vectors containing the decoded data from `section1A` and `section2` were concatenated to make a vector containing the fully decoded data. The function checked the concatenated vector for any instances of NaN and created a vector of logical ones and zeros to determine if a NaN value existed or not in the concatenated vector at each position.

The vector was checked for any nonzero entries and if any were found, the read rate was automatically set to zero (no read). Otherwise, the read rate was determined by sending the resulting vector from concatenating the outputs of functions `section1A` and `section2` along with the encoded data read in from the respective *Encoded.mat* file at the current iteration to the `rate` function previously described. The total accuracy of the program was calculated in the same manner as previously described for the conventional barcode scanner program.

4.0 RESULTS

4.1 CLASSIFIER TRAINING

4.1.1 Number Set A Classifier

In order to create the bag of visual words for Number Set A, feature point locations for the images were selected by the function using the grid method where the grid step was [8 8] and the block width was [32 64 96 128]. The function extracted 815,768 features from 19,999 images and kept 80% of the strongest features from each category. In an effort to improve clustering, the function balanced the number of features from each image category by determining which image category had the least number of strongest features and only keeping the same number of the strongest features from each of the other image categories. Image category 10 was found to have the least number of strongest features with 64,563. A 500 word visual vocabulary (500 clusters) was created using K-Means clustering with 645,630 features. Clustering was completed (converged) in 19 iterations at approximately 1.03s/iteration and the bag of visual words was created in a total of 211.4373s (3.5239min).

In training the image category classifier for Number Set A, features were encoded for 19,999 images across all ten image categories and training the classifier took a total of 94.7103s (1.5785min). The trained classifier was evaluated using test data previously defined to obtain the confusion matrix of the classifier shown in Fig. 3 which took 114.1234s (1.9021min). The average accuracy of the trained classifier was calculated by taking the mean of the diagonal elements of the obtained confusion matrix and was found to be 97.77%.

$$\begin{bmatrix}
0.9899 & 0.0020 & 0.0005 & 0.0035 & 0 & 0.0005 & 0.0005 & 0 & 0 & 0.0030 \\
0.0005 & 0.9925 & 0.0015 & 0.0020 & 0.0005 & 0 & 0.0005 & 0 & 0.0005 & 0.0020 \\
0 & 0.0025 & 0.9835 & 0 & 0.0015 & 0 & 0.0030 & 0.0045 & 0 & 0.0050 \\
0.0025 & 0.0015 & 0 & 0.9850 & 0 & 0.0090 & 0.0010 & 0 & 0.0005 & 0.0005 \\
0.0005 & 0 & 0.0030 & 0.0005 & 0.9540 & 0.0020 & 0.0245 & 0.0095 & 0.0035 & 0.0025 \\
0 & 0.0035 & 0 & 0.0205 & 0.0005 & 0.9735 & 0.0005 & 0 & 0.0010 & 0.0005 \\
0.0005 & 0 & 0.0015 & 0 & 0.0110 & 0.0010 & 0.9795 & 0.0010 & 0.0020 & 0.0035 \\
0 & 0 & 0.0020 & 0.0015 & 0.0100 & 0.0065 & 0.0025 & 0.9700 & 0.0050 & 0.0025 \\
0.0015 & 0 & 0 & 0.0025 & 0.0045 & 0.0065 & 0.0165 & 0.0095 & 0.9580 & 0.0010 \\
0 & 0 & 0.0030 & 0 & 0.0015 & 0 & 0.0010 & 0.0030 & 0.0005 & 0.9910
\end{bmatrix}$$

Figure 3: Number Set A classifier confusion matrix.

4.1.2 Number Set C Classifier

In creating the bag of visual words for Number Set C, feature point locations for the images were selected by the function using the grid method where the grid step was [8 8] and the block width was [32 64 96 128]. The function extracted 810,108 features from 20,000 images and kept 80% of the strongest features from each category. In order to improve clustering, the function balanced the number of features from each image category by determining which image category had the least number of strongest features and only keeping the same number of the strongest features from each of the other image categories. Image category 4 was found to have the least number of strongest features with 60,003. A 500 word visual vocabulary (500 clusters) was created using K-Means clustering with 600,030 features. Clustering was completed (converged) in 19 iterations at approximately 0.96s/iteration and the bag of visual words was created in a total of 208.2364s (3.4706min).

In training the image category classifier for Number Set C, features were encoded for 20,000 images across all ten image categories and training the classifier took a total of 977.1823s (16.2863min). The trained classifier was evaluated using the test data previously

defined to obtain the confusion matrix of the classifier shown in Fig. 4 which took 120.6699s (2.0111min). The average accuracy of the trained classifier was calculated by taking the mean of the diagonal elements of the obtained confusion matrix and was found to be 97.81%.

$$\begin{bmatrix} 0.9830 & 0.0035 & 0.0010 & 0.0105 & 0 & 0 & 0 & 0 & 0 & 0.0020 \\ 0.0035 & 0.9870 & 0 & 0.0040 & 0 & 0.0030 & 0.0015 & 0 & 0.0005 & 0.0005 \\ 0 & 0.0010 & 0.9815 & 0.0005 & 0.0020 & 0 & 0.0030 & 0.0045 & 0 & 0.0075 \\ 0.0030 & 0.0010 & 0 & 0.9840 & 0.0000 & 0.0095 & 0.0005 & 0.0015 & 0 & 0.0005 \\ 0 & 0 & 0.0030 & 0.0010 & 0.9615 & 0.0020 & 0.0130 & 0.0085 & 0.0095 & 0.0015 \\ 0.0010 & 0.0005 & 0.0010 & 0.0165 & 0 & 0.9775 & 0.0010 & 0.0005 & 0.0015 & 0.0005 \\ 0 & 0.0005 & 0.0020 & 0.0010 & 0.0110 & 0.0030 & 0.9760 & 0.0010 & 0.0045 & 0.0010 \\ 0.0010 & 0 & 0.0030 & 0.0030 & 0.0085 & 0.0035 & 0.0045 & 0.9740 & 0.0015 & 0.0010 \\ 0 & 0.0005 & 0.0015 & 0 & 0.0040 & 0.0055 & 0.0115 & 0.0075 & 0.9675 & 0.0020 \\ 0.0005 & 0 & 0.0045 & 0.0015 & 0 & 0 & 0.0010 & 0.0030 & 0.0005 & 0.9890 \end{bmatrix}$$

Figure 4: Number Set C classifier confusion matrix.

4.2 TEST CODES

Table 6 shows the results acquired after running the set of 10,000 UPC-A barcodes generated with the Barcode Studio program through each of the barcode scanner programs when the generated barcodes were of perfect quality and after the random white and black defects were added to the codes.

4.2.1 Test Codes With White & Black Defects

Accuracy of the barcode scanner program with the image classifiers implemented was found to decrease due to three possible causes:

1. Misread - The program decoded the actual encoded value of the data as the wrong value.

2. No Majority - The program saw the same number of instances of two or more values after decoding the encoded data.

3. Misread & No Majority - The program saw at least one instance of both a Misread and a No Majority as described above.

Figures 5 and 6 show histograms of the number of instances the program saw a Misread, No Majority, or Misread & No Majority after adding the white and black defects respectively. Similarly, Figures 7 and 8 show histograms of the errors broken down by the instances seen by each of the trained number set classifiers.

The histogram shown in Fig. 9 shows the instances of misreads encountered by each of the trained number set classifiers when the test codes were contaminated with both black and white defects, indicating which values the classifiers read as opposed to which value was actually encoded into the data. Similarly, the histogram shown in Fig. 10 shows the number of instances when each of the trained number set classifiers obtained the same number of instances of two separate values and could not obtain a majority to output when the test codes were contaminated with both black and white defects and indicates which values the classifiers obtained the same number of instances of.

Table 6: Accuracy and average time required for each barcode scanner algorithm to decode an individual UPC-A barcode generated with and without black and white defects added.

	Perfect Quality		**White Defects**		**Black Defects**	
Scanner	Accuracy	Time (s)	Accuracy	Time (s)	Accuracy	Time (s)
Conventional	100%	0.0064	43.39%	0.0043	43.10%	0.0032
Classifier Implemented	100%	3.3434	99.84%	3.2414	99.82%	3.2152

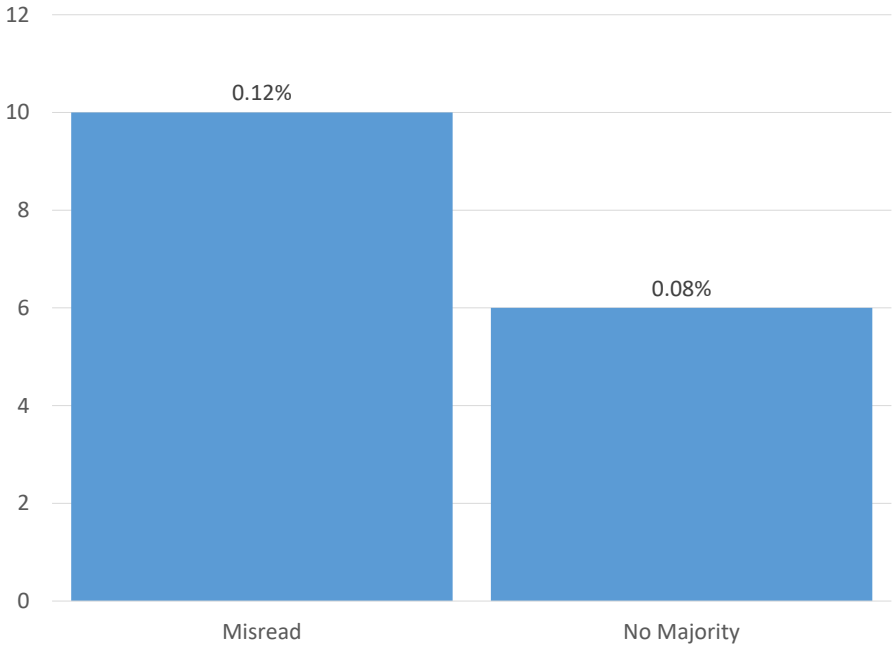


Figure 5: Histogram of errors found after adding white defects.

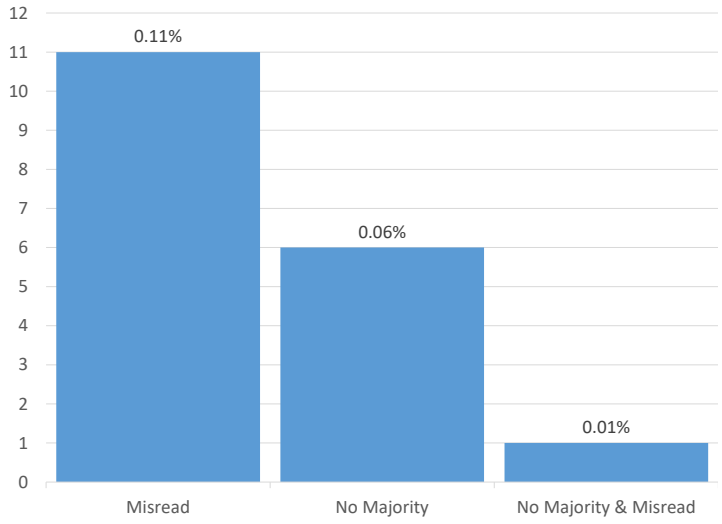


Figure 6: Histogram of errors found after adding black defects.

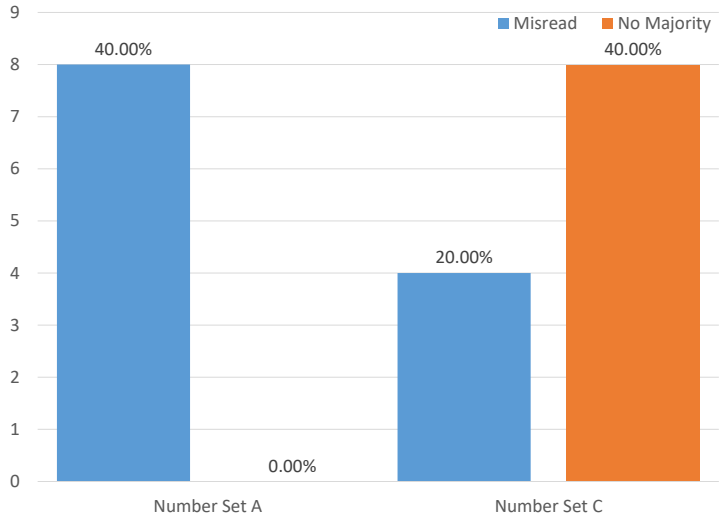


Figure 7: Histogram of errors found after adding white defects separated by type of error and which number set classifier caused the error.

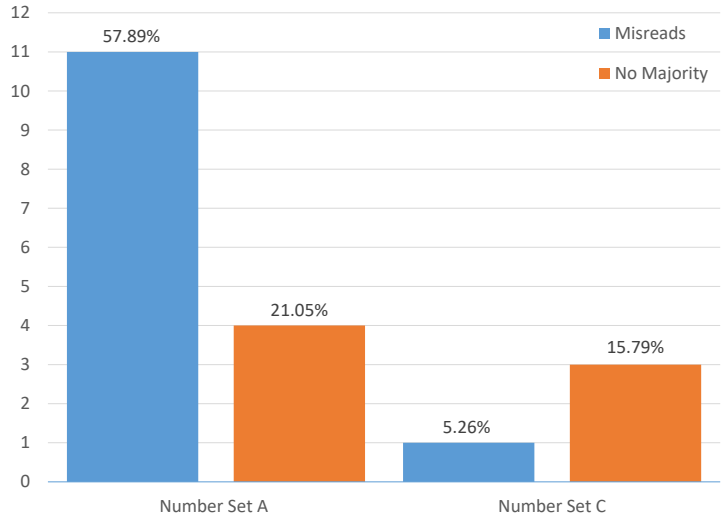


Figure 8: Histogram of errors found after adding black defects separated by type of error and which number set classifier caused the error.

4.3 CONTRASTS

Data collected on the set of UPC-A barcodes of different contrasts and decodabilities during experimental testing is shown in Table 7 for both Scanner A and Scanner B. Fig. 11 shows

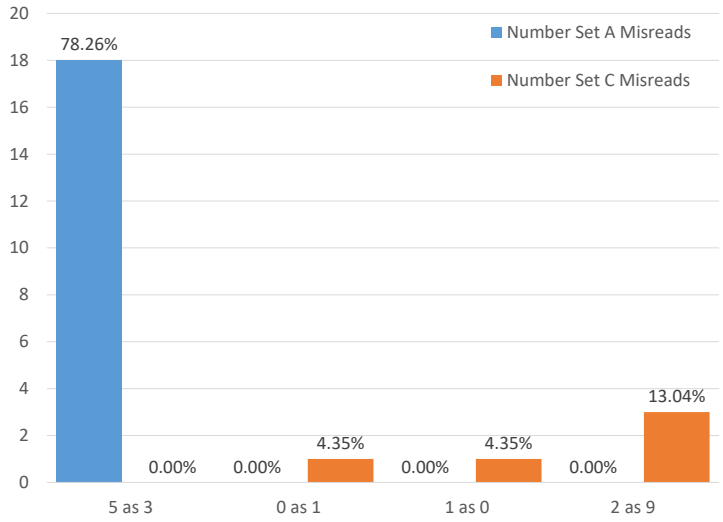


Figure 9: Histogram of misreads encountered with black and white defects separated by number set and what the trained classifiers read compared to what they data actually was.

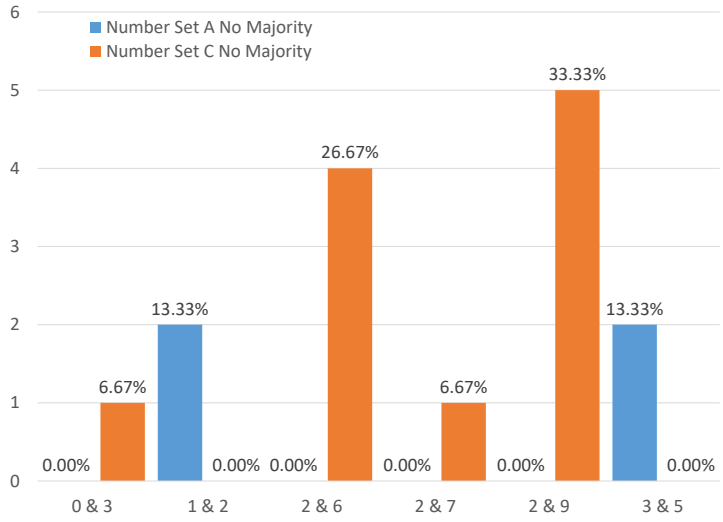


Figure 10: Histogram of errors caused by no majority being found by the trained classifiers with black and white defects separated by number set and the values the trained classifiers saw the same number of instances of.

a plot of the accuracy versus a range SNR's obtained using zero-mean Gaussian noise to contaminate each of the images of various contrasts generated to replicate the barcodes used in the experimental testing for both the conventional barcode scanner program as well as the classifier implemented barcode scanner program.

Table 7: Data collected from both Scanner A and Scanner B during experimental testing on each of the UPC-A barcodes tested.

	Scanner A			Scanner B		
Image	Read Rate 1 ft/s	Read Rate 5 ft/s	Read Rate 10 ft/s	Read Rate 1 ft/s	Read Rate 5 ft/s	Read Rate 10 ft/s
1.png	100.00	100.00	100.00	97.40	97.90	97.40
2.png	95.30	92.00	87.60	0.00	0.00	0.00
3.png	100.00	100.00	100.00	96.10	98.20	95.20
4.png	98.20	98.50	96.80	0.30	0.20	0.10
5.png	100.00	100.00	100.00	96.50	96.00	95.60
6.png	99.00	99.50	97.90	0.00	0.60	0.20
7.png	100.00	100.00	100.00	97.10	96.00	94.10
8.png	100.00	100.00	100.00	0.50	0.20	0.30
9.png	100.00	100.00	100.00	97.80	99.10	98.40
10.png	100.00	100.00	100.00	2.10	2.80	2.30
11.png	99.90	100.00	100.00	99.90	99.90	99.50
12.png	99.70	99.40	96.40	5.90	3.70	0.40
13.png	100.00	100.00	100.00	83.80	87.00	86.80
14.png	0.40	0.60	0.20	43.80	39.80	33.20
15.png	97.60	90.00	95.50	90.50	90.40	92.80
16.png	98.10	99.20	98.70	69.30	60.00	58.70

Fig. 12 shows a plot of the accuracy versus a range of contamination densities (percentage of image contaminated) using Salt & Pepper noise to contaminate each of the images of various contrasts generated to replicate the barcodes used in the experimental testing for both the conventional barcode scanner program as well as the classifier implemented barcode scanner program.

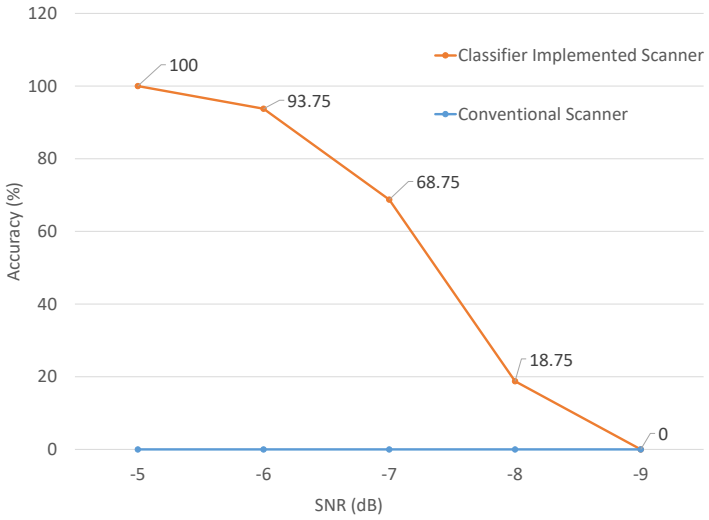


Figure 11: Plot of accuracy versus a range SNR's using zero-mean Gaussian noise to contaminate each of the images of various contrasts generated to replicate the barcodes used in the experimental testing.

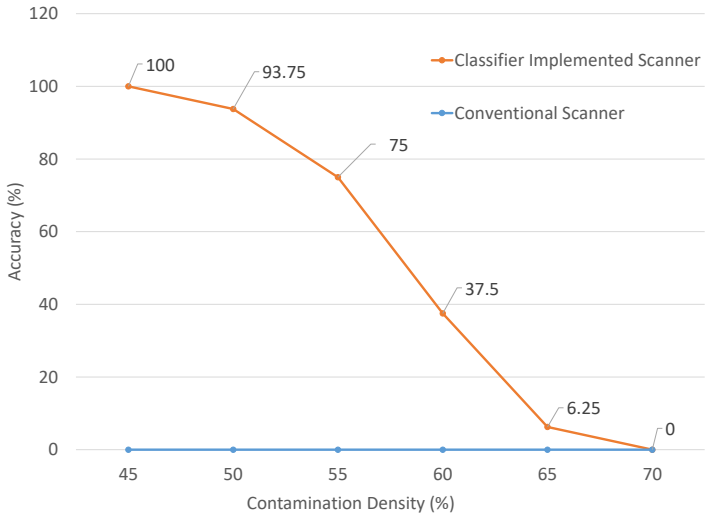


Figure 12: Plot of accuracy versus contamination density (percentage of image contaminated) using Salt & Pepper noise to contaminate each of the images of various contrasts generated to replicate the barcodes used in the experimental testing.

5.0 DISCUSSION

In this paper, we proposed a machine learning algorithm to decode barcodes more accurately than current barcode scanners on the market. This was accomplished by implementing image classifiers trained with SVM's into a barcode scanner algorithm. Results show that although preprocessing images for classifier training is a long and tedious process, training and testing the image classifiers with SVM's was a relatively fast process as it took under 8 minutes to train the classifier for Number Set A and under 22 minutes to train the classifier for Number Set C. Results also show that the proposed approach of implementing machine learning to decode barcodes yields higher decode accuracies in all tests in comparison to the conventional scanner algorithm tested as well as the experimental general distribution test results. Optimizing the machine learning algorithm should be considered in future work in order to increase decodability times. However, this current algorithm could be used as a backup for conventional scanners in the event they fail at reading specific barcodes in a test, as it will still decode the barcodes faster than having human intervention in a distribution environment.

The results presented when testing the barcodes with both random black and white defects implemented show that the implementation of machine learning is a very viable approach as it yielded accuracies above 99% in both cases. This is in comparison to the accuracies yielded by the conventional barcode scanner algorithm which were below 65% in both cases with random black and white defects implemented. It should be noted however, that the low accuracy of our conventional barcode scanner algorithm is most likely due to the fact that it doesn't accurately represent algorithms used by barcode scanner manufacturers. This is due to the proprietary nature of actual conventional barcode scanner algorithms. It is expected that if the test set of 10,000 barcodes could be printed and tested with barcode

scanners currently used in industry that the results would be much higher than those obtained using our conventional barcode scanner algorithm as barcode scanners used in industry most likely use significantly more advanced image processing techniques than were implemented in the conventional barcode scanner algorithm.

Results presented after testing the barcodes used in the experimental general distribution test cited also show conclusively that the implementation of the image classifiers is again a very viable approach. Analyzing the experimental test results presented in Table 7, it is clear that depending on the decodability and contrast grades given to each of the barcodes before testing along with at which speed they were tested at drastically reduces the accuracy of the scanners used in decoding the barcodes. Similarly, results after implementing the trained image classifiers on the same barcodes used in the experimental general distribution test show that as both Gaussian noise at various SNR's and Salt & Pepper noise at various densities is added to each image, the accuracy of the algorithm with the image classifiers implemented decreases. However, the images are beyond recognizable to the naked eye after adding the Gaussian noise and/or Salt & Pepper noise before the accuracy of the algorithm begins to decrease to values below those found in the results from the experimental general distribution testing.

All tests with the image classifiers implemented show that the majority of the errors produced are due to the trained classifiers simply reading the image presented to it as the incorrect value. However, using a majority rules approach also causes the image classifiers to encounter instances when the classifiers see the same number of two separate values as shown in the presented results.

6.0 CONCLUSIONS & FUTURE DIRECTIONS

6.0.1 Conclusions

This paper proposes a barcode scanner algorithm that, unlike conventional barcode scanning algorithms, uses a machine learning algorithm that implements trained image classifiers in order to decode the data contained in the barcodes. Decoding barcodes as accurately and as efficiently as possible is essential to our way of life as we know it as barcodes are used in a countless number of applications worldwide. The results of this study show that when it comes to accuracy, implementing image classifiers into the barcode scanner algorithm yields far better results than those obtained using our conventional barcode scanner algorithm as well as those obtained using barcode scanners used in industry.

6.0.2 Future Directions

The future direction of this work should consider optimizing the machine learning algorithm in order to process data more efficiently and at a higher rate of speed to decrease decodability times. However, the current algorithm could still serve as a backup if scanners fail to decode a barcode as it will be faster than having to introduce some type of human intervention to the distribution environment. If this work proves still proves viable once implemented in hardware, machine learning could then be used to train image classifiers that can be used for other symbologies previously discussed in this paper. Although classifier training is a long and tedious process, as was shown in this work, results are significantly better than what top of the line scanners being used in industry are currently capable of obtaining, making moving forward with this approach and research very worthwhile.

BIBLIOGRAPHY

[1] *GS1 General Specifications*, GS1 General Specifications 17.0.1, 2017.

[2] G. Weightman, *Eureka: How Invention Happens*. Yale University Press, 2015.

[3] N. J. Woodland and S. Bernard, "Classifying Apparatus and Method," U.S. Patent 2 612 994 A, Oct. 7, 1952.

[4] A. Q. Morton, "Packaging History: The Emergence of the Uniform Product Code (UPC) in the United States, 1970 75," *History and Technology*, vol. 11, no. 1, pp. 101–111, 1994.

[5] K. L. Yam, P. T. W. Takhistov, and J. W. Miltz, *Intelligent Packaging*. John Wiley & Sons, Inc., 2000.

[6] S. J. Shellhammer, D. P. Goren, and T. Pavlidis, "Novel Signal-Processing Techniques in Barcode Scanning," *IEEE Robotics & Automation Magazine*, vol. 6, no. 1, pp. 57–65, 1999.

[7] L. Chen, H. Man, and H. Jia, "On Scanning Linear Barcodes from Out-of-Focus Blurred Images: A Spatial Domain Dynamic Template Matching Approach," *IEEE Transactions on Image Processing*, vol. 23, no. 6, pp. 2637–2650, 2014.

[8] E. Joseph and T. Pavlidis, "Barcode Waveform Recognition Using Peak Locations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 6, pp. 630–640, 1994.

[9] O. Gallo and R. Manduchi, "Reading 1D Barcodes with Mobile Phones Using Deformable Templates," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 9, pp. 1834–1843, 2011.

[10] G. T. Kaya, "A Hybrid Model for Classification of Remote Sensing Images with Linear SVM and Support Vector Selection and Adaptation," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 6, no. 4, pp. 1988–1997, 2013.

[11] C. Huang, L. S. Davis, and J. R. G. Townshend, "An Assessment of Support Vector Machines for Land Cover Classification," *International Journal of Remote Sensing*, vol. 23, no. 4, pp. 725–749, 2002.

[12] C. Affonso, A. L. D. Rossi, F. H. A. Vieira, and A. C. P. d. L. F. de Carvalho, "Deep Learning for Biological Image Classification," *Expert Systems with Applications*, vol. 85, Supplement C, pp. 114–122, 2017.

[13] A. Rashidi, M. H. Sigari, M. Maghiar, and D. Citrin, "An Analogy Between Various Machine-Learning Techniques for Detecting Construction Materials in Digital Images," *KSCE Journal of Civil Engineering*, vol. 20, no. 4, pp. 1178–1188, 2016.

[14] B. Shi, X. Bai, and C. Yao, "An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and its Application to Scene Text Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 11, pp. 2298–2304, 2017.

[15] C. Creusot and A. Munawar, "Real-Time Barcode Detection in the Wild," in *2015 IEEE Winter Conference on Applications of Computer Vision*, pp. 239–245.

[16] W. Sawaya, S. Derrode, M. Ould-Barikalla, and J. Rivaillier, "Detection and Iterative Decoding of a 2D Alphabetic Barcode," in *2009 IEEE International Workshop on Machine Learning for Signal Processing*, pp. 1–6.

[17] M. Katona and L. G. Nyl, "A Novel Method for Accurate and Efficient Barcode Detection with Morphological Operations," in *2012 Eighth International Conference on Signal Image Technology and Internet Based Systems*, pp. 307–314.

[18] M. Katona and L. G. Nyl, *Efficient 1D and 2D Barcode Detection Using Mathematical Morphology*, pp. 464–475. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[19] J. Mayes, "Machine Learning 101." 2017.

[20] "*Machine Learning with MATLAB*." Section 1: Introducing Machine Learning, The Mathworks Inc., 2016, Available: https://www.mathworks.com/campaigns/products/offer/machine-learning-with-matlab.html?s_tid=solmain_mlr_cta2. Accessed: 01-Feb-2018.

[21] "*Machine Learning with MATLAB*." Section 3: Applying Unsupervised Learning, The Mathworks Inc., 2016, Available: https://www.mathworks.com/campaigns/products/offer/machine-learning-with-matlab.html?s_tid=solmain_mlr_cta2. Accessed: 01-Feb-2018.

[22] "*Machine Learning with MATLAB*." Section 4: Applying Supervised Learning, The Mathworks Inc., 2016, Available: https://www.mathworks.com/campaigns/products/offer/machine-learning-with-matlab.html?s_tid=solmain_mlr_cta2. Accessed: 01-Feb-2018.

[23] "*Machine Learning with MATLAB*." Section 2: Getting Started with Machine Learning, The Mathworks Inc., 2016, Available: https://www.mathworks.com/campaigns/

products/offer/machine-learning-with-matlab.html?s_tid=solmain_mlr_cta2. Accessed: 01-Feb-2018.

[24] D. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*. Prentice Hall, 2003.

[25] R. Gonzalez and R. Woods, *Digital Image Processing*. Prentice Hall, 2002.

[26] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual Categorization with Bags of Keypoints," in *Workshop on Statistical Learning in Computer Vision, ECCV*, vol. 1, pp. 1–2, Prague.

[27] D. G. Lowe, "Object Recognition from Local Scale-Invariant Features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, pp. 1150–1157 vol.2.

[28] T. Leung and J. Malik, "Representing and Recognizing the Visual Appearance of Materials Using Three-Dimensional Textons," *International Journal of Computer Vision*, vol. 43, no. 1, pp. 29–44, 2001.

[29] M. Varma and A. Zisserman, "Classifying Materials from Images: To Cluster or Not To Cluster," ECCV.

[30] S. Lazebnik, C. Schmid, and J. Ponce, "A Sparse Texture Representation Using Affine-Invariant Regions," in *Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003*, vol. 2, pp. II–II, IEEE.

[31] S. Routray, A. K. Ray, and C. Mishra, "Analysis of Various Image Feature Extraction Methods Against Noisy Image: SIFT, SURF and HOG," in *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pp. 1–5.

[32] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "Speeded-Up Robust Features (SURF)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[33] D. Pelleg and A. W. Moore, "X-Means: Extending k-Means with Efficient Estimation of the Number of Clusters," in *ICML*, vol. 1, pp. 727–734.

[34] B. Barrois and O. Sentieys, "Customizing Fixed-Point and Floating-Point Arithmetic - A Case Study in k-Means Clustering," in *2017 IEEE International Workshop on Signal Processing Systems (SiPS)*, pp. 1–6.

[35] R. Duda, P. Hart, and D. Stork, *Pattern Classification*. Wiley, 2012.

[36] R.-E. Fan, P.-H. Chen, and C.-J. Lin, *Working Set Selection Using Second Order Information for Training Support Vector Machines*, vol. 6. 2005.

[37] J. Xu, Y. Y. Tang, B. Zou, Z. Xu, L. Li, and Y. Lu, "The Generalization Ability of Online SVM Classification Based on Markov Sampling," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 3, pp. 628–639, 2015.

[38] V. N. Vapnik and V. Vapnik, *Statistical Learning Theory*, vol. 1. Wiley, New York, 1998.

[39] O. Chapelle, P. Haffner, and V. N. Vapnik, "Support Vector Machines for Histogram-Based Image Classification," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1055–1064, 1999.

[40] V. Vapnik, *The Nature of Statistical Learning Theory*. Springer Science & Business Media, 2013.

[41] J. Platt, "Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines," 1998.

[42] L. J. Cao, S. S. Keerthi, C. J. Ong, J. Q. Zhang, U. Periyathamby, X. J. Fu, and H. P. Lee, "Parallel Sequential Minimal Optimization for the Training of Support Vector Machines," *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 1039–1049, 2006.

[43] *GS1 General Specifications 1D Barcode Verification Process Implementation Guidline*, GS1 1D Barcode Verification Process Implementation Guidline 24.1,2015.

[44] P. C. Cosman, R. M. Gray, and R. A. Olshen, "Evaluating Quality of Compressed Medical Images: SNR, Subjective Rating, and Diagnostic Accuracy," *Proceedings of the IEEE*, vol. 82, no. 6, pp. 919–932, 1994.

[45] K. Vijayaraghavan, K. Parpyani, S. A. Thakwani, D. A, and N. C. S. N. Iyengar, "Methods of Increasing Spatial Resolution of Digital Images with Minimum Detail Loss and its Applications," in *2009 Fifth International Conference on Image and Graphics*, pp. 685–689.

[46] J. F. Blinn, "What Is a Pixel?," *IEEE Computer Graphics and Applications*, vol. 25, no. 5, pp. 82–87, 2005.

[47] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.