USING K-NEAREST NEIGHBOR IN OPTICAL CHARACTER RECOGNITION

Veronica Ong[1]; Derwin Suhartono[2]

[1,2]Computer Science Department, School of Computer Science, Bina Nusantara University
Jln. K.H. Syahdan No. 9 Palmerah, Jakarta Barat, 11480
[1]veronicaong510@binusian.org; [2]dsuhartono@binus.edu

ABSTRACT

The growth in computer vision technology has aided society with various kinds of tasks. One of these tasks is the ability of recognizing text contained in an image, or usually referred to as Optical Character Recognition (OCR). There are many kinds of algorithms that can be implemented into an OCR. The K-Nearest Neighbor is one such algorithm. This research aims to find out the process behind the OCR mechanism by using K-Nearest Neighbor algorithm; one of the most influential machine learning algorithms. It also aims to find out how precise the algorithm is in an OCR program. To do that, a simple OCR program to classify alphabets of capital letters is made to produce and compare real results. The result of this research yielded a maximum of 76.9% accuracy with 200 training samples per alphabet. A set of reasons are also given as to why the program is able to reach said level of accuracy.

Keywords: *optical character recognition, k-nearest neighbor, image processing, computer vision*

INTRODUCTION

It is easy for our human eyes to discern a three dimensional structure/object. The number of people can be easily counted in a certain picture.Their expressions are also guessable by looking at their expressions. It is possible to do to this, because our eyes can detect the objects through different aspects such as lightning, shadow, angles, or even backgrounds (Szeliski, 2011). Vision is an information processing task, where it is a process which turns images of the external world into a description that is useful to the viewer and not cluttered with irrelevant information. (Marr, 1982). It involves not only the study of extracting images (representing images), but also about how information is captured from images, which can serve as a basis for our thoughts and actions (processing of information).

Computer Vision is the study about the processes that a machine has to go through that enables it to have vision (to be able to see). The goal of computer vision is to make useful decisions about real physical objects and scenes based on sensed images (Shapiro & Stockman, 2001). Computer vision is a very influential field as it plays an important role in different kinds of fields such as industrial use where cameras are used to check whether mechanical parts have been created with the right size or forensic where they use computers to recognize people using the texture of their irises (Nixon & Aquado, 2002). There are various kinds of techniques used to depict vision such as image processing and pattern recognition.

Image processing is a method to perform operations on an image by converting it into a digital form of data. This operation is usually used to extract certain information from an image, where the information is then used in a decision-making process. Image processing usually consists of 3 steps:

importing the image into the program, analyzing and manipulating the image, then print the output of the image to the user (Engineers Garage, n.d.).

One of the results of the implementation of image processing is a program called Optical Character Recognition, or abbreviated as OCR. OCR is a program that can be used to recognize characters that exist in a certain image. In order to use the program, it needs to be fed with some examples of how each letter looks like. These examples are usually called training data. It can recognize both handwritten and printed text, but its accuracy highly depends on the training data that is given to the program.

Tesseract is an example of an OCR engine. It is an open source OCR engine which runs on various operating systems such as Linux, Windows, and Mac OS X. It has the ability to detect different kinds of languages, due to its ability to be able to train the system. This project was initially developed by HP Labs, but is now taken over by Google. It is considered as one of the most accurate OCR engines available. Willis (2006) mentioned that Tesseract was able to correctly recognize 97.74% of the file (6 mistakes out of 266 words) when the program was tested. An example of an Optical Character Recognition program based on the Tesseract engine has successfully been created on the Android operating system (Mithe, Indalkar, & Divekar, 2013).

The OCR is a great technology, as it has aided society with lots of tasks such as data entry, automatic number plate recognition, make electronic images of printed documents searchable, and assistive technology for visually impaired users (Wikipedia, 2015). A research has also been presented to recognize historical documents, either printed or handwritten, without any knowledge of the font (Vamvakas, Gatos, Stamatopoulos, & Perantonis, 2008). Taking character recognition to a higher level, a research on text detection and character recognition in scene images has also been conducted (Coates et al., 2011). The purpose of this research is to find out about the workings behind this convenient technology. In this research, the K-Nearest Neighbor machine learning approach will be used to train the program.

The K-Nearest Neighbor algorithm is a machine learning algorithm which is usually used in pattern recognition. It is considered as the top 10 most influential data mining algorithm in the research community (Wu et al., 2007). The K-Nearest Neighbor is a non-parametric type of algorithm (Thirumuruganathan, 2010), meaning it doesn't have to create an assumption about its environment. The number of parameters depends on the number of training data. Classification is done by calculating the average/majority distance from a test vector to its neighboring training vectors.

In this experiment, an Optical Character Recognition program is created using the K-Nearest Neighbor algorithm. Through this experiment, the process of how said algorithm works will be explained. The program will also calculate in how precise the K-Nearest Neighbor algorithm is in providing predictions for an OCR. The program will be provided with training and testing images to calculate its accuracy. An analysis is performed as to why the program is able to reach a certain precision.

Shah and Jethava (2013) mentions that a working Optical Character Recognition program consists of 6 steps: Image Acquisition, Preprocessing, Segmentation, Feature Extraction, Classification, and Post Processing. The following figure 1 illustrates the procedures of a working Optical Character Recognition program.

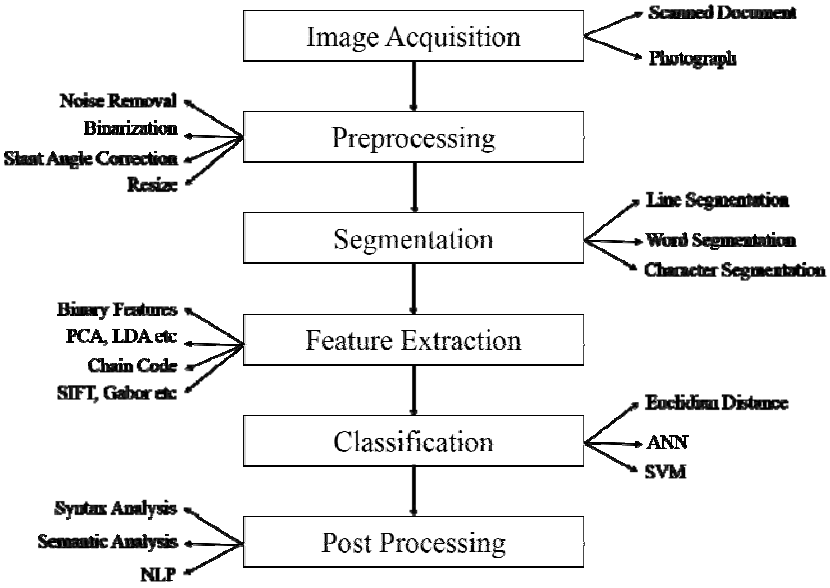


Figure 1 Steps of Optical Character Recognition Process
(Source: Shah & Jethava, 2013)

Each of these steps has their own respective tasks, which are as follows: (1) Image Acquisition: the process of acquiring the images by using hardware. (2) Preprocessing: consists of the operations needed to enhance the image for segmentation. (3) Segmentation: the document is first segmented by row histogram, and then segmented again by column histogram, to extract the words contained in the document. (4) Feature Extraction: the techniques which are used to extract the features of the characters. The extracted features will be used to train the system. (5) Classification: tested image is inputted into the program for classifying. Classifier may use different approaches like artificial neural network or support vector machine. Classification is done by comparing the tested image's feature with the pattern which has been stored for training. (6) Post Processing: consists of high level processes that help improve the accuracy of recognition, such as syntax analysis and semantic analysis.

An example of these steps can be seen on the Tesseract engine, as described by Mithe, Indalkar, and Divekar (2013). It firsts accepts an input image as a binary image (black on white text, or white on black text). This input image is sent to a connected component analysis, where the text is analyzed to find its pitch (the number of characters that can fit side-by-side in one inch). Afterwards, the document is broken down into words by analyzing the character spacing. Next, Tesseract performs two passes to recognize words. The first pass is sent to an adaptive classifier, which is able to recognize the text more accurately. The second pass is done to recognize the characters which were not recognized well in the first pass. In figure 2, the diagram shows the steps that the Tesseract engine has to go through to achieve character recognition.

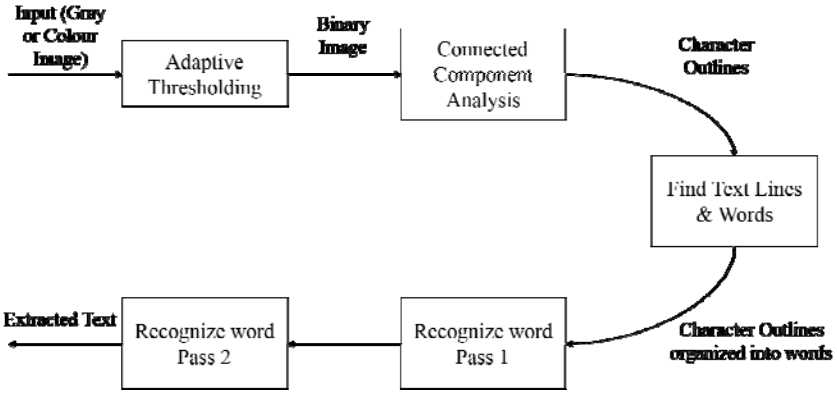


Figure 2 Architecture of Tesseract
(Source: Mithe, Indalkar, & Divekar, 2013)

A more complicated approach is used by Vamvakas, Gatos, Stamatopoulos, and Perantonis (2008), as the program is used to read historical documents, with no knowledge of the fonts used in the documents. Just like the steps used by Mithe, Indalkar, and Divekar (2013) and Shah and Jethava (2013), the program will first input the image into the system, and then converts it into a binary image (grayscale image). In this stage, an enhancement of the image is done to preserve the quality of characters, since the input is a historical document. After binarization and enhancement of the image, the program will do a segmentation of the document by text line detection, and word and character segmentation. The text line detection will segment the document vertically, where it will analyze the average height of the document, and detect potential text lines. Once the document has been segmented vertically, the characters' stroke are analyzed and divided into different segments. These segments are used to find the character's feature points. In this stage, the program will come up with different kinds of segmentation paths. The program will choose the best possible segmentation path, which satisfies the segmentation criteria. Next, since the program uses fonts it doesn't recognize, it will have to create its own database of characters using training images. To ensure accuracy of the database, the database will be finalized by the user to check if there are any incorrect data in the database. Finally, the program is ready to recognize a historical document, using the Support Vector Machine algorithm.

METHODS

There are various kinds of algorithm which can be used to create an Optical Character Recognition program. This experiment aims to find out the process behind the OCR program by using the K-Nearest Neighbor approach. The experiment also calculates the accuracy of said algorithm when implemented into an OCR. As K-Nearest Neighbor approach is an existing algorithm, this experiment conducts its data collection process through an Observational Method. This experiment is conducted in an observational way because its main focus is to explain the process of a simple OCR. In this method, the project is monitored and its data is collected over time. The collected data will then be analyzed as to why it is able to reach a certain result.

As mentioned before in section I, this experiment uses the K-Nearest Neighbor approach to train the system. The program is trained to classify different kinds of capital letter alphabets. The algorithm used in this experiment is based on a basic OCR program by Escrivá (2008). The experiment also requires some image processing functions to extract information from the image tested into the system. For that purpose, this experiment requires an additional library called OpenCV

(Open Source Computer Vision Library) for its image processing functions and K-Nearest Neighbor machine learning function.

The program will also need two kinds of images: training images and testing images. A training image is the image used to train / teach the program of how a certain capital letter alphabet looks like. A sufficient number of training images will be needed to reach an acceptable accuracy. A testing image is an image inputted into the program, and classifying which alphabet it belongs to. The testing image used must be different from the images used for training. These images are created in a ".pbm" extension, which is a file format for monochrome images. The images are created in GIMP, an open source image manipulation program, with a paintbrush tool. Every image has a dimension of 500px x 500px. There are a total of 5,226 images prepared for this experiment, which consists of 201 images per class / alphabet. These 201 images are divided into two groups: 200 training images, and 1 testing image for each alphabet.

In order for the machine learning function to work properly, those training images have to be inputted into the system. In this experiment, the basic OCR functions are observed by comparing the results of the program when inputted with different amount of training images. This program is divided into 3 main functions: preprocessing, training, and classification.

Preprocessing

After the image is inputted into the program, the preprocessing function enhances the image to ensure better accuracy in classification. The image is processed in such a way, so that it will be easier for the program to detect different kinds of strokes which belong to the same class/alphabet. To do this, every time an image is inputted, the program will determine the binary values of the image (whether the pixel is black or white). These values are then stored into a matrix. The size of the matrix is determined by the dimensions of the image.

In some cases, these images may contain a lot of white spaces, with the actual character occupying only some parts of the image. The white spaces may hinder the accuracy of the program, because it might cause two of the same letters to appear as different letters to the program.

Referring to the first image in Figure 3, it can be seen that the character is written in the left part of the image, leaving the right part blank. In the second image, it is written on the right side. But because their positions differ so much, the program will treat them as two different letters. To prevent this, the blank part of the picture should be lessened.The program has to crop the part of the image which contains the character. The program will use a bounding box to find the coordinates of the part which is to be cropped.

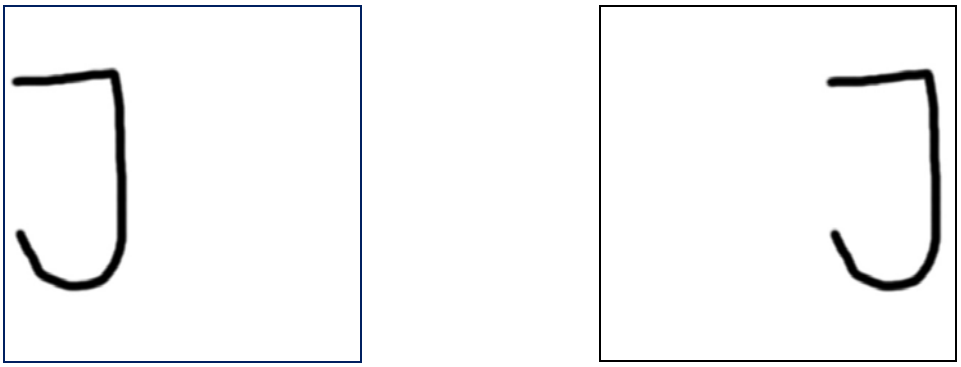


Figure 3 Letters with Blank Spaces

Figure 4 shows the result of the cropped letter Js from figure 3. After the cropping process, the two images now appear similar to each other. The coordinates of the bounding box are recorded. The pixel values from the cropped letters are also copied into another matrix. Afterwards, a new blank square image is created in the new program, where its dimensions depend on the result of the bounding box. The previously recorded bounding box coordinates are then compared: if its width is larger than its length, then its width will be used for the dimensions of the new image, or vice versa. After determining the dimensions of the blank image, the pixel data which has been copied into a matrix, is copied into the new image. This new image, is then scaled to a designated size (in this experiment, its designated size is 100px x 100px) to prevent training samples from having different dimensions.

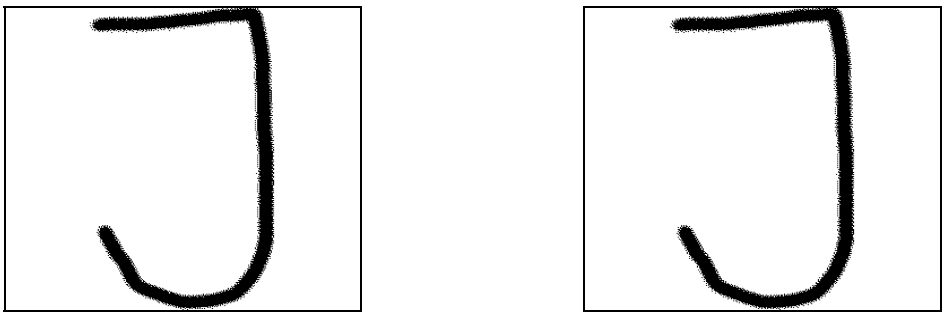
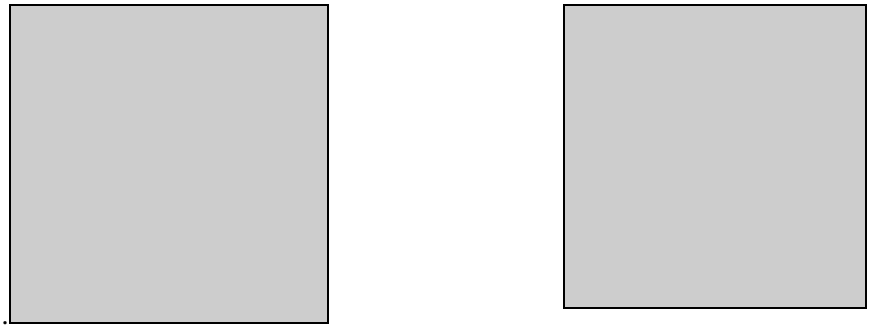
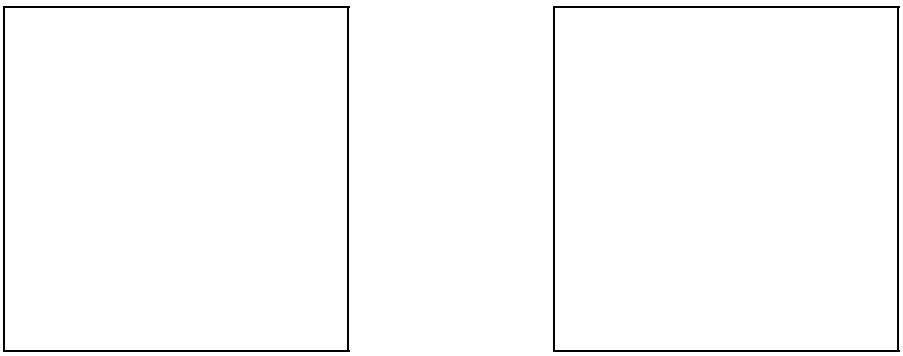


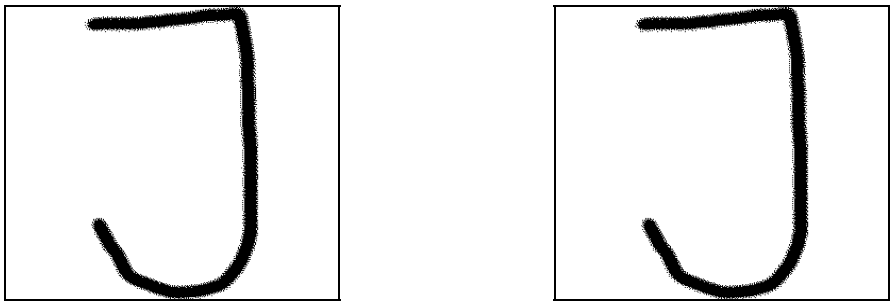
Figure 4 Cropped Letters



(a)



(b)



(c)

(d)

Figure 5 Steps in Processing the Images.
(a) A new square image is created based on the bounding box coordinates.
(b) Image is assigned to white color.
(c) Pixel values from the uncropped image are copied to the new image.
(d) Image is scaled to a designated size.

Figure 5 depicts the process of the preprocessing function on the letter J from figure 4. Once the coordinates of the bounding box are calculated, these coordinates are used to create a new square image. The program then assigns a white colored background to the new image. Next, the pixel values of the letter J from the old image are copied to the new image. After copying the pixel values, the new smaller letter J image is scaled to smaller size to match the dimensions of the training images. The dimensions of the training images and testing image (in this case, the new letter J image) should match to allow easier classification. This new scaled image is then passed to the next process: training.

Training

Images from the preprocessing functions are brought to the training function, so that the program can store the training data into its database. The K-Nearest Neighbor function is already built into the OpenCV library, so the function only needs to be called while passing the training data to said function. Since this experiment uses the K-Nearest Neighbor algorithm approach, the training data are stored in a feature space. A feature space is a dimensional vector of numerical features that represents a certain object. These objects have their respective coordinates / positions in the feature space according to their features. Their positions scatter around the feature space, but objects which belong in the same class are usually grouped near to each other because they own similar features. When drawn in a graph, the feature space would look like this:

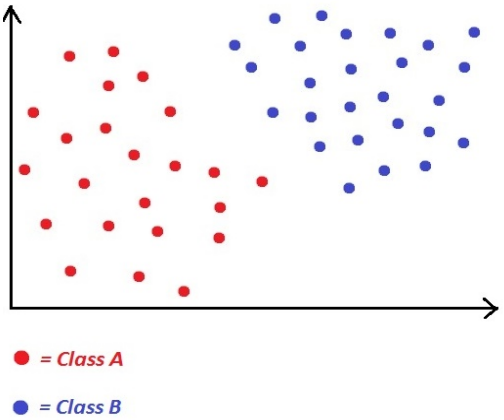
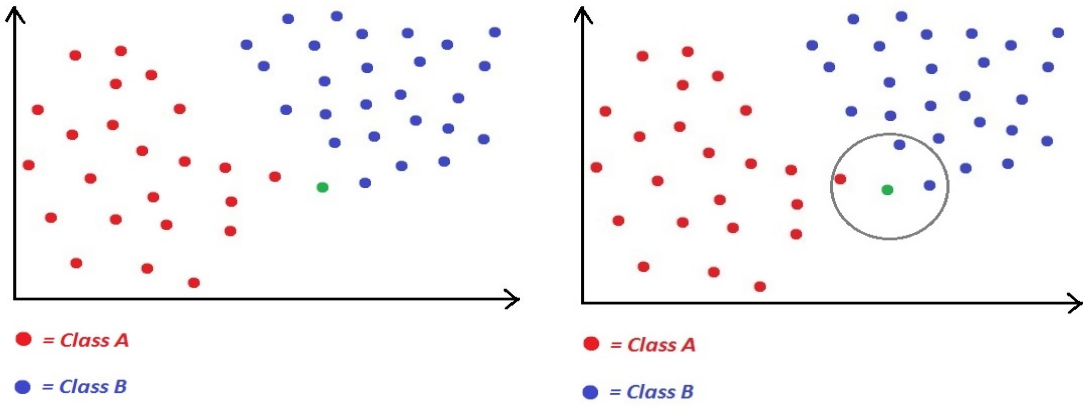


Figure 6 Feature Space

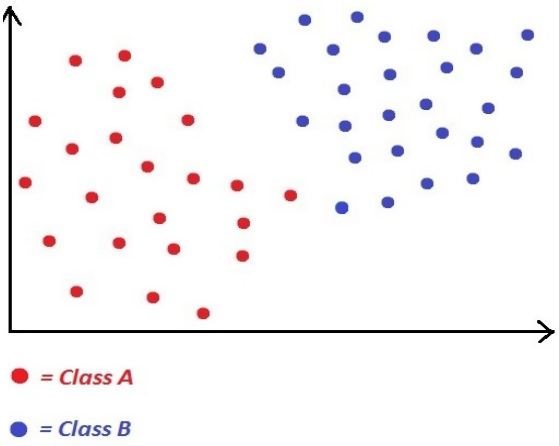
Figure 6 shows a picture of a feature space with two classes: class A and class B. In this figure, objects of class A are marked with red dots, while objects of class B are marked with blue dots. Each class has their own unique features which distinguish them from other classes. These features are what determine each object's position in the feature space. This is why class A looks as if they are grouped together in one location. The same thing applies to class B. In this research's case, there would be 5,200 objects scattered around the feature space, with 26 different colors (classes).

Classification

This function focuses more on the actions performed on the testing image. Similar to the training images, the testing image is first preprocessed to obtain a new cropped and scaled image. The processed image is then compared with the training images by using the K-Nearest Neighbor algorithm. The data of this processed image is passed to the feature space, and positions itself on the space depending on its features. Once in position, it determines some neighbors near to it, and does a comparison on the neighbors. The tested image is classified as the neighboring class with the highest amount of objects.



(a) (b)



(c)

Figure 7 Classifying New Data in Feature Space.
(a) Tested image positions itself in feature space.
(b) Objects near the tested image's position are compared.
(c) Tested image is classified as class B because the amount of class B grabbed by the image is greater than class A.

In Figure 7, the object in green is assumed as the tested image to be classified by the program. It grabs some of the neighbors nearest to it, and classifies itself as the class with the highest amount of objects, which in this case is Class B. Therefore, the tested image belongs to class B.

RESULTS AND DISCUSSIONS

This program was created with C programming language, and compiled in Microsoft Visual Studio 2010. As the interface is in the form of a terminal, the number of training images and file path should be inputted manually on the code. Upon execution, the program will load and preprocess the images from a folder containing all the training images that have been prepared for the program. After all training images have been extracted and stored into the program, the program proceeds to load and classify the tested image. The result of the classification is then printed out in the terminal, along with the percentage of the precision.

In this experiment, the program has run four times, with different amount of training images on each execution. The amount of training images is differed to see how well the program would respond with different levels of training given to it. The results are displayed in the table below, showing the results of the program's classification, expected answer, and the program's precision.

Table 1 Results With 50 Training Samples

1st Execution		
Result	**Expected answer**	**Precision**
L	A	80.00%
P	B	40.00%
L	C	50.00%
J	D	50.00%
E	E	70.00%

Table 2 Results With 100 Training Samples

2nd Execution		
Result	**Expected answer**	**Precision**
L	A	90.00%
B	B	30.00%
L	C	50.00%
D	D	40.00%
E	E	70.00%

Table 1 Results With 50 Training Samples (continued)

1st Execution		
Result	**Expected answer**	**Precision**
F	F	50.00%
H	H	90.00%
I	I	60.00%
J	J	80.00%
F	K	70.00%
L	L	90.00%
Y	M	70.00%
U	N	60.00%
G	O	50.00%
L	P	60.00%
O	Q	40.00%
F	R	70.00%
J	S	50.00%
J	T	100.00%
U	U	90.00%
V	V	50.00%
U	W	60.00%
X	X	60.00%
Y	Y	80.00%
J	Z	50.00%
Total correct answers: 11		

Table 2 Results With 100 Training Samples (continued)

2nd Execution		
Result	**Expected answer**	**Precision**
F	F	70.00%
H	H	100.00%
L	I	50.00%
J	J	100.00%
F	K	50.00%
L	L	90.00%
L	M	60.00%
U	N	50.00%
O	O	60.00%
F	P	60.00%
O	Q	90.00%
F	R	60.00%
J	S	50.00%
J	T	60.00%
U	U	100.00%
V	V	70.00%
W	W	50.00%
X	X	80.00%
Y	Y	100.00%
Z	Z	70.00%
Total correct answers: 15		

Table 3 Results With 150 Training Samples

3rd Execution		
Result	**Expected answer**	**Precision**
A	A	50.00%
B	B	60.00%
L	C	50.00%
D	D	70.00%
E	E	50.00%
F	F	100.00%
G	G	90.00%
H	H	80.00%
I	I	70.00%
J	J	100.00%
L	K	60.00%
L	L	100.00%
M	M	70.00%
N	N	40.00%
O	O	60.00%
F	P	40.00%
O	Q	80.00%
F	R	30.00%
J	S	50.00%
S	T	100.00%
U	U	100.00%
V	V	90.00%
W	W	50.00%
X	X	100.00%
Y	Y	100.00%

Table 4 Results With 200 Training Samples.

4th Execution		
Result	**Expected answer**	**Precision**
A	A	60.00%
B	B	60.00%
G	C	40.00%
D	D	70.00%
L	E	60.00%
F	F	100.00%
G	G	90.00%
H	H	80.00%
I	I	80.00%
J	J	100.00%
L	K	100.00%
L	L	100.00%
M	M	100.00%
N	N	60.00%
O	O	60.00%
P	P	60.00%
O	Q	80.00%
R	R	60.00%
J	S	50.00%
S	T	100.00%
U	U	100.00%
V	V	100.00%
W	W	50.00%
X	X	100.00%
Y	Y	100.00%

Table 3 Results With 150 Training Samples (continued)

3rd Execution		
Result	**Expected answer**	**Precision**
Z	Z	70.00%
Total correct answers: 19		

Table 4 Results With 200 Training Samples (continued)

4th Execution		
Result	**Expected answer**	**Precision**
Z	Z	70.00%
Total correct answers: 20		

Table 1, Table 2, Table 3 and Table 4 represent the results of the program's execution. Each table has a different amount of training samples, with Table 1 consisting of 50 training samples, Table 2 consisting of 100 training samples, Table 3 with 150 training samples, and Table 4 with 200 training samples.

During the first execution, the program was able to get 11 correct answers out of 26 testing images with 50 training samples. In the second attempt, it was able to score 15 out of 26 correct answers with 100 training samples. The program scored better in the third attempt with 19 out of 26 correct answers with 150 training samples. Finally, in the fourth execution, it was able to score 20 out of 26 correct answers with 200 training samples.

From Table 1, Table 2, Table 3 and Table 4, it can be seen that though the program is trained with a number of training samples, it can still make a mistake. But the difference in result on all four executions proves that the number of training samples play a big part for the program to be able to classify a character accurately.

Apart from the correct answers, there were also wrong answers by the program. There are three reasons that were deduced as to why the program is able to produce these wrong results. The first reason is because the lack of training images. The results show that the program improves when more training images are added to its knowledge base. Therefore, if more training images are added into the program, it will yield an even better result.

The second reason is because of some stroke similarities in the alphabet. The results in Table 1, Table 2, Table 3 and Table 4 show that the program did make some mistakes in parts where the alphabets have a similar stroke. For example, in Table 4, the program mistakes the test image C as the letter G, and test image Q as the letter O. A simple approach to solve this is to identify and create training images that features a more distinguishable feature from one letter to another, and by adding more training images.

The third reason about the occurrence of these errors is because of the K-Nearest Neighbor algorithm itself. The K-Nearest Neighbor algorithm gets the job done fairly well, reaching up to 76.9% (20 out of 26 correct answers) accuracy when trained with 200 training images. However, despite being a reliable algorithm, it is sensitive to noise and unbalanced data sets. To resolve the noise issue, a clearer training image may be used. Another alternative is by providing the program with an additional function to minimalize the noise. The solution to the unbalanced data sets issue is similar to the solution for the second reason, which provides consistent training images where each letter has a distinguishable feature.

CONCLUSIONS

This research shows and explains the use of the K-Nearest Neighbor algorithm in an Optical Character Recognition program. Through this experiment, it can be seen that the K-Nearest Neighbor algorithm can be used to classify images into alphabets in an OCR. It executes the job fairly well too, achieving a precision of 76.9%.

From the results, it can be inferred that the number of correct answers increases as more training samples are added into the program. The reason for this occurrence is because by adding more training samples into the program. The program is taught to recognize more varieties of possible strokes for a certain character. It will be easier for the program to adapt to any stroke tested to it, if it has a good knowledge base.

Despite its precision, there are a few solutions which can be done to improve the program. In order to achieve more precise results, the program's knowledge base too, has to be accurate. This can be done by adding more training images into the program. The experiment also shows that the number of correct answers increases when more training images are added into the program. Besides adding training images, identifying the distinguishable features of each letter will also help when creating the training images. This allows the program to accurately classify letters which look similar to each other. A cleaner training image is also needed for the program to be able to recognize the letter.

Some improvements can be applied to this program to make it more robust and useful. First, by creating a more robust algorithm that can detect noise in the images, it will allow the program to ignore irrelevant strokes or scribbles contained inside an image. Second, by adding a feature that will allow the OCR program to detect spaces and lines, which will enable the OCR to detect more than one letter contained in an image.

REFERENCES

Coates, A., Carpenter, B., Case, C., Satheesh, S., Suresh, B., Wang, T., Wu, D. J., & Ng, A. Y. (2011). Text Detection and Character Recognition in Scene Images with Unsupervised Feature Learning. *Document Analysis and Recognition (ICDAR)*, 440-445.

Engineers Garage. (n.d.). *Introduction to Image Processing*. Retrieved on July 8, 2015 from http://www.engineersgarage.com/articles/image-processing-tutorial-applications.

Escrivá, D. M. (2008, November). *Basic OCR in OpenCV*. Accessed on June 11, 2015 from http://blog.damiles.com/2008/11/basic-ocr-in-opencv/.

Marr, D. (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. New York: W.H. Freeman

Mithe, R., Indalkar, S., & Divekar, N. (2013). Optical Character Recognition. *International Journal of Recent Technology and Engineering*, *2*(1), 72-75

Nixon, M. S., & Aquado, A. S. (2002). *Feature Extraction and Image Processing*. Great Britain: Newnes

Shah, M., & Jethava, G. B. (2013). A Literature Review on Hand Written Character Recognition. *Indian Streams Journal*, *3*(2), 1-19.

Shapiro, L. G., & Stockman, G. C. (2001). *Computer Vision*. Upper Saddle River, N.J: Prentice-Hall

Szeliski, R. (2011). *Computer Vision: Algorithms and Applications*. London: Springer-Verlag

Thirumuruganathan, S. (2010, 7 May). *A Detailed Introduction to K-Nearest Neighbor (KNN) Algorithm.* Retrieved on July 21, 2015 from https://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm/.

Vamvakas, G., Gatos, B., Stamatopoulos, N., & Perantonis, S. J. (2008). A Complete Optical Character Recognition Methodology for Historical Documents. *The Eighth IAPR Workshop on Document Analysis Sytems*, 525-532.

Wikipedia. (2015). Optical Character Recognition. Retrieved on July 8, 2015 from https://en.wikipedia.org/wiki/Optical_character_recognition

Willis, N. (2006, 28 September). Google's Tesseract OCR engine is a quantum leap forward. Retrieved on July 8, 2015 from http://archive09.linux.com/articles/57222.

Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., & Motoda, H. (2007). Top 10 Algorithms in Data Mining. *Knowledge and Information Systems*, *14*, 1-37.