Digital Convergence : Digitization of Doctors Hand Written Prescriptions
Team 9

***Abstract*— This paper attempts to solve handwritten text recogniton, specifically in medical prescriptions. Instead of using the more prevalent method of directly using word level recognition, we attempt to split the words into characters, which is the hardest part of handwriting recogniton. We probabilistically split the word into characters, using not just local but global optimum and come up with a set of words with corresponding probabilities and then use dictoinary based models for choosing the right one out of the set. To reduce the complexity of there methods we came up with some heurestics.**

I. INTRODUCTION

The objective of DigiCon is that let allow a doctor to write his prescriptions the conventional way (i.e., using their pen and paper). From the scanned version of the prescription, a handwritten text recognition is followed to capture the data (name of the patient, symptoms, findings, prescription of medicine, tests, advice, etc.) written by the doctor. Since, the accuracy rate of the state-of- the-art hand written character reorganization is not still up to the acceptable level, we propose to apply an error correction mechanism to reduce the errors. The solution does not oppose the age-old convention and affordable as it is mostly a software solution with a minimum hardware requirement.

II. IMAGE PRE-PROCESSING

A. Rescaling of input prescription image

The scanned input may vary in dimensions and resolution and the processing algorithms depend highly on the resolution, both their complexity and accuracy. Hence, we rescale the input image to a proper resolution(768px in width) maintaining the aspect ratio and then treat this as the input to all our algorithms.

B. Denoising

An input image coming from a scanner or camera, consisting of text will mostly have the following types of noise:

- Salt and pepper noise
- Uneven lighting
- Light and contrast gradient

We hereby make a valid assumption that a scanned image won't be skewed. Hence we don't bother about skew correction. To correct the above types of noise we first apply erosion with dynamic window size which eliminates the salt and pepper noise. As for the rest types of noise the binarization step takes care of it.

C. Binarization

For Binarization we first find the gradient of the gray scale image and then apply the thresholding.

- For gradient: The Sobel Operator is a discrete differentiation operator. It computes an approximation of the gradient of an image intensity function. The Sobel Operator combines Gaussian smoothing and differentiation Assuming that the image to be operated is :
  1) We calculate two derivatives:
     a) **Horizontal changes**: This is computed by convolving with a kernel with odd size. For example for a kernel size of 3, would be computed as:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$

     b) **Vertical changes**: This is computed by convolving with a kernel with odd size. For example for a kernel size of 3, would be computed as:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

  2) At each point of the image we calculate an approximation of the gradient in that point by combining both results above:

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

- For thresholding : In global thresholding, we used an arbitrary value for threshold value, right? So, how can we know a value we selected is good or not? Answer is, trial and error method. But consider a bimodal image (In simple words, bimodal image is an image whose histogram has two peaks). For that image, we can approximately take a value in the middle of those peaks as threshold value, right ? That is what Otsu binarization does. So in simple words, it automatically calculates a threshold value from image histogram for a bimodal image.
  Otsu's algorithm tries to find a threshold value (t) which minimizes the weighted within-class variance given by the relation :

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

$$q_1(t) = \sum_{i=1}^{t} P(i) \quad \& \quad q_1(t) = \sum_{i=t+1}^{I} P(i)$$

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)} \quad \& \quad \mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{q_2(t)}$$

$$\sigma_1^2(t) = \sum_{i=1}^{t} [i-\mu_1(t)]^2 \frac{P(i)}{q_1(t)} \quad \& \quad \sigma_2^2(t) = \sum_{i=t+1}^{I} [i-\mu_1(t)]^2 \frac{P(i)}{q_2(t)}$$

It actually finds a value of t which lies in between two peaks such that variances to both classes are minimum.

D. ROI detection

For localizing words in the image we find the contours after applying Morphological Transformation. Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity.

E. Skew correction

For skew correction bounding rectangle is found with minimum area for each localized words in the image, so it considers the rotation also. We find the following details - (center (x,y), (width, height), angle of rotation) for minimum area rectangle.

Both the rectangles are shown in a single image. Green rectangle shows the normal bounding rect. Red rectangle is the rotated rect. For correcting skew we calculate an affine

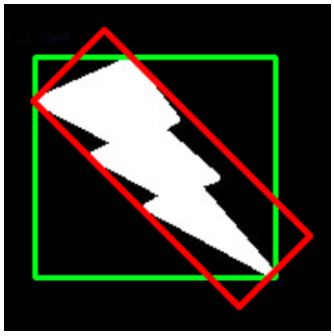

Fig. 1. Recognition tree

matrix of 2D rotation for each localised word from the slope of minimum area rectangle. we calculate the following matrix:

$$\begin{bmatrix} \alpha & \beta & (1-\alpha) * centre_x - \beta * centre_y \\ -\beta & \alpha & \beta * centre_x - (1-\alpha) * centre_y \end{bmatrix} * I$$

$$where:$$

$$\alpha = scale * \cos(angle)$$

$$\beta = scale * \sin(angle)$$

Then we transform the each localized image using the respective specified matrix:

$$dst(x,y) = src(M_1 1x + M_1 2y + M_1 3, M_2 1x + M_2 2y + M_2 3)$$

III. Handwritten Text Recognition

A. State-of-the-Art

State of the art handwriting recognition methods generally use deep neural networks trained on a handwritten dataset of words such as the IAM dataset. These methods work well when the handwriting does not vary much.

B. Improved - I

Our method tried to solve the text recognition in a probabilistic method. The hardest part of handwritten text recognition is the boundary detection of each character in the text. Since the handwritten text is almost always continuous(cursive) it becomes hard to decide where to actually split for each character separation. For example: In the below image if we split at line-1,2 and 3 we get i, n and m as the recognized character using the state-of-the-art OCR methods.

We propose to solve this problem differently as follows. We decide a small window size(**windowWidth**) and split each word image into several parts of this width and call each such partition a window now. We also decide a **maxWindowAggregation** as the number of windows each character is expected to span.

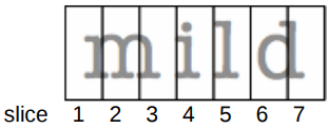

Fig. 2. Splitting word image into windows

Now we start by taking one slice, two slice... upto **maxWindowAggregation** slices and probabilistically detect which character those slices correspond to. And for each of those slices, we detect further characters in a recursive manner as shown in figure below. This all recursion when represented as a tree is referred to as recognition tree.

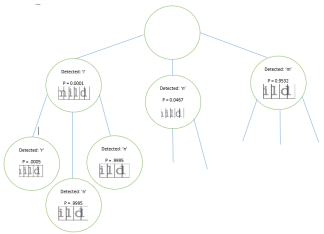

Fig. 3. Recognition tree

At the end of this recursion we get possible words with their probability at the leaf nodes. Now the problem is to decide which of these words is the actual word. That is done with the help of an overall word score for each possible words dependent on the individual character detection probability. This score for our implementation is defined as:

$$Score_{word} = \Sigma probability_{characters}$$

where $probability_{characters}$ is the probability of the detection characters in the path from root node tot hat word's leaf node.

The overall complexity of the above algorithm is of the order of $O(3^n)$ where n = number of windows, which can be too much for $n > 20$. Hence, we apply certain heuristics for

early pruning of the recognition tree which forms the basis of our next method.

C. Improved - II

To improve upon the time complexity of the previous algorithm we came up with two heuristics:

- When we look at the tree what we realize is that some of the path's score goes extremely low much before it even reaches the leaf node. Once, the score of a node reaches very low compared to other nodes at the same level, it doesn't rise much compared to the other nodes on the same level. So it's futile computations to compute any further possibilities for that node. So what we can do is at each level grow the tree further for only the top certain number of nodes based on their score. For our implementations we pick top ten or top ten percent of nodes possible at a level, whichever is minimum.
- Another thing that we realize is that a lot of the paths in the tree doesn't correspond to valid vocabulary words. This could be because of the wrong spelling written by the doctor in the first place. So to take care of this we create a trie of the dictionary and then go on paths which are available in the dictionary only. To take care of the wrong spelling by the doctor we ralax our tree growing to at most two consecutive wrong letter suffix which doesn't exist in the dictionary.

IV. Character level recognition

This works as a subroutine to the previous section, recognizing character in each slice of the word image. Input to this is a small slice of word images and the output is the probability of each character occurring in that slice.

A. DL based probabilistic character recognition

The deep learning character recognition is based on a Convolution Neural Network trained on handwritten characters dataset, Chars74k dataset in this case. It gave an accuracy of about 83%.

V. Sloppy handwriting correction

Doctor's handwritten text can be expected to have high distortions, and the OCR output from Section III is based purely on character recognition, hence has less co-relation with the intended word from the doctor's prescription. Therefore, relaying on the error correction function to search for more matches from the dicitonary is necessary.

We have used a modified corpus which contains 3 seperate dictionaries of Drugs names, Symptoms and English words. A Bag-of-words model is implemented to return the most probable Drug/Sympton/Word based on how commonly used it is. If the word has been identified as Drug or a Symptom a tag is attached to it, based on which higher priority is given to matching the word in the corresponding dictionary.

Output of Improved-II gives us a list of possible words with the corresponding probabilities. The words may or may not be valid dictionary words, so the first step of identifying the right word is to find out the closest valid dictionary words to each of the probable words. The most appropriate word corresponding to that word image is subject to the probabilities of the word obtained from Improved-II and the Levenshtein Distance from the match in the dictionary. This has been modeled as an objective function as follows:

$$Score = prob_{word}/(1 + Levenshtein\_Distance)$$

$$CorrectedWord = argmax(Score)$$

References

[1] Zhang, X., He, K., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. (2016)

[2] Charles Jacobs, Patrice Y. Simard, Paul Viola, and James Rinker: Text Recognition of Low-resolution Document Images ''