[[1]](#footnote-2)

*Abstract*— Optical Character Recognition (OCR) is the problem of recognizing characters from a visual format and converting them to a machine readable format. It has a wide variety of applications ranging from Automatic Number Plate Recognition to simplifying document storage and retrieval. This involves the task of first detecting text blocks, then individual characters and classifying them. For the classification process I will be using K-Nearest Neighbor classifier and a Support Vector Machine. Conventionally increasing OCR accuracy has focused on improving the accuracy of method used for conversion of individual characters. I propose that once the characters for a word have been detected, we check for it in a dictionary. If it is not found, we can run a spell checker algorithm on it to get a plausibly better conversion for the given word. However, one drawback here will be that it might lead make the conversion actually worse for proper nouns.

Optical Character Recognition

Saravanan Setty, UFID : 3329 9221

*Index Terms*— Optical Character Recognition, OCR, SVM, Support Vector Machine, KNN, K-Nearest Neighbors

# INTRODUCTION

IN today’s advanced day and age a lot of things are moving slowly from the offline world to the online world. People prefer to do things from the comfort of their homes without standing in long queues leading to changes like these. There are many reasons and scenarios why one might use OCR. Some of them are listed below :

## Increasing Efficiency of Data Entry

Whether it is buying tickets, applying for jobs, cashing checks, most of these things can be done online. But a lot of these things also involve physical items and forms from which information needs to be extracted and converted into the digital form. For example some banks allow people to upload scans of their checks online and the bank’s OCR software will automatically extract the relevant information about amount to be transferred, the accounts involved etc. Now, this task can be automated to speed up the process and a human can only verify all the information is correct before finally approving the check instead of manually filling in the data by himself.

## Digitizing old Books and Documents

Before computers were widely adopted in corporate usage, most of the official record were present only in physical form. The problem with physical form is that it takes up space, can get easily destroyed and takes a lot of time to search through it to find a specific piece of information. Now, by taking a scan

of the physical document, we can solve the problem of it taking up physical space and fragility but then it still takes up a lot of space in computer memory and can’t be searched through. Here is where OCR can help. Since most of the documents don’t have color information often, we can discard it and effectively just use 2 colors black and white by binarizing the document and hence solving the problem of space. After this, the later stages of OCR will create an actual digital document to facilitate easy searching of the data.

## Rule Enforcement and Vehicle Tracking

In electronic toll collection systems we can use Automatic Number Plate Recognition (ANPR) to detect and fine drivers who haven’t paid the toll. While this can be done manually as well, but then it would defeat the whole point of electronic toll collection which is to reduce human intervention levels to as low as possible. Similarly, Law enforcement can use a set of CCTV cameras to track movement of suspicious cars around the city.

OCR can be classified into different types based on different parameters :

## Type of Input Images

If we get only the final image of a document on which OCR is to be done, it is called as offline OCR. On the other hand, if we have chronological information about how each stroke in the character was made, it is called as online OCR. Typically online OCR has higher accuracy than offline OCR since we have access to more information compared to offline OCR.

## Type of Text

Text can be handwritten or machine generated text. Generally we get higher accuracy on machine generated text due to the uniformity of characters generated. Handwritten text can be further classified into cursive or non-cursive. Older OCR systems were able to initially work with machine generated text and that too of a specific. Modern OCR systems are able to deal with different fonts and handwritten text as well. Here only handwritten non-cursive text and machine generated text of single font are considered.

## Zone-based/Template OCR

If the input image is something like a form, the OCR software has the advantage of knowing where exactly the text will be present and hence it doesn’t need to deal with the problem of detecting where text is present. This type of OCR is also called as Automated Forms processing. Other examples of Automated Forms include extracting information from a Passport, Resume etc.

# Process of ocr

The process of OCR can be broadly split into the following 6 tasks :

## Obtaining the Raw Images

The raw images used for the purpose of this project are screenshots of computer generated text from software like Powerpoint and word. Additionally images of some handwritten text are also used for testing but, as expected we get lower accuracy there compared to that for computer generated text. In total 62 character classes are present. 26 for capital alphabets, 26 for small alphabets and 10 for numbers from 0 to 9.

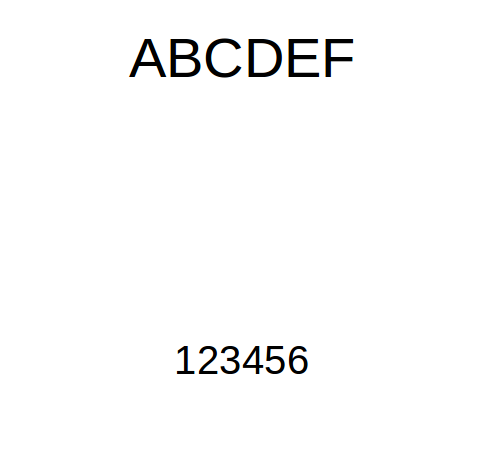


Figure 1 : Sample Raw Image

## Preprocessing

The raw images obtained from Step 1 may have noise in them if they are actual images taken from a camera and not a screenshot. A Gaussian blur is applied to remove the noise from the images. Then, the document is binarized in such a way that the resulting image will consist of only the text as foreground pixels and everything else as background pixels. To achieve this we use the adaptive threshold function in Open CV. One other alternative here was Otsu for thresholding, but in case of unbalanced lighting in the image, Otsu resulted in some text being omitted. Adaptive threshold is better for such cases since it adaptively decides different thresholds for different regions of the image.

## Segmentation

Here we further process the image to identify the text only regions first. As discussed in [1], extracting Maximally Stable Extremal Regions(MSERs) are a good point to start as potential character regions. After this we can further increase the accuracy by applying filter based on geometric properties like minimum area, stroke width, eccentricity. An implementation of MSER is available in OpenCV[2] and documentation is contained in [3].

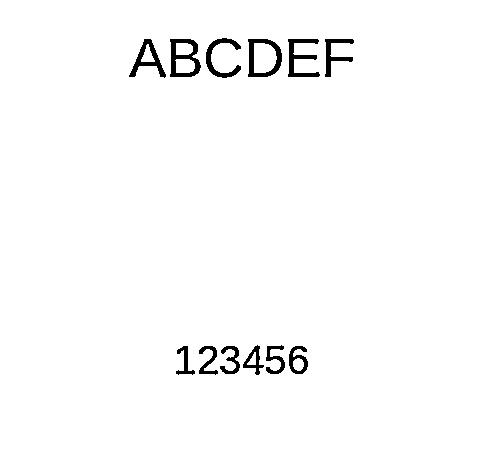


Figure 1 : Input after Preprocessing

Once we get the regions which have high probability of containing text, a bounding box is created around it and information about it is stored. We first need to identify bounding boxes for each word in the image and then further inside each word, we need to find characters. But before dilation, the image needs to be converted in a form such that the foreground pixels are lighter than the background pixels. Hence if the image contains darker pixels as foreground the image is negated else the image is left unchanged.

As can be seen in the sample, the foreground pixels are darker and hence the image was negated. After this the image is dilated so that all the characters of word merge together to form a blob. As a result, running MSER returns the location of word blobs.

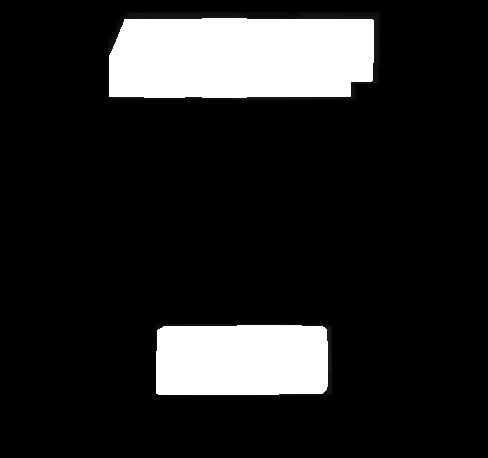


Figure 1 : Text Blobs Detected after Dilation

Once we have those blobs, we run the MSER algorithm on a cropped version of the image containing only the word block. This gives MSER regions for individual characters which are then used to get bounding boxes for those characters and stored.

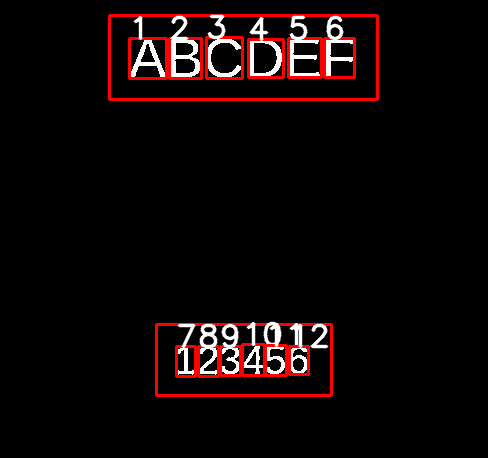


Figure 1 : Image after Segmentation

The numbers are added so we can visually ensure that no extra character boxes are created. To remove similar boxes the groupRectangles function from OpenCV[2] was used. Additionally in some cases, noise was also considered as a character. To avoid such cases, a minimum threshold for the character box area was set. Similarly, to ensure that the entire image is not returned as MSER, a maximum area threshold is set as well.

## Feature Extraction

Before features can be extracted we have to make sure that all the images have same color for background and foreground pixels. To detect the foreground pixel color, inside each character box, majority voting is done to determine the background color. If background has white and foreground black, the colors are inverted else the image is left as it is. The different features then extracted are :

1. Mean X Value

For each pixel that lies on the character inside the bounding box, we sum up the x coordinate values and divide by the number of pixels to get this feature.

1. Mean Y Value

For each pixel that lies on the character inside the bounding box, we sum up the y coordinate values and divide by the number of pixels to get this feature.

1. Number of Black Islands in Image

A region is considered a black island if all pixels lying in it consist of only background or black pixels. To calculate number of such islands, a recursive procedure is used. Additionally to speed up the process, the image is scaled down. This feature is really helpful at is indifferent to the size of the character and ratio of the strokes in characters.

1. Grid Values

To calculate this feature the image is split into n\*n by grids. Corresponding to each “grid” of the image, we set the value in feature matrix 1, even if there is a single foreground pixel present in the grid else feature matrix gets value 0. The 2-D matrix is later flattened to a 1-D list. Sample 2-D matrix is shown below.

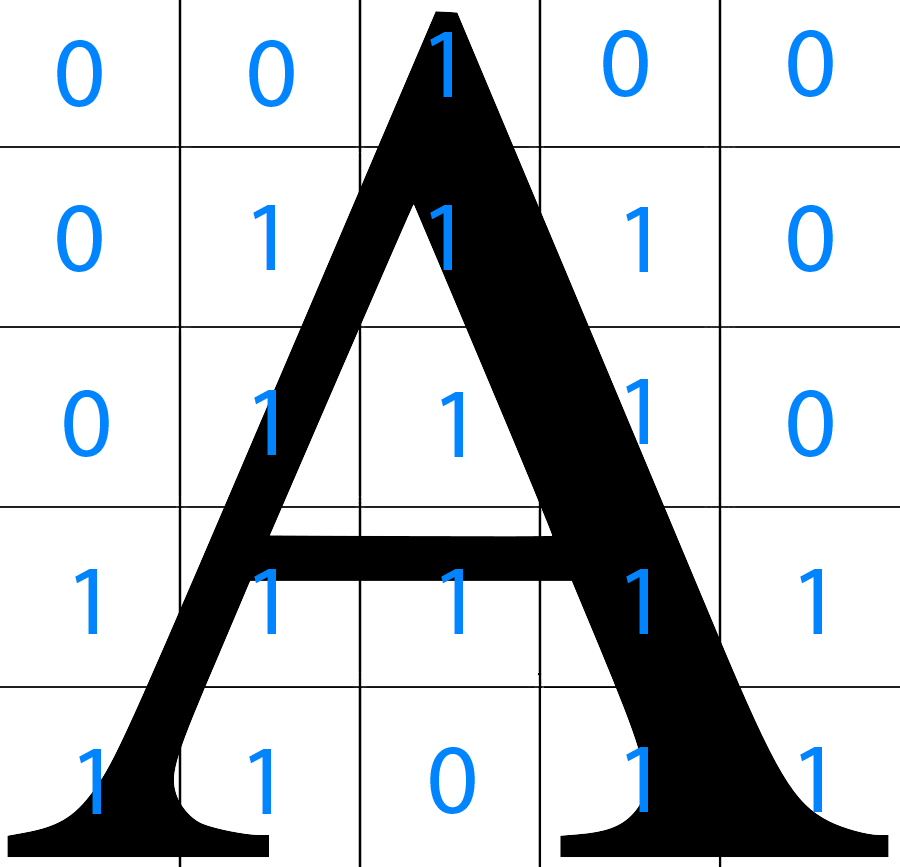


Figure 1 : 2-D Matrix for 5 by 5 grid on character A.

1. Hu Moments

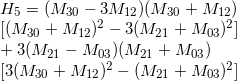
For each pixel that lies on the character, the moment M of order (i+j) is defined as follows :



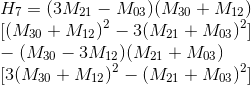
These moments are then further used to get the 7 Hu Moments [5] which are invariant to translation, rotation and scale. The 7 Hu Moments are defined as follows :











Additionally, all of them except the last one are invariant to reflection as well [5].

## Classification

For the actual classification of each character from its visual form to textual data, K-Nearest Neighbors and Support Vector Machines are used.

1. K-Nearest-Neighbors

In KNN classification each new instance gets assigned to the class label which is the most common among its K Nearest Neighbors and hence the name. It is a lazy learning algorithm since KNN doesn't perform any operation on the training data till a query is received.

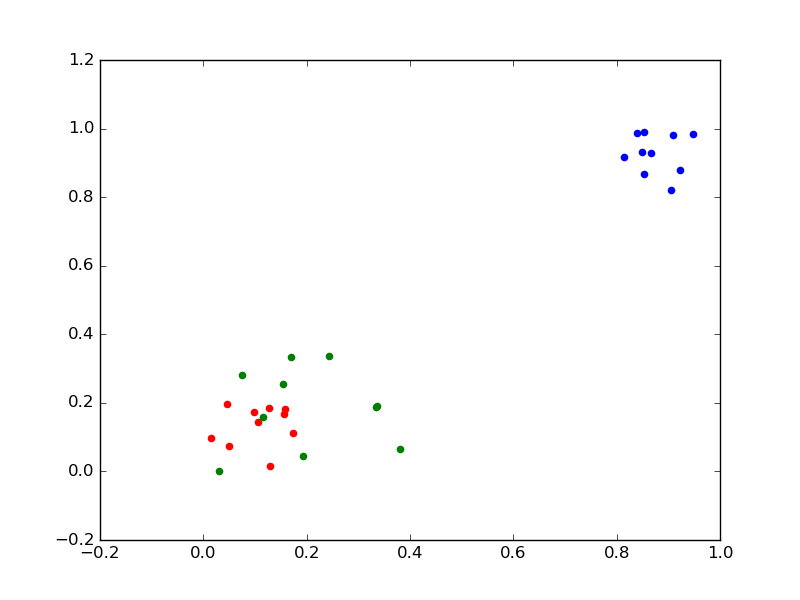


Figure 1 : How KNN Works

Consider the following case where 10 red points and 10 blue points were used for training of the KNN and each point belongs to class with name of that color. As, it is clear visually, if KNN is used with k value=10 for the new test points which are green, all of them will get classified as belonging to class Red since majority of the points in red are closer to green than blue points.

In KNN I have set the value of K to 1 since each image is classified based on the character that it looks most similar to. For a 12 class, classifier, accuracy as high as 100% has been obtained using both KNN and SVM methods. With increase in number of classes, the accuracy decreases.

The similarity in KNN is calculated on the basis of a distance measure. Some types of distance measures used are :

1. Euclidean Distance

For given 2 points p(p1,p2,..,pn) and q(q1,q2,...,qn) in n-D space, the Euclidean distance between them is mathematically defined as :



1. Taxi Cab Distance

In this type of distance only the absolute difference in corresponding dimensions is considered and summed up. For given 2 points p(p1,p2,..,pn) and q(q1,q2,...,qn) in n-D space, the Euclidean distance between them is mathematically defined as :



This distance is also called as Manhattan distance. The name originates from the idea that to travel from one part of the city to another, the actual distance to be covered was sum of the absolute differences in x and y dimensions.

1. Minkowski Distance

There is a pattern in the previous 2 distances used. The Taxi Cab distance can be rewritten as :



and Euclidean distance as :



The general form of this is the Minkowski Distance. Minkowsi Distance of order x is mathematically defined as :



If x=1 it is Taxi Cab distance, x=2 it is Euclidean Distance.

Additionally, KNN can be weighted or non-weighted. In non-weighted KNN all the points are given equal weightage whereas in the other different points get different weightage which is inversely proportional to the distance between the points.

1. Support Vector Machine

nan Support Vector Machines are a type of supervised learning algorithm. It constructs a set of hyperplanes which act as decision boundaries which are used to decide which class an element belongs to. In its raw form, it is suitable only for the binary classification problem. But it can be used to solve multi-class classification problem as well by splitting the problem into multiple binary classification problems. In the easy case, the input given to SVM is linearly separable, however that need not be the case always.

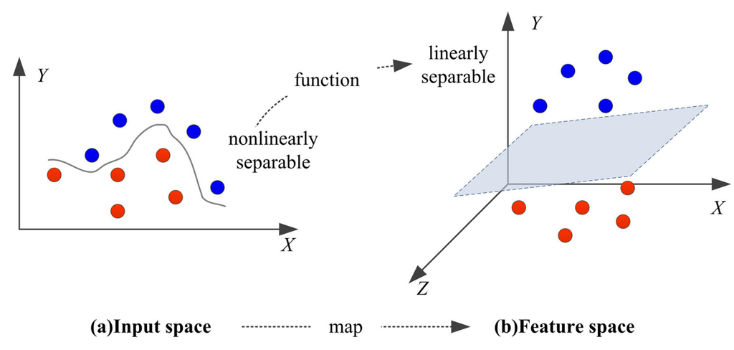


Figure 1 : Mapping from Input Space to Feature Space [6]

As shown in image, if the data is not linearly separable, it is mapped to a higher dimension where those points might be separable. For example, in Figure 7, the input is not linearly separable in 2-D space but when it is mapped to a 3-D space, it becomes linearly separable.

As is visible from Figure 7, many such hyperplanes exit which will linearly separate the red and blue points. In such a case, SVM will select the hyperplane which maximizes the distance between training points and the hyperplane.

As seen in the evaluation section, both KNN and SVM gave exactly same results in all inputs tested yet. This perhaps implies that the only way to improve accuracy to even further would be to generate better features.

## Post Processing

It is difficult to distinguish between characters that look similar visually. In such cases we can look for other useful information like the context in which each character appears. Once all the characters for a word are converted, we can search if it exists in a dictionary. If it doesn’t we pass it through a spell checker algorithm [7] to suggest possible corrections for the word. This can be further extended to the word level by checking if the structure of a sentence itself makes sense or not by using rules of English grammar. But for purposes of this project we have limited ourselves only to the word correction scheme.

Finally before the text can be displayed after conversion, we need to figure out the right order in which it needs to be displayed. The order in which words are displayed is how humans read it, top to bottom and left to right. To get the text in this specific order, the word boxes are first sorted on the basis of y coordinate and then x coordinate. A stable sort is used for this purpose. After this, inside each box, the characters are sorted on the basis of x-coordinate. Additionally, in some cases the words appear to be in the same line but it so happens that the bounding box for one of the words may actually be a bit higher resulting in messed up order. An example of such case is shown below :

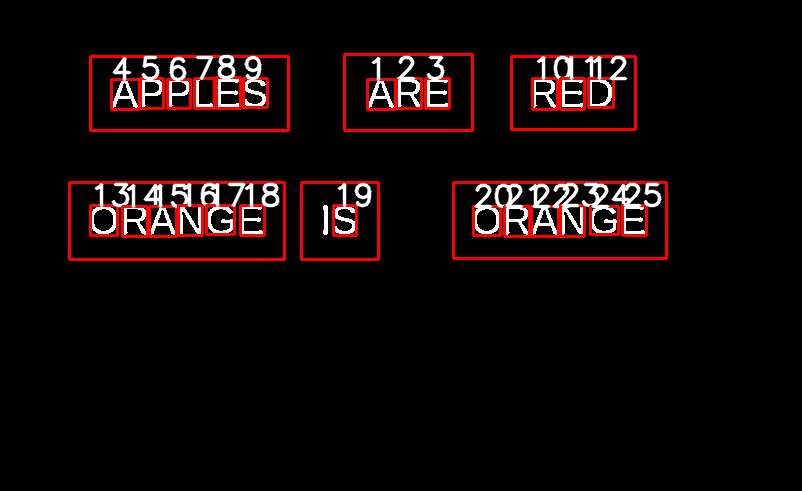


Figure 1 : Disordered Word Rectangles

The words “ARE” and “APPLES” in Figure 8 appear to be on the same line on visual inspection but the bounding box for “ARE” is slightly here than “APPLES” resulting in the wrong order of words being printed out finally.

To fix such issues, for sorting we use a custom comparator which is defined as follows:

def compare(x,y):

if(x<0.9\*y):

return -1

elif(x>0.9\*y):

return 1

else:

return 0

This allows us a 10% variance in equivalence while sorting effectively fixing the problem.

# QUANTITATIVE EVALUATION

The accuracies attained using different combinations of features for computer generated text is as follows :

TABLE I

**Features Used vs Accuracy Attained for a 26 Class Classifier of Alphabets A-Z**

|  |  |
| --- | --- |
| Feature Used | Accuracy Attained |
| Mean X Value | 0.45 |
| Mean Y Value | 0.36 |
| Mean X and Y Value | 0.72 |
| Black Islands | 0.09 |
| Mean X and Y Value + Black Islands | 0.81 |
| Grid Vector | 0.91 |
| Hu Moments | 0.54 |
| All Combined | 0.91 |

As seen from the data, most of the features individual themselves have poor performance except Hu moments and grid which tend to have the highest accuracy. Grid features in fact lead to higher performance compared to Hu moments. Thus, this is a very good descriptor for character images.

The accuracies attained using different combinations of features for Handwritten text is as follows :

TABLE II

**Features Used vs Accuracy Attained for a 26 Class Classifier of Alphabets A-Z**

|  |  |
| --- | --- |
| Features Used | Accuracy Attained |
| Mean X Value | 0.2 |
| Mean Y Value | 0.4 |
| Mean X and Y Value | 0.4 |
| Black Islands | 0 |
| Mean X and Y Value + Black Islands | 0.2 |
| Grid Vector | 0.6 |
| Hu Moments | 0 |
| All Combined | 0.6 |

The n used in grid must not be too small else it loses a lot of the information. If it is too large, it can lead to overfitting. Hence, we need to decide on the optimal size. A plot was made for the test image for n ranging from 1 to 30 in steps of 1. Only grid feature vector was used for this classifier.

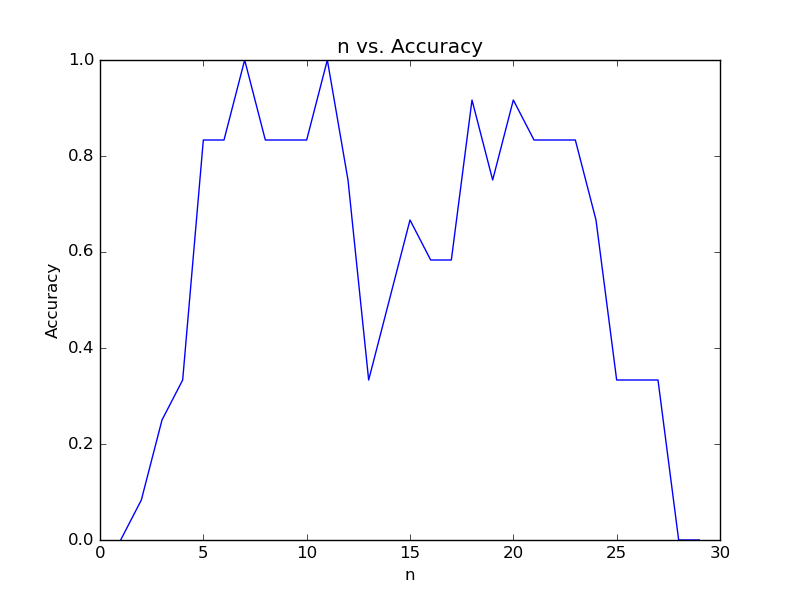


Figure 1 : n vs Accuracy Plot

As seen, the maximum accuracy is attained for the value of n=7 and n=11. For low values and high values of n we attain very low accuracy. The optimal n value will differ for images of different size, but we limit ourselves to images of similar size and hence pick value of n as 7.

# RELATED WORK

In [1], methods to further improve detection of text regions is discussed. The author points out that a majority of the possible character regions detected by MSER turn out to be false positives. However, the advantage here compared to other methods is that it manages to detect most of the text regions even if the image has poor contrast, noise and low resolution. A set of filters based on size, aspect ratio, complexity, texture and border energy can be used to eliminate the false positives.

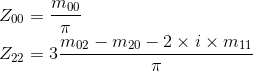
In [7], Peter Norvig talks about building a spell checker based on Bayes' Theorem and probability of occurrence of each word. While this is a good approach for a normal spell check, this doesn't work well in our case since we need corrections to words that are visually similar and not on the basis of how frequently other similar words occur.

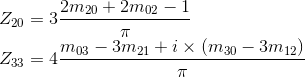
One possible improvement that can be done here is, once a word is not found in a dictionary, we can try to make substitutions for visually similar characters and check if that occurs in a dictionary. For example consider the image below and output generated.

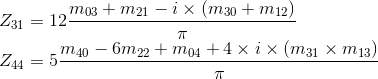
Consider the image in Figure 8. Here, ORANGE is detected as CRANGE in both the cases. O is visually similar to C and 0, so we can make check for occurrence of 0RANGE and ORANGE in a dictionary and hence ORANGE will be successfully returned.

In [5], other moments that might be useful for OCR are discussed. It is also emphasized that Hu moments are very sensitive to noise and their invariance is maintained only in a continuous image. Zernike moments are suggested as an alternative which are less sensitive to noise [8]. They are Zernike moments and affine moments.

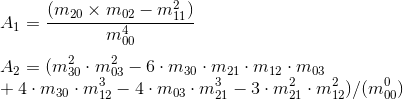
The 1st 6 Zernike Moments are defined as follows :







The Affine moments are invariant to affine transformations, rotation and translation [9]. They are defined as follows :





# SUMMARY AND CONCLUSIONS

The accuracy of computer generated text is 0.60 which is lower than 0.91 which is lower and expected. This is because the various properties used in feature vector are not uniformly maintained in hand-written text.

For example, consider the Q's in the image above. The black island property is , and in the respective images. The positioning and orientation of each character in the box is also slightly different and not centered. Hence, this results in reducing the effectivenes of mean x and y values.

References

1. Yin, X.C., Yin, X., Huang, K. and Hao, H.W., 2014. Robust text detection in natural scene images. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 36(5), pp.970-983.
2. Bradski, Gary. "OpenCV." Dr. Dobb’s Journal of Software Tools (2000).
3. <http://code.opencv.org/projects/opencv/wiki/MSER>
4. Sezgin, Mehmet. "Survey over image thresholding techniques and quantitative performance evaluation." *Journal of Electronic imaging* 13.1 (2004): 146-168.
5. Lacrama, Dan L., and Ioan Snep. "The use of invariant moments in hand-written character recognition." *arXiv preprint arXiv:0904.3650* (2009).
6. Cheng, Chun-Tian, et al. "Heuristic Methods for Reservoir Monthly Inflow Forecasting: A Case Study of Xinfengjiang Reservoir in Pearl River, China."*Water* 7.8 (2015): 4477-4495.
7. <http://norvig.com/spell-correct.html>
8. Khotanzad, Alireza, and Yaw Hua Hong. "Invariant image recognition by Zernike moments." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 12.5 (1990): 489-497
9. . Flusser, Jan, and Tomáš Suk. "Affine moment invariants: a new tool for character recognition." *Pattern Recognition Letters* 15.4 (1994): 433-436.
10. Dhandra, Basanna V., R. G. Benne, and Mallikarjun Hangarge. "A single euler number feature for multi-font multi-size Kannada numeral recognition."*arXiv preprint arXiv:1111.4290* (2011).
11. https://www.kaggle.com/shilparai/digit-recognizer/96-accuracy-with-nearest-neighbour/discussion
12. [http://scikit-learn.org/stable/auto\_examples/neighbors/plot\_classification.html#example-neighbors-plot-classification-py](http://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html" \l "example-neighbors-plot-classification-py)
13. [http://docs.opencv.org/2.4/modules/imgproc/doc/structural\_analysis\_and\_shape\_descriptors.html#id1](http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html" \l "id1)

1. [↑](#footnote-ref-2)