Separation and Recognition of Machine Printed and Handwritten Text

Bachelor's Project report for semester 7

Ajay Charan
UG201211002
Dept. of Electrical Engineering
IIT Jodhpur

Table of Contents

Introduction

Text recognition is conversion of printed or handwritten text in images to machine-encoded text. This problem is an important area of research, particularly in data entry from printed records, digitizing printed texts and is also used in machine translation and text mining.

The process of OCR involves several steps including segmentation, feature extraction, and classification. Each of these steps is a field unto itself. A few examples of OCR applications are listed here. The most common for use OCR is the first item; people often wish to convert text documents to some sort of digital representation.

1. People wish to scan in a document and have the text of that document available in a word processor.

2. Recognizing license plate numbers

3. Post Office needs to recognize zip-codes

This project aims to develop an algorithm to achieve separation and recognition of handwritten and machine printed texts in images. Various scenarios are explored, and efforts are made to develop a real-time robust algorithm which could even be applied for scene text recognition.

Problem Specification

The complete scope of Text recognition or Optical Character Recognition (OCR) is vast. In the most general case, images contain several words and symbols. If the image contains both handwritten and machine printed text, the problem of Separation of these two also arises. In general, the texts are separated first and then sent for recognition.

Generally, machine printed text represents the original document and handwritten texts add or alter it. This can be widely used in processing of forms and bank cheques.

In the first part of the project we focus on recognition aspect of the task.

Tasks such as OCR fall into a class of problems known as pattern recognition which deals with classifying the given data. Other Examples of Pattern Recognition:

1. Facial feature recognition (airport security) – Is this person a bad-guy?

2. Speech recognition – Translate acoustic waveforms into text.

3. A Submarine wishes to classify underwater sounds – A whale? A friendly ship?

The Classification Process:

(Classification in general for any type of classifier) There are two steps in building a classifier: training and testing. These steps can be broken down further into sub-steps.

1. **Training**

a. Pre-processing – Processes the data so it is in a suitable form

b. Feature extraction – Reduce the amount of data by extracting relevant information. This usually results in a vector of scalar values. (We also need to normalize the features for distance measurements!)

c. Model Estimation – from the finite set of feature vectors, need to estimate a model (usually statistical) for each class of the training data

2. **Testing**

a. Pre-processing

b. Feature extraction – (both same as above)

c. Classification – Compare feature vectors to the various models and find the closest match. One can use a distance measure.

Literature Review

Word Recognition:

Recognition of handwritten words from unconstrained scanned documents remains to be an open research problem. Complex structures in documents hinder segmentation into lines and words. Additionally the variability in handwriting imposes another level of challenge for word recognition. Figure below depicts the word recognition problem.

Figure: Word recognition problem: Handwritten previously segmented word images are to be recognized as shown

Word recognition can be classified under the following categories

1. **Lexicon driven/Lexicon independent**: Lexicon driven approach uses a fixed length lexicon, and associates each word image in the document with the top ranked matching lexicon. Lexicon independent methods rely solely on automatic character recognition.
2. **Segmentation free/Segmentation based**: Segmentation free methods try and recognize the entire word image using its global features. On the other hand segmentation based methods, rely on breaking the word image into smaller segments identifiable as characters and associate a character label to these segments. Here contextual dependencies between neighboring segments are exploited here using time series models such as Hidden Markov Models (HMMs).

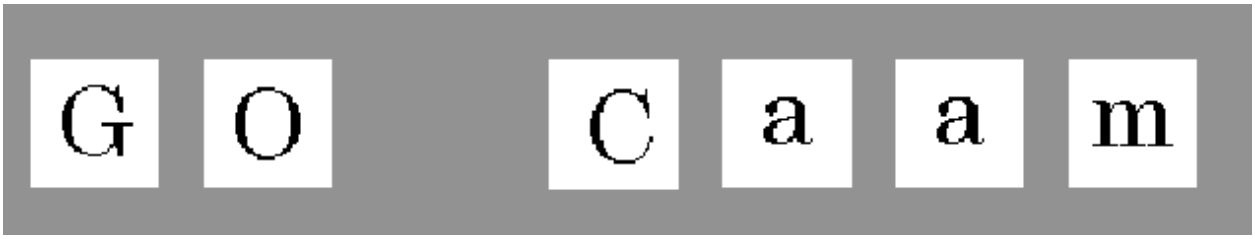
In segmentation based methods, HMMs can be used to capture the spatial dependencies amongst neighboring segments; they are generative models and try to model the joint probability of the data and the labels. Markov models (MEMMs) and other discriminative Markov models based on directed graphical models, can be biased towards states with few successor states (label bias problem). HMMs can be used for segmentation free word recognition; the model captures the transition to the same character label or to the next character in the word. CRFs are also used for recognition at the word level, but this approach leads to a large number of parameters as the model has to learn state parameters for every word in the lexicon and transition parameters for every possible pair of neighboring words in the lexicon. Some methods use a lexicon driven approach based on character segmentation

Approach 1: Template Matching

Template matching or matrix matching, is one of the most common naive recognition methods. Here individual image pixels are used as features. Classification is performed by comparing an input character with a set of templates (or prototypes) from each character class. Each comparison results in a similarity measure between the input characters with a set of templates. One measure increases the amount of similarity when a pixel in the observed character is identical to the same pixel in the template image. If the pixels differ the measure of similarity may be decreased. After all templates have been compared with the observed character image, the character's identity is assigned the identity of the most similar template. Template matching is a trainable process as template characters can be changed.

Methodology

We can compare each segmented set of contiguous pixels directly against a pre-stored library of letters. To make this a reasonable approach we assume that we have prior knowledge of the font size and font type that we are trying to read. For instance consider the Century font, with a couple of letters shown here:



A simple algorithm to decode a letter:

1. Create a three dimensional array of pixels of size Nrows x Ncolumns x Nchars, where Nrows, Ncolumns are large enough that all of the reference letters can fit inside an array of Nrows x Ncolumns pixels. Nchars is the number of characters in the library. For robustness we add extra layers of pixels to each reference letter
2. Extract a block of size Nrows x Ncolumns that contains all the pixels of the same number from the segmented image.
3. Store these pixels in a temporary matrix L, so that their center of mass is located at the center of the matrix. We add layers of extra pixels to the extracted character if the reference letters are padded.
4. Loop through all the library pixels and compute the Frobenius norm of the difference between the extracted letter matrix **L** and the n'th library character, with formula:

$$\|\mathbf{L}-\mathbf{C}^n\|_F = \sqrt{\sum_{i=1}^{i=N}\sum_{j=1}^{j=M}\left(L\left(i,j\right)-C\left(i,j,n\right)\right)^2}$$

5. Find the reference character that yields the smallest Frobenius norm difference.

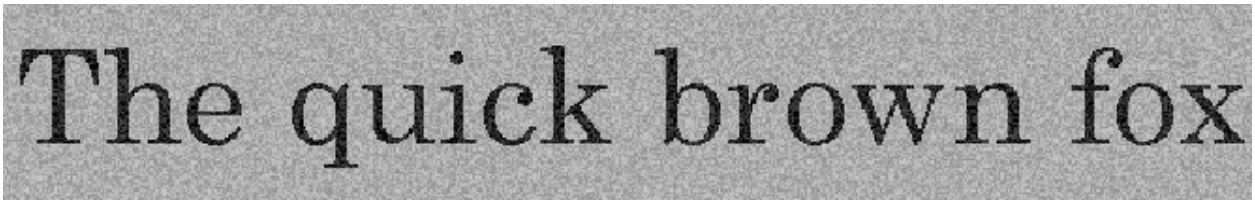
We then optimistically assert that the segmented character will be the n'th reference letter.

6. Move onto the next segmented character until none are left.

Results

Consider this example:

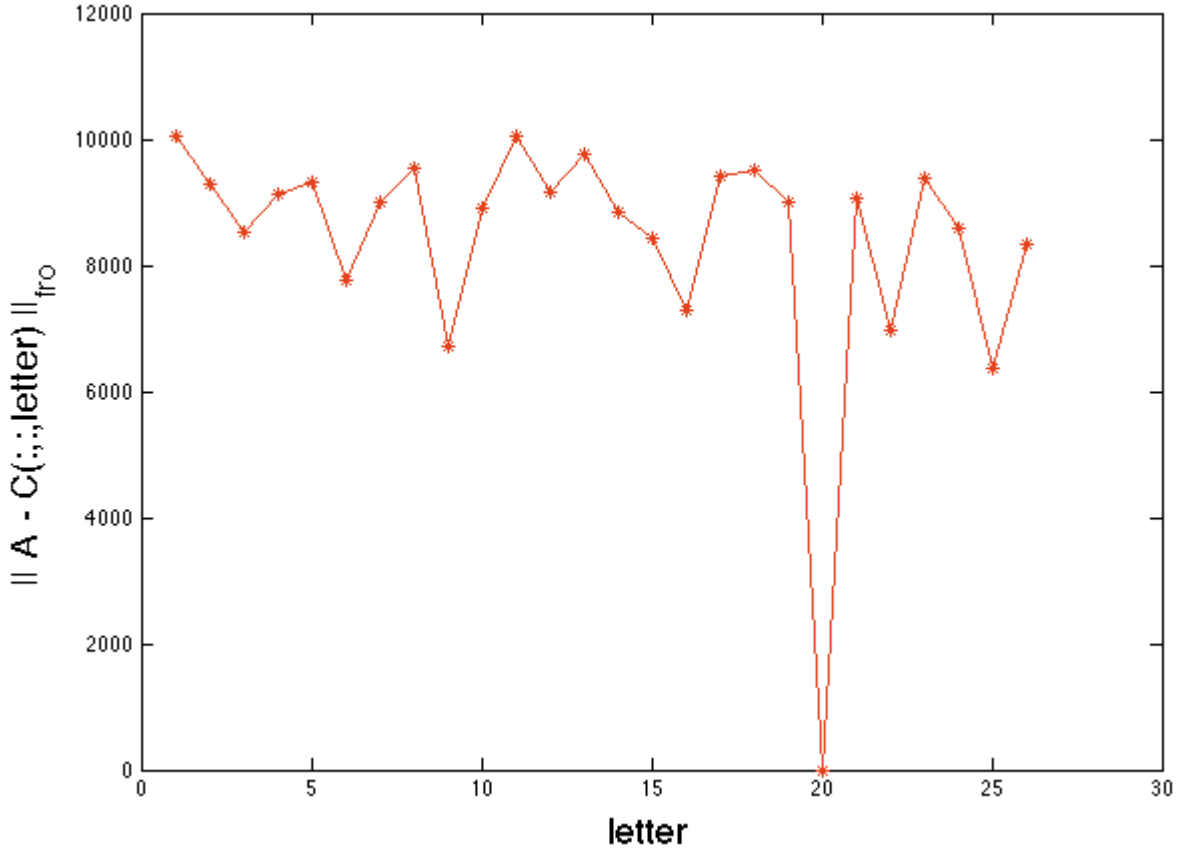
Adding noise to it:



After extracting the first letter, padding it, and storing it in a reference block, it looks like:

This looks exactly the same as the library version

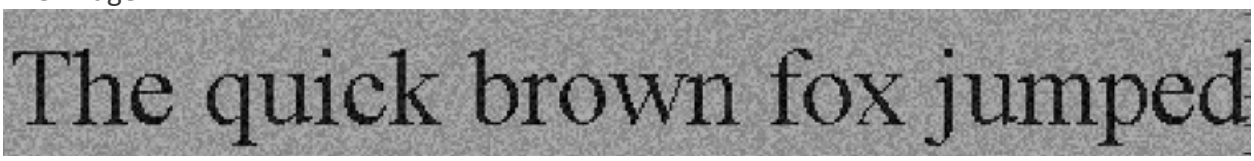
Using MATLAB to compute the distance between these images using the Frobenius norm and checking all the letters



Clearly, the letter 20 (i.e. T) is indeed the best fit.

Then I performed OCR on a piece of text that is in a different font to the library font. I used Arial (a rounded font with relatively few features) to recognize Times which has lots of “features”.

The image:



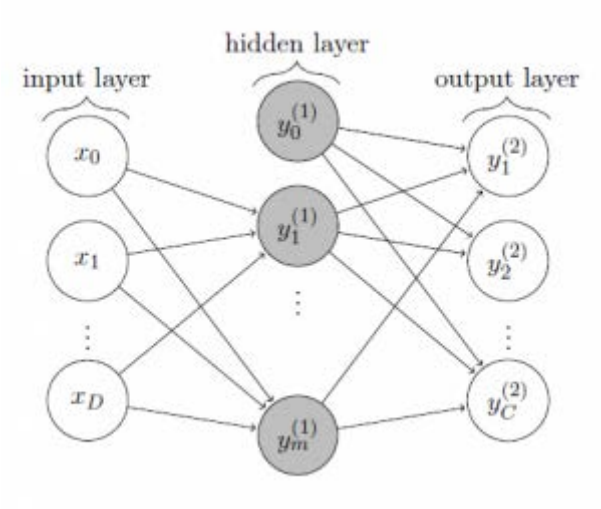
Running the template matching code, these characters are recognized as: **Ihc`u`vLhioorox`umFcJc`**

Approach 2: Neural network for handwritten digit recognition

The MNIST dataset provides a training set of 60,000 handwritten digits and a validation set of 10,000 handwritten digits. The images have a size of 28×28 pixels. We want to train a two-layer perceptron to recognize handwritten digits, i.e. given a new 28×28 pixels image, the goal is to decide which digit it represents.

Methodology

I implemented a two-layer perceptron in MATLAB to the problem of recognizing handwritten digits. For this purpose, the two-layer perceptron consists of *28·28=784* input units, a variable number of hidden units and *10* output units. The general case of a two-layer perceptron with ***D*** input units, ***m*** hidden units and ***C*** output units is shown in figure

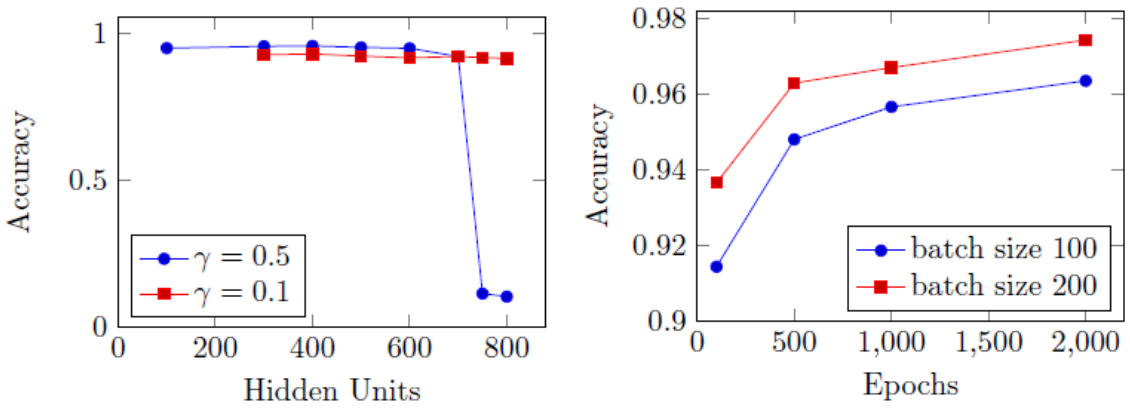


The network is trained using a stochastic variant of mini-batch training, the sum-of-squared error function and the error backpropagation algorithm. The method returns the weights of the hidden layer and the output layer after training as well as the normalized sum-of-squared error after the last iteration.

First the MNIST dataset needs is loaded using the methods `loadMNISTImages` and `loadMNISTLaels`. The labels are provided as vector where the ***i***th entry contains the digit represented by the ***i***th image. We transform the labels to form a 10×N matrix, where N is the number of training images, such that the ***i***th entry of the nth column vector is 1 iff the nth training image represents the digit ***i***–1.

The network is trained using the logistic sigmoid activation function, a fixed batch size and a fixed number of iterations. The training method returns the weights of the hidden layer and the output layer as well as the normalized sum-of-squared error after the last iteration. Then, the numbers of classification errors on the validation set are counted.

Results



(a) 500 epochs with batch size 100.

(b) 500 epochs with learning rate $\gamma = 0.5$.

Figure (a) plots the achieved accuracy depending on the number of hidden units for a fixed learning rate of 0.5. For 750 or more units we need to adjust the learning rate in order to ensure convergence. But this may result in slow convergence such that we get worse results for few hidden units. Figure (b) shows the impact of increasing the number of epochs. As we would expect, the accuracy increases with rising training set.

Approach 3: Convolutional Neural Network for word recognition

As a supervised approach of Deep Learning, Convolutional Neural Network is famous for its noise robustness and translation invariances. It has been widely used to recognize all kinds of patterns, such as hand written digits and human faces. Detection of text and identification of characters in scene images is a challenging visual recognition problem. As in much of computer vision, the challenges posed by the complexity of these images have been combated with hand designed features and models that incorporate various pieces of high-level prior knowledge. CNN on the other hand attempts to learn the necessary features directly from the data as an alternative to using purpose-built, text-specific features or models

Methodology

I extended the existing code for character recognition using single layer CNN classifier described in Coates, Adam, et al. "Text detection and character recognition in scene images with unsupervised feature learning." ICDAR 2011. For character recognition, system proceeds in several stages:

1) Apply an unsupervised feature learning algorithm to a set of image patches harvested from the training data to learn a bank of image features.

2) Evaluate the features convolutionally over the training images. Reduce the number of features using spatial pooling.

3) Train a linear classifier for either text detection or character recognition.

Each of these stages is described in more detail:

A. Feature learning

1) Collect a set of 8x8 grayscale image patches from training data.

2) Apply simple statistical whitening to the patches of the input to yield a new dataset

3) Run a variant of K-means clustering (using inner product as similarity metric so that it yields a dictionary of normalized basis vectors) on it to build a mapping from input patches to a feature vector.

B. Feature extraction

1) We consider 32-by- 32 pixel images and compute the representation described above for every 8-by-8 sub-patch of the input, yielding a 25- by-25-by-d representation.

2) To reduce the dimensionality of the representation we use average pooling: sum up the vectors over 9 blocks in a 3x3 grid over the image, yielding a final feature vector with 9d features for this image.

C. Text detector training

1) We train a binary classifier that to distinguish 32x32 windows that contain text from windows that do not.

2) Training set for this classifier built by extracting from the ICDAR 2003 dataset, using the word bounding boxes to decide whether a window is text or non-text.

3) We then use the feature extraction method described above to convert each image into a 9d feature vector.

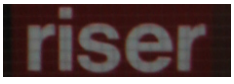
4) These feature vectors and the ground-truth "text" and "not text" labels acquired from the bounding boxes is then used to train a linear SVM. The feature extractor and the trained classifier for detection is used in the usual "sliding window" fashion.

Detection

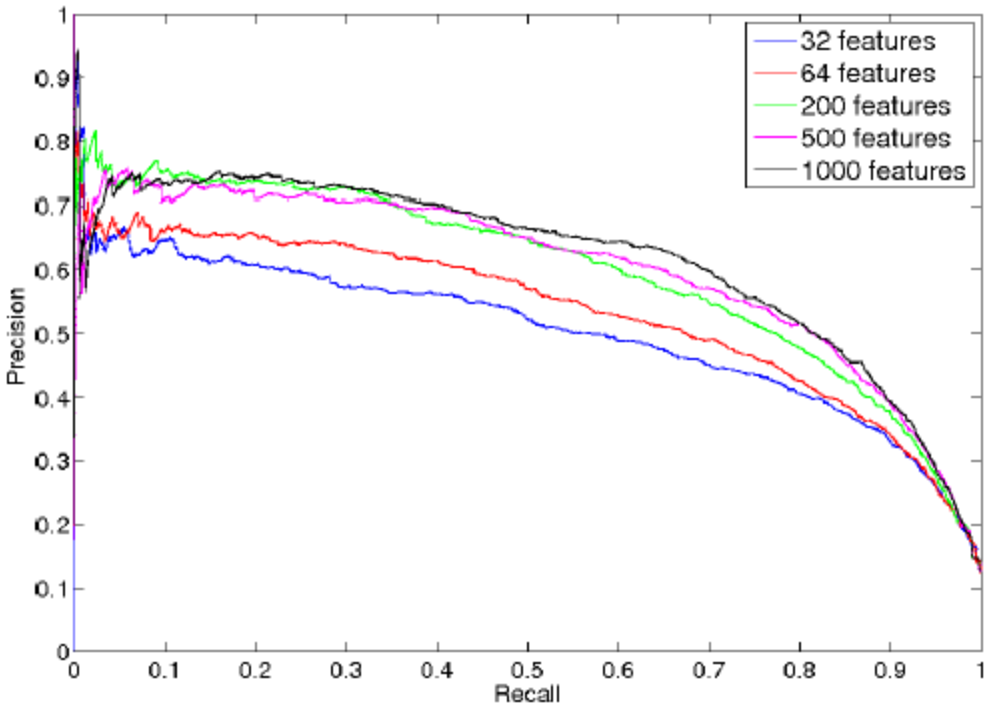
The trained classifier computes the features and classifier output for each 32-by-32 window of the image. This process is done at multiple scales and then, for each location in the original image we assign it a score equal to the maximum classifier output achieved at any scale. Thus, each pixel is labelled with a score according to whether that pixel is part of a block of text. These scores are then thresholded to yield binary decisions at each pixel. By varying the threshold and using the ICDAR bounding boxes as per-pixel labels, we sweep out a precision-recall curve for the detector and report the area under this curve is the final performance measure.

Results

 was detected correctly as "Produkt"

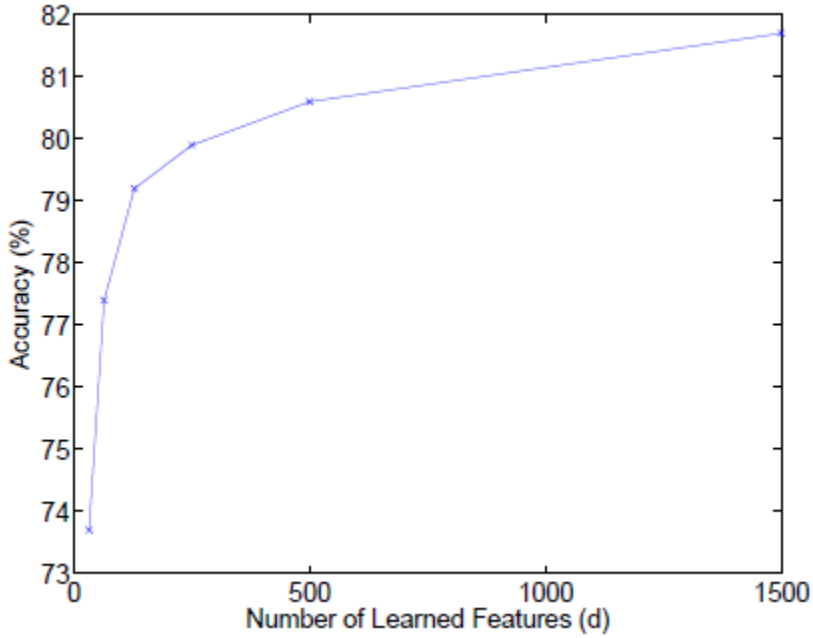
 was detected correctly as "riser"

 was detected correctly as "CHINA"



For characters, the performance improves consistently as we increase the number of features.

Note that these numbers are per-pixel accuracies (i.e. the performance of the detector in identifying, for a single window, whether it is text or non-text).



As with the detectors, we trained our character classifiers with varying numbers of features. We then tested this classifier on the ICDAR 2003 test set, which contains 5198 test characters from 62 classes (10 digits, 26 upper- and 26 lower-case letters). The average classification accuracy on the ICDAR test set for increasing numbers of features is plotted in Figure above. Again, we see that accuracy climbs as a function of the number of features.

Approach 4: Hidden Markov Model for word recognition

Several other approaches use pre-defined template to calculate vertical, horizontal and slant counts in order to determine whether a block belongs to machine printed or handwritten text. Violante et al. make similar assumptions on the uniform block and use similar templates to calculate vertical and horizontal features of a block. A neural network classifier is then used. Because of similar horizontal and vertical features that have been used, both Umeda and Violante's work tend to be sensitive to the variation of machine printed text such as fonts, style and size. The algorithms cannot be readily applied to a document with mixed machine printed text and handwritten annotations.

In natural language, there is an embodied Markovian structure. For example, the probability of seeing a letter "U" after letter "q" is usually greater than seeing "U" after any of the other letters. In a discrete Markov process, each state corresponds to an observable deterministic event. But in a hidden Markov model, the output of each state corresponds to an output probability distribution. In a hidden Markov model, the observation is a probabilistic function of the state. The HMM is a doubly embedded stochastic process with an underlying stochastic process that is hidden, namely the identity of the character in our scenario. Let N be the number of states in the model and M the number of distinct observation symbols per state. A is the state transition probability matrix, where

$$a_{ij} = P[q_{t+1} = S_j | q_t = S_i],\ 1 \leq i, j \leq N$$

B is the observation symbol probability matrix, where

$$b_j(k) = P[v_k\ at\ t\ | q_t = S_j],\ 1 \leq j \leq N,\ 1 \leq k \leq M$$

π is the initial state distribution, i.e

$$\pi_i = P[q_i = S_i],\ 1 \leq i \leq N$$

Methodology

We hypothesize that in machine printed text we will observe a large number of regularities on the projection profile because of regularities in machine printed text. On the other hand, handwritten annotations tend to vary by style, author and environment such that they appear more irregularly.

In classification, we perform the discrimination on a word level. It is not necessary to descend to the character level since a single word (or string) is typically uniform with respect to style.

In our model, we use 62 hidden states corresponding to the 52 upper and lower case Latin characters and 10 numerical digits. We use 10 symbols corresponding to 10 levels of projection

values in observation sequences. Thus, the states are the unknown letters and the observation sequence is a probabilistic function of the states. In this model, element of the state transition probability matrix A is the transition probability between two neighboring letters within a word; element of B is the probability of a symbol occurring in a given letter or digit, and element of π is the probability that the word starts with that given letter or digit.

A. Training

The probability matrices A and π are obtained from samples of ASCII text created from ispell 42869 english words list. The matrix B is computed from a set of bitmap images of various styles of letters and digits.

In training matrix A, we computed the statistics of transition probability between two neighboring letters within a word. The same sample was used to compute the probability of each state (letter or digit) starting a word, to construct matrix π.

B. Text Detection

The first step of the text detection module is to extract Maximally Stable Extremal Regions (MSER) in a given still frame in order to obtain text-region candidates.

At this stage the algorithm produces relatively pure text only groups, with almost all text parts clustered together in separate groups. The detection process finishes with a simple post-processing step aiming to obtain text line level bounding boxes, based on an orientation independent collinearity test.
The Extremal Regions are extracted in parallel for different channels and recognition is done in parallel for different detections (we initialize 10 instances to do recognition in parallel).

The region (contour) in the input image is normalized to a fixed size pf 35x35, while retaining the centroid and aspect ratio. Then, feature vector is extracted based on gradient orientations along the chain-code of its perimeter. After extraction, MSERs are grouped depending on their orientation.

C. Recognition

The detections along with their bounding boxes are sent for decoding to get the probable word along with its confidence. Viterbi algorithm is run to get the maximum probable states. Then K nearest neighbor classifier is used to make final decision based on whether the difference between the maximum probability and second is greater than the average difference.

Results

An interface for live streaming from webcam was done and the code was implemented in OpenCV. Despite a lot of noise, the program was able to recognize text in front of the camera if lighting conditions were proper. The program worked in real-time. Some other examples of recognition done using HMMs:

The text in above image was recognized as “pRioRy galleRies at the sbip”

The text in following images was correctly recognized:

NOTICE

DOUBLE

PARKING

PROHIBITED

AT ALL TIMES

“the copy centre”

HOTEL

Google Tesseract is a widely popular OCR engine. Comparison of this method with Tesseract on existing datasets like ICDAR would be done as the next logical step.

Conclusion and Future Work

The goal of the thesis is to explore Text Recognition and Separation techniques in images from variety of sources, and develop algorithms for separation and recognition. The algorithms for recognition have been analyzed and demonstrated to recognize text of various styles and settings, and the goal is to obtain successful recognition with minimal error.

The results obtained by the first approach (template matching) had lot of inaccuracies and was very sensitive to changes in font. This makes it infeasible for application in generalized setting. However, the second (Neural Networks) and third (CNN) approach were successfully able to recognize to good extent. The output was satisfactory, although not wholly free from error at all times. We also implemented HMMs for word recognition. The results obtained were satisfactory but need to be improved further to obtain better recognition, especially for handwritten text.

The next task is to use HMMs for separation of text and possibly use CNNs or Neural network for recognition task as they seem to be more accurate. Another task is to explore use of Conditional Random Fields for recognition. Deep learning has shown great results in many recognition tasks and I would like to use additional layers in CNN to get better recognition.

However, there are scopes of improvement in the quality of recognition. Firstly, the preprocessing needs to be improved to reduce a lot of noise, especially in the natural images and videos from webcam. Secondly, use of some other feature like projection profiles in case of HMMs can improve the results, this possibility would be explored. Lastly, the training data set needs to be bigger in order to improve the training of the classifier (especially CNN) which no doubt would improve the results.

References

1. Separating Handwritten Material from Machine Printed Text Using Hidden Markov Models; Jinhong K. Guo and Matthew Y. Ma
2. Text Detection and Character Recognition in Scene Images with Unsupervised Feature Learning; Adam Coates, Blake Carpenter, Carl Case, Sanjeev Satheesh, Bipin Suresh, Tao Wang, David J. Wu, Andrew Y. Ng
3. A method for text localization and recognition in real-world images; Lukas Neumann and Jiri Matas
4. HMM+KNN classifier for facial expression recognition, Ch J. Wen and Y.H. Zhan
5. MSER-based Real-Time Text Detection and Tracking; Llu´ıs Gomez and Dimosthenis Karatzas
6. Pattern Recognition and Machine Learning; Christopher M. Bishop
7. “convnet’, A direct Convolution Neural Network implemention in pure C++
8. Text Localization in Real-world Images using Efficiently Pruned Exhaustive Search, Neumann L., Matas J.
9. Globally Trained Handwritten Word Recognizer using Spatial Representation, Convolutional Neural Networks and Hidden Markov Models; Yosua Bengio, Yann Le Cun, Donnie Henderson
10. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition
11. Automatic recognition of a car license plate using color image processing; ER Lee, PK Kim, HJ Kim