

DIP Project Report

LaTeX Code Generation from Printed Equations

Team Escher

Kritika Prakash¹ and Karthik Chintapalli²

¹20161039

²201501207

2nd November 2018

1 Introduction

To create a system that takes a photograph, scan or screenshot of a printed mathematical equation and produces a valid \LaTeX markup code representation to be able to generate the equation.

1.1 Problem Statement

Working with lengthy involved mathematical equations can be cumbersome, tedious and error-prone. Mathematical equations in the printed form are not easily reproducible in new \LaTeX documents. This is because, once a \LaTeX document has been rendered, the underlying producer's code to recreate it is inaccessible.

1.2 Motivation

\LaTeX is a typesetting system that is extremely useful for technical documents, in particular mathematical equations. However, once rendered, the output cannot be modified without access to the underlying code. Re-writing code for lengthy equations is time consuming and prone to errors. Through this project, we aim to enable a user to take a photograph/screenshot of a printed equation and produce the corresponding \LaTeX markup code to generate the equation.

1.3 Overview

High-level view of the pipeline:

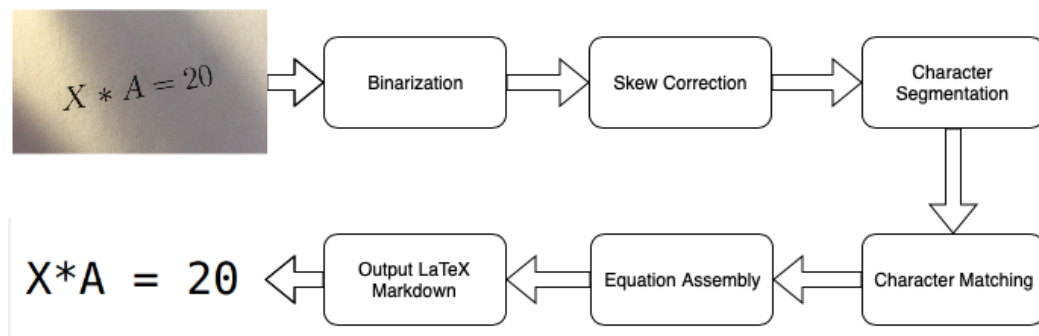


Figure 1: Image to \LaTeX Markdown System Pipeline

1.4 Scope

The scope is defined by the kind of mathematical operations and expressions the system will be able to recognize and recreate. The equation is assumed

to be the primary data in the input image, as opposed to extraction of the mathematical equation from an entire page full of content other than the equation.

2 Solution Approach: Baseline

The Latex Code Generation from Printed Equations Project aims to automatically generate valid LaTeX expressions for a photograph, scan or a screenshot of a printed mathematical equation.

2.1 Page Optimization

2.1.1 Binarization

The input RGB image is first converted into a binary image for processing. The input images considered are either photographs or screenshots. Photographs generally contain shadows and hence, are more challenging to threshold. In order to differentiate between the two image types, an image is first converted to grayscale and the proportion of pixels that are gray (having an intensity value between 15 and 240) is extracted. If this proportion is less than 0.1, the image are treated as scanned images or screenshots. The other images are treated as photographs. The binarization method is described for each of the two categories of images.

- Screenshots and Scanned Images: It is assumed that these images do not require lighting correction. Hence, it suffices to perform Otsu's thresholding on these images.

- Photographs: Photographs generally contain uneven lighting and various page imperfections. Hence, adaptive thresholding is performed, followed by additional noise removal.

First, high-resolution images are blurred with a Gaussian filter to smoothen edges and remove high frequency noise. This pre-filtering is very beneficial for character recognition. in high-resolution images. However, this is not done for low-resolution images as it does not preserve sufficient details due to the low number of total pixels in the image. The threshold for high resolution is taken to be 2000×1000 pixels, and images which are above this threshold are smoothed using a 10×10 Gaussian filter with standard deviation of 3. Adaptive thresholding is performed for the binarization in order to compensate for uneven lighting. The window size used is equal to $1/60^{\text{th}}$ of the smaller dimension of the image. The window size is chosen such that it can handle uneven lighting while still being small enough to preserve the characters.

After the binary image is obtained, it is inverted so that the foreground is now the text. Noise removal is performed using a morphological opening operation. Then, in order to close gaps in character edges, a closing operation is performed. Finally, hole filling is performed to fill the gaps within characters. The hole size threshold is chosen carefully so as to avoid filling holes that are actually part of characters. The final binarized output is obtained by restoring the original parity.

2.1.2 Skew Correction

The next step is to correct the orientation of images so that the equation is horizontally aligned. In order to do this, the dominant orientation needs to be determined. To compute the dominant orientation, the Hough transform of the image is computed. Generally, most equations have multiple horizontal lines, such as fraction bars, equal signs, and negative signs. Based on this fact, it is reasonable to assume that the dominant orientation is given by the correct, horizontal orientation. To prevent large magnitude peaks given by long diagonal lines, such as the diagonal division bar, from being considered as the dominant orientation, the mode of the top four magnitude Hough peaks is chosen.

There are some other details to be considered while implementing this. While an image could be rotated by a small positive angle, the orientation in Hough space is given as the angle between the normal and the horizontal axis, which is the complement of the required angle. This can be corrected by subtracting 90° . After deskewing, edge softening is performed to smoothen edges that are jagged due to rotation. Edge softening is performed using an opening operation.

2.2 Character Recognition

2.2.1 Character Segmentation

The next step is to extract each of the individual characters within the equations. The output of the de-skewing step is provided as input to the segmen-

tation step. The chosen approach is based on finding centroids and bounding boxes of connected regions in edge maps, because it aids in extracting characters that are surrounded by other characters.

Firstly, the edge map is obtained by performing binary erosion on the inverted, de-skewed image and then XORing the result with the original image. The result is a white edge map on a black background. Then, for each connected component within the edge map, the centroid, bounding box and convex hull are extracted. A significant fraction of all the characters can be extracted this way. However, there are several edges that have to be handled. For example, the two inner loops in the capital letter *B* will also be segmented as separated characters by this method. To handle this, we check if a character's convex hull is fully contained within another convex hull. If so, the edge map is examined to determine if the inner character is fully surrounded by the outer character. If this is the case, the inner segmentation can safely be discarded. The inner segmentation can't be discarded just on the basis of the convex hull because of cases such as equations with a square root symbol where many of the characters underneath the square root are completely within its convex hull.

Lastly, for each extracted bounding box, only the largest single character within the bounding box is kept, so that only the character of interest is extracted. For example, in cases such as the square root, the characters within the square root shouldn't be extracted as a part of the square root

symbol.

2.2.2 Character Identification

To perform character identification, we need to build a feature vector corresponding to each extracted character. The feature vector is then used to match the segmented character with characters in the database. The features are chosen such that they are invariant to translation, scaling, and rotation. The chosen features are - the normalized central moment of inertia, circular topology, and Hu Invariant Moments.

The first element in the vector is the normalized central moment of inertia. The central moment of inertia is invariant to translation and rotation. Further, it can be made invariant to scale by normalization. The central moment of inertia can be calculated as:

$$I_N = \sum_{i=1}^N \frac{((x_i - c_x)^2 + (y_i - c_y)^2)}{N^2} \quad (1)$$

where N is the total number of pixels within characters in the image, and c_x and c_y are the coordinates of the character's centroid. The next fifteen elements of the feature vector are extracted from circular topology. Since circles are invariant to rotation, the topology of a segmented character along a circular path is invariant. First, k equally spaced concentric circles that are centered at the characters centroid are chosen. Then, the spacing of the circles is determined by finding the maximum distance from the centroid to an edge of the character and then dividing by $k + 1$ where k is the number of

circles. In our model, we choose $k = 8$ based on empirical results. The first eight elements of these topology elements are the number of times each circle crosses the segmented character. It should be noted that only the first k out of $k + 1$ circles are considered. This is because the outer circle which is on the edge of the character, creates topological aliasing errors due to imperfections on the character edges. The counts are obtained by unwrapping each circle into a line segment and then counting the number of black/white segments.

The last seven topology elements measure the spacing between the character crossings for each circle to differentiate between characters that have equal numbers of intersections. We take the two longest black pixel arcs from each circle, find the difference between them, and normalize by the circumference of the circle.

$$D_i = \frac{arc_1 - arc_2}{circumference} \quad (2)$$

We do this for the $k - 1$ largest circles. The innermost circle is always either completely background or completely foreground, and hence is ignored. Finally, we calculate the Hu Invariant Moments to get the last six features of the feature vector. The seven Hu moments are chosen as they are invariant to translation, scaling, and rotation. However, the first Hu Moment is essentially the same as the normalized central moment of inertia, which is the first feature. Hence, we only use the last six moments. With μ_{pq} as the central moment of order p and q . The seven Hu Moments are listed below for reference:

$$\begin{aligned}
\eta_{pq} &= \frac{\mu_{pq}}{\mu_{00}^\gamma}, \quad \gamma = 1 + \frac{p+q}{2} \\
H_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
H_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
H_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} - \eta_{03})^2 \\
H_5 &= (\eta_{30} - 3\eta_{12})^2(\eta_{30} + \eta_{12})^2[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\
&\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
H_6 &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
&\quad + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{12} + \eta_{03}) \\
H_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\
&\quad - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]
\end{aligned}$$

Figure 2: The 7 Hu Moments

2.2.3 Character Matching

A predefined character palette that contains the possible characters that can be identified, is built. Then, a character template database is created by pre-calculating the feature vectors for every character in the palette and associating them with their respective L^AT_EX code. Our character palette consists of 116 unique characters. Each segmented character in the given equation is then matched to the closest character in the template database through a nearest neighbor classifier using Manhattan distance. Manhattan distance worked the best compared to other distances such as Euclidean distance and cosine similarity. Finally, the best matched character is output

as the detected character. The result of the matching algorithm along with the centroid and bounding box of the segmented character with respect to the original equation are then passed to the final equation assembly module.

2.3 Equation Assembly

The basic idea in this step is that each equation is assembled sequentially from left to right using each recognized character's bounding box. As we process each character, we also keep track of a "previous" centroid to detect the presence of superscripts and subscripts. Note that this is not always the centroid of the previous matched character. For example, if the previous character were a part of a fraction, we need to consider the centroid of the fraction bar instead. The complete equation assembly algorithm is as follows:

2.3.1 Assembly Algorithm

If the current character's bounding box does not overlap with any subsequent bounding boxes, we add the character to the equation directly based on the following rules:

- If the bottom-left corner of the bounding box is higher than the "previous" centroid, it is either a superscript or the first character after the end of a subscript. A "subscript" flag is used to differentiate between the two cases.
- If the top-left corner of the bounding box is lower than the "previous" centroid, it is just the inverse of the previous case.

- If the character is a \LaTeX control sequence, ' \backslash ' is pre-appended to the character and a space is appended at the end of the character. The result is added to the equation.
- Else, the character is added directly without any modifications.

Else, if the current character overlaps with subsequent bounding boxes, we check for specific cases:

- If the character is a square root symbol, we select all of the subsequent overlapping characters and recursively assemble a sub-equation. Then, the appropriate \LaTeX sequence is appended to the final equation.
- If the character is '-' and only overlaps with a single other '-', we append '=' to the equation.
- Else, if the character is '-' and does not only overlap with a single other '-':
 - If the width of the '-' is equal to a significant portion of the overall width of the overlapping bounding boxes, the overlapping region is recognized as a fraction. The characters are then divided into denominator and numerator characters. Finally, these two sets are recursively assembled into sub-equations. The \LaTeX markdown corresponding to the fraction is then assembled and appended to the final equation.
 - Else, we recognize it as a negative sign.

- Finally, we assemble equations for summations, integrations, products etc. by recursively assembling the equations of the lower and upper limits.

It should be noted that this algorithm is not complete by any means as the number of symbols are limited to the character palette. Elements such as matrices are not handled appropriately.

3 Alternate Approaches

3.1 Image skew correction

The skew correction method described above is based on using the Hough Transform to find lines with maximum evidence in the image. This created issues in low resolution images and images which lacked long lines in the horizontal direction. Hence, we also tried an alternate approach based on Principal Component Analysis. In the PCA-based approach, we treat each foreground pixel as a point on a 2-D Euclidean plane. Then, we perform Principal Component Analysis on this set of points and obtain the direction of maximum variance (principal component). The idea is that the principal component corresponds to the dominant direction in the image which is generally the horizontal direction.

While this performed better than the original method for images of low resolution, the average error of the extracted direction with respect to the desired direction was higher using this method, and hence it was not added to the

final pipeline.

4 Example

In this section, we provide a walk through of the algorithm via an example. Consider Fig. 3 as the skewed input image of a printed equation under poor lighting conditions. Upon binarization and deskewing the image, we obtain Fig. 4. The target characters of the equation are shown in Fig. 7. We then invert the cleaned image, and identify each character in the equation by identifying its bounding box, centroid and convex hull, as shown in Fig. 5. Using these constructs, we extract and then match the characters of the equation as shown in Fig. 6.

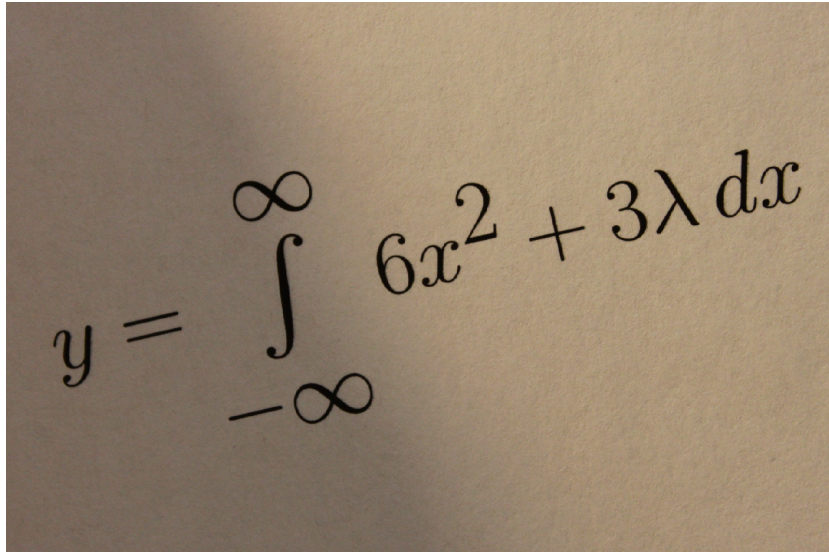

$$y = \int_{-\infty}^{\infty} 6x^2 + 3\lambda dx$$

Figure 3: Picture of Example Equation

$$y = \int_{-\infty}^{\infty} 6x^2 + 3\lambda dx$$

Figure 4: Cleaned Example Equation Image

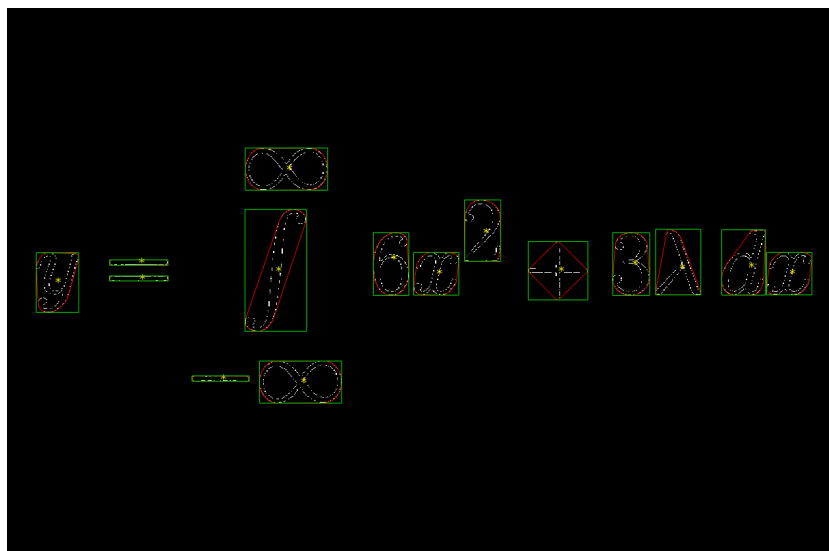


Figure 5: Bounding Box, Convex Hull and Centroid of Characters

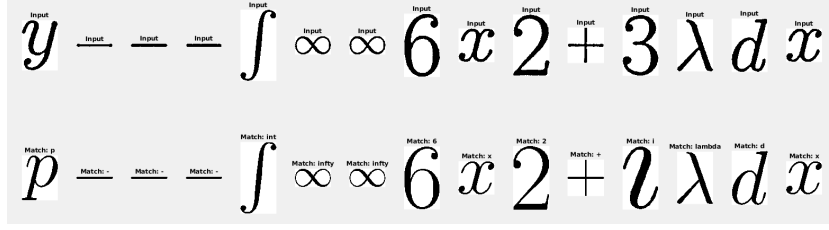


Figure 6: Matching Characters

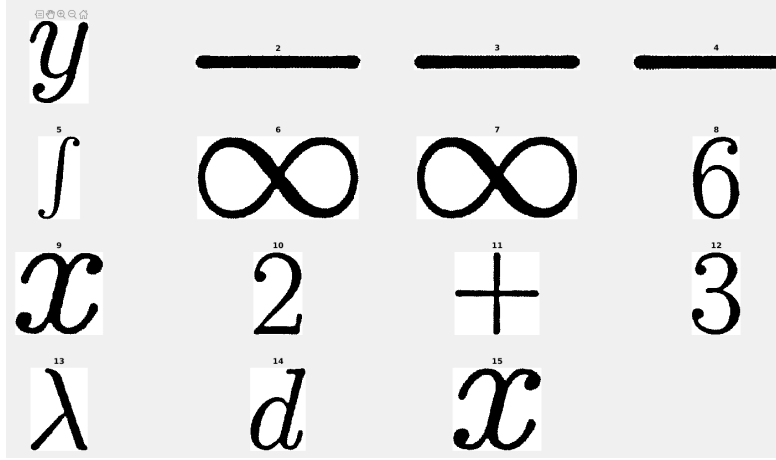


Figure 7: Characters in the Example Equation

5 Experiments

We conduct experiments for different approaches used for character matching, as shown in Tables 5, 5 and 5. We use three different kinds of distances - Manhattan, Euclidean and Cosine, for K-Nearest Neighbour (KNN) classification algorithm, and observe that KNN with Manhattan distance outperforms the other two metrics - on an average, it correctly matches 77% of the characters. With Euclidean distance, it correctly matches 69% of the characters, and with Cosine distance, it correctly matches 52% of the characters.

Image	Correct symbol count	Total symbol count	Percent correct	Reason for Error
Image 1	6	7	85.71	-
Image 2	5	7	71.43	Low resolution
Image 3	5	9	55.55	Poor de-skewing
Image 4	8	11	72.73	Low resolution
Image 5	12	19	63.16	Poor de-skewing
Image 6	13	15	86.67	-
Image 7	10	12	83.33	-
Image 8	19	22	86.36	-
Total	78	102	76.47	-

Table 1: Results for KNN based on Manhattan Distance

Image	Correct symbol count	Total symbol count	Percent correct
Image 1	5	7	71.43
Image 2	5	7	71.43
Image 3	5	9	55.55
Image 4	7	11	63.63
Image 5	12	19	63.16
Image 6	13	15	86.67
Image 7	10	12	83.33
Image 8	14	22	63.63
Total	71	102	69.6

Table 2: Results for KNN based on Euclidean Distance

Image	Correct symbol count	Total symbol count	Percent correct
Image 1	4	7	57.14
Image 2	5	7	71.43
Image 3	4	9	44.44
Image 4	5	11	45.45
Image 5	9	19	47.36
Image 6	11	15	73.33
Image 7	9	12	75
Image 8	6	22	27.27
Total	53	102	51.96

Table 3: Results for KNN based on Cosine Similarity

6 Results

The entire solution pipeline performs character recognition with an accuracy of 99% on images which have no skew or lighting problems. On input images which are de-skewed and require lighting correction, the best method is the baseline along with Manhattan distance based k-Nearest-Neighbours classification for character matching.

Our heuristic based method of thresholding runs approximately twice as fast as Adaptive Thresholding using Otsu's method. Using any subset of features from our feature vector performs worse as compared to using the entire 22-feature vector.

7 Analysis of Results

7.1 Page Optimization

7.1.1 Binarization

The binarization procedure described works well for most cases. However, there are issues with the hole filling procedure. In cases where the equation occupies a smaller fraction of the entire images, the holes within characters are similar in size to holes due to noise. Hence, they are identified as holes and are filled.

7.1.2 Skew Correction

The skew correction procedure works well for high resolution images as there is sufficient evidence for the dominant direction. However, for low-resolution images where there are dominant directions other than the desired direction, the results are not as expected.

The PCA-based method for de-skewing resolves the above issue for low resolution images, but it has the issue of the dominant direction being along the diagonal of the image rather than along the horizontal direction. Hence, we stick with the original method.

7.2 Character Recognition

7.2.1 Character Segmentation

The character segmentation procedure works well on most characters when they are non-overlapping and have just one continuous region. In cases where there are overlapping characters, extra steps are taken to properly segment the individual characters. Examples of such cases include the square root symbol and the characters under it.

However, there are several cases where it does not perform as desired. Since, the method of segmentation is region-based, characters such as the English letter 'i' are segmented into two different characters (the stem and the dot on top). A method of segmentation that takes some semantics into account would resolve this issue.

7.2.2 Character Identification & Matching

Based on the distance metrics for the k-Nearest-Neighbours algorithm used for character matching, we observe that certain distances perform well on certain kinds of characters. For example, cosine similarity based KNN identified some characters which neither Manhattan, nor Euclidean distance based KNNs were able to match correctly. The two major reasons for poor matching results were:

- Large degree of skew in the input image (failure of the de-skewing algorithm)
- Low resolution of the input image.

7.2.3 Equation Assembly

The equation assembly algorithm is handcrafted on a case-to-case basis. As a result, it works well for the limited list of character patterns that were considered. Examples of patterns that fail are matrices.

8 Discussion

8.1 Limitations

8.1.1 Binarization & Deskewing

The Binarization algorithm used turns out to be sufficient for almost all cases. The algorithm is good at deciding between Otsu's Global Thresholding algorithm and Local Adaptive Thresholding.

However, the deskewing algorithm has several limitations. The algorithm's performance is poor in the following cases:

- The image resolution is low and hence, there isn't sufficient evidence for the dominant direction.
- The image contains symbols such as integrals or fraction bars which provide false evidence for the dominant directions.

8.1.2 Character Matching

The character matching algorithm is fundamentally limited by the fact that it relies on a palette of symbols. This means that a manual addition of symbols is required to improve the algorithm. Another limitation is that since the algorithm uses features that are scale invariant, the capital and small versions of certain English letters are detected as equivalent.

8.1.3 Equation Assembly

The equation assembly module is the least extensible module in the entire pipeline. The current algorithm is entirely rule-based and the rules are hand-crafted on a case-to-case basis. If more complex structures such as matrices are to be included, the algorithm will have to be modified significantly.

8.2 Future Work and Improvements

- Expand the library of \LaTeX constructions supported.

- Extend the work to any sort of \LaTeX text or equation.
- Improve the quality of binarization as well as skew correction.
- Extend the work to handwritten equations (difficult).
- Deploying the application as a service.

9 GitHub Repository

<https://github.com/Kritikalcoder/Latex-Generation-from-Printed-Equations>

10 Task Assignment

S.No	Stage		Task	Member
1	Baseline Model	Page Optimization	Image Thresholding	Karthik
2			Binarization	Karthik
3			Skew Correction	Kritika
4		Character Recognition	Character Segmentation	Karthik
5			Character Identification	Kritika
6			Character Matching	Kritika
7		LaTeX Compilation	Equation Assembly	Both
8	Improvement	Testing		Both
9		Improving Baseline Model		Both
10		Comparative Study		Both
11	Presentation	Preparing Presentation		Both

11 References

- LaTeX Generation from Printed Equations, Jim Brewer, James Sun, Stanford University
- B. Potocnik, "Visual pattern recognition by moment Invariants" in Multimedia Signal Processing and Communications, International Symposium ELMAR-2006
- M. Usman Akram, "Geometric Feature Points Based Optical Character Recognition", in IEEE Symposium on Industrial Electronics & Applications (ISIEA), 2013
- V. Muralidharan. (2015, Nov.) Hu's invariant momentss. [Online]. Available: <http://www.mathworks.com/matlabcentral/fileexchange/52259-hu-s-invariant-moments>