TITLE

Thesis submitted in partial fulfillment
of the requirements for the degree of

MS by Research
in
Computer Science and Engineering

by

Ajeet Kumar Singh
201407655
ajeetkumar.s@research.iiit.ac.in

Center for Visual Information Technology
International Institute of Information Technology
Hyderabad - 500 032, INDIA
Dec 2015

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled " " by Ajeet Kumar Singh, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Adviser: Prof. C.V. Jawahar

To My Parents

Acknowledgments

Acknowledgements goes here ...

Abstract

Abstract goes here ...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We live in a world which is increasingly becoming multilingual and at the same time, increasingly automated. The amount of multimedia and captured data are rapidly increasing and stored in digital format. These data also includes multilingual document images, scene images and videos containing text which can play a crucial role in understanding the document images as well as scene/video images. For example, there are many museums which store images of all old and fragile documents which are of importance to historians and other researchers for analysis and archival purposes. While touring in foreign countries, we may not what all the sign boards say.

Several Optical Character Recognition (OCR) tools and systems are used to convert different types of documents, such as scanned document images, PDF files or images captured by a digital camera into editable and retrievable data. Given a document image or scene images, a general OCR will do following tasks:

- **Preprocessor:** It typically includes binarization and noise cleaning followed by skew detection and correction. This step is required by the subsequent steps which works best with binary images. Popular binarization techniques include Otsu method for document images and [4] for texts in scene or video images.

- **Segmentation:** Once the documents are processed, a layout engine is used to extract the structure of the documents. Several document exists which can be used to find the text area, graphics area, table area and other document elements in the document. Words and lines are then segmented from the text area using connected component analysis. The resulting symbols are then recognized by the recognition tool.

- **Feature Extraction:** Local and global features are then extracted from the symbols extracted above. Some commonly used features include HOG, SIFT, profile based features, appearance based features etc.

- **Classification:** Features extracted above is then given as input to off-the-self classifiers to classify the input features to one of the possible output labels. For linearly separable features, linear

Support Vector Machines (SVM) is used else, classifiers like Neural Networks. HMMs, Random Forests, etc.

- **Post-processor:** Output obtained from the classifier is then corrected by dictionary based language models.

It can be seen from the above OCR process, all the existing OCR systems makes an assumption that the language of the text document and scene/video images is known beforehand. Individual OCR tools have been developed to deal best with only one specific language. A Hindi OCR package will deal very well with Hindi, and may be able to cope passably with some other Devanagari alphabet languages such as Marathi or Gujarati. It will not, however, be very helpful if given a Telugu or Tamil document to process. In an automated environment such document processing systems relying on OCR would clearly need human intervention to select the appropriate OCR package, which is obviously not desirable. A *pre*-OCR language or script identification system would enable the correct OCR system to be selected in order to achieve the best character interpretation of the document. This area has widely researched to date with its growing importance to the document image processing community and the progression towards the "paperless office".

Script identification in printed and handwritten document images is a highly researched problem. Contrary to the scanned and handwritten images, script identification in scene images poses several problems.

- **Lack of context**. Scene text often appears as a single word or a group of words, and applying larger sentence or paragraph level context is hard.

- **Complex background**. Scene text come with highly complex natural scene background, on the other hand document images often contain predominantly text.

- **Quality of image.** The quality of the image will directly affect the identification accuracy. As scene texts are often captured under uncontrolled environments, the difficulties in identification may be caused by several factors such as low resolutions, noises and illumination changes.

- **Similarity in scripts.** Some scripts/languages have relatively minor differences *e.g.* Hindi, Gujarati and Bangla. These languages share a subset of characters that have exactly the same shapes. Distinguishing them relies on special characters or character components, and is fine-grained classification problem.

- **Variations.** Text images have arbitrary aspect ratios, since text strings have arbitrarily lengths, ruling out some image classification methods that only operate font-sized inputs. Scene text images often contains stylish fonts to attract the viewers and do not easily generalize to the training data.

Figure 1.1 In order to move towards a "paperless office", there are million of documents in several scripts and languages to be digitized. But due to the limitation of existing OCR systems, the inherent script and language of the documents should be known beforehand. Hence, a script identification module is added in OCR system which will identify the scripts and language at word or line level before passing it to corresponding scripts/language OCR.

1.1 Prior Art

Script and language identification is highly researched area in document image analysis community. There have been several methods and approaches which deals with the script and language identification at page, line or word level. Although there are many research related to script identification at all the levels. But there is a dearth in research on language identification beyond document level. We describe the several approaches in following paragraphs which has been used in the past for the script and language identification at aforementioned levels.

Text Symbol based Approach: Hochberg *et al.* present a method for automatically identifying the script of a binarized document using cluster-based text symbol templates. Text symbols are rescaled to 30×30 pixels. Symbols are grouped into clusters based on Hamming distances. The centroid of the clusters will act as template which will be used to calculate the distance score with test image. The script template with which distance score is low is then allotted to the test image. Note that, a distance score of the test image is calculated with all the script templates. This method is, however, prone to misclassification due to variations in fonts and is not suitable for low resolution images.

Upward Concavities: Spitz proposes a method for distinguishing between Asian and European languages by examining the upward concavities of connected components. The upward concavities is shown to be markedly different between the languages aforementioned. Gross or document level script identification is performed by analyzing the variance of the distribution. For Asian languages such as Chinese, Japanese and Korean, an optical density function is computed whereby the total number of black pixels in each character is tabulated in reading across the text. Study of the distributions of these optical density shows distinct differences amongst the Asian languages. Character shape codes relate to the dimensions of characters rather than the actual characters themselves. Language identification of several Roman alphabet languages is performed by statistical analysis of frequent combinations of

character shape codes in the languages investigated.

Texture Based approach: Global approaches generally refer to the texture of the documents, line or word. These textures are complex visual pattern composed of sub-patterns. These textures can be used to measure the periodicity of the image. These textures produces variations in character density and stroke orientation. Gabor filters act as good model for texture classification [41]. A multichannel gabor filter has been employed which requires an $N \times N$ pixel as input, orientation and frequencies. Then a rotational invariant texture features were computed.

Neural Networks: Lee and Kim use a self-organizing neural network in order to determine not the script of whole document but the individual characters within the document []. After an initial character shape normalization. Zero, first, second order features are calculated using a Mesh feature system, overlapping contour direction codes []. There are then two classification stages, a coarse classifier, followed by a fine classifier which classifies the characters in the mixed groups and presumably performs actual character identification. In recent there have been an efforts to use the discriminative features learned using Convolutional Neural Networks (CNN) for multi-script recognition [36]. These features are automatically extracted and learned at connected component level of the document image.

The approaches mentioned above mainly deal at script level identification. But, when the inherent script of document images are same, visual features are hard to separate between different languages, especially when the identification is required at word level. This problem is described as language identification in document images. There are many attempts in the past in textual domain to separate the languages using statistics(*e.g. n*-gram probabilities of characters. In image domain, language identification is attempted at page level or paragraph level in the past. There have some methods proposed which categorizes the characters based on a number of character shape features such as character ascenders and descenders. For example, [44] group the character images into a small set of categories first. Then, based on the classification results, each word image is converted into a word shape token. Latin-based languages are finally determined according to the frequency of a single word [44], word pair and word trigram. Shijian and Tan [41] combined the script and language identification using a document vectorization framework. They convert document into vertical cut vector based on the number and positions of vertical cut to capture the shape of the word directly.

1.2 Goals of thesis

This thesis addresses the problem of script and language identification in scanned document images as well as in the camera captured wild images. The major goals of thesis are, i) Developing a simple and effective solution for script identification in the wild and ii) Script and language identification in document images at word and line level for the multi-lingual documents using deep learning, specifically

Recurrent Neural Networks (RNN). Moreover, the scene text recognition in Indian languages is recently growing, many of the datasets for this problem are either very small or not very challenging for real scenario, hence as a part of this thesis we also introduce and benchmark Indian Language scene text datasets.

1.3 Major Contributions

This thesis has follwing major contributions to the area:

1. **Script Identification in Wild.** We first proposed an approach for automatically identifying the script of the text localized on the scene images Chapter 3. This approach is inspired by the advancements in mid-level stroke-based features. The scene-text are then represented with these stroke-based features, and then we use an off-the-self classifier to identify the script of the text image.

2. **Script and Language Identification using Recurrent Neural Networks.** We investigate the utility of recurrent neural networks for script and language identification (Chapter 4). In this we argue that one can predict the script and language with minimal evidence (e.g. given only a word or a line) very accurately with the help of pre-trained RNN. We propose a simple and generic solution for the task of script and language identification which do not require any special tuning.

3. **Datasets**. We also a introduce a Indian Language Scene Text (ILST) dataset which is a comprehensive dataset for Indian language scene text containing six scripts commonly used in India, namely Telugu, Tamil, Malayalam, Kannada, Hindi and English. The dataset contains 500 scene images with more than 4000 words. It can be used for following tasks: text localization, recognition, script identification. In this work we use this dataset for two tasks- cropped word script identification and text localization with script identification (i.e., end-to-end pipeline).

1.4 Thesis Outline

- **Chapter-2.** In this chapter, we provide the necessary background for this thesis and briefly summarize the aspects of recurrent neural networks, bag-of-strokes directly relevant to the works as follows.

- **Chapter-3.** This chapter focuses on end-to-end script identification in the wild using mid-level bag-of-strokes features. The proposed method is compared with recently organized CVSI [38] and also a new dataset for Indian Languages is introduced and benchmarked.

- **Chapter-4.** This chapter focuses on usage of deep learning methods such as Recurrent Neural Networks for script and language identification in document image with minimal evidence (only

words and lines). The proposed method have been tested on 55K documents from 15 different scripts and languages.

- **Chapter-5.** This chapter summarizes our work, comparisons and impacts of the work. Here we also discuss the final version of thesis.

Chapter 2

Background

Many of the methods proposed in this thesis proposal are inspired by the success of many computer vision techniques. For example, in Chapter-3, we use bag-of-strokes, inspired by bag-of-words method, to represent the scene text as strokes. In Chapter-4 we take the motivations from deep learning and specifically recurrent neural networks to solve the script and language identification problem in document images at word and line level. In this chapter, Section 2.1 details about the Bag-of-Words (BOW) methods and support vector machines (SVM) used for script identification in the wild, Section 2.2 details about the deep learning and recurrent neural networks used for script and language identification in document images at word and line level.

2.1 Bag-of-Words(BOW and SVM)

2.1.1 Image Descriptors

In computer vision, image descriptors or visual descriptors are descriptions of the visual features of the contents in images, videos, or algorithms or applications that produce such descriptions. They describe elementary characteristics such as the shape, the color, the texture or the motion, among others. Visual descriptors are divided in two main groups:

- Global descriptors which are computed on whole images.
- Local descriptors which are computed locally on a small regions or point of interests on image.

In this subsection, we will describe about common image descriptors like Local Binary Pattern (LBP), Scale Invariant Feature Transform (SIFT).

2.1.1.1 Local Binary Patterns LBP

A Local Binary Pattern (LBP) [33] is a local descriptor that captures the appearance of an image in a small neighborhood around a pixel. An LBP is a string of bits, with one bit for each of the pixels in

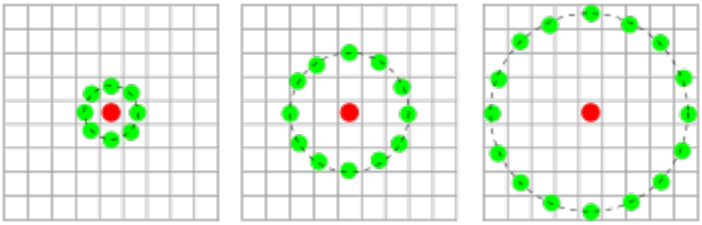


Figure 2.1 The basic LBP operator. The figures shows the circular $(8,1)$, $(16,2)$ and $(8,2)$ neighborhoods. The pixels are bilinearly interpolated whenever the sampling point is not at the center of a pixel. Figure source [33].

the neighborhood. Each bit is turned on or off depending on whether the intensity of the corresponding pixel is greater than the intensity of the central pixel. Figure 2.1 depicts the basic LBP operator.

2.1.1.2 SIFT

Scale-Invariant Feature Transform (SIFT) proposed by Lowe [31] is one of the popular local feature detector and descriptor. SIFT descriptor is invariant to Affine transformation, Lighting changes, Noise. The original SIFT implementation includes both an interest point detector and feature descriptors at the interest points. The descriptor associates to the regions a signature which identifies their appearance compactly and robustly. Figure 2.3 shows an example of SIFT descriptor computation at some keypoints, and how they can be used to match points in different images.

Computing SIFT descriptor at a point starts with sampling the image gradient magnitudes and orientations in a region around the point. The samples are then accumulated into orientation histograms (bin size = 8), summarizing the contents over 4×4 subregions. These orientation histograms capture the local shape. The final descriptor is formed from a vector containing the values of 4×4 array of orientation histogram around the point. This leads to a SIFT feature vector with $4 \times 4 \times 8 = 128$ elements. To obtain illumination invariance, the feature vector is normalized by the square root of the sum of squared components.

2.1.1.3 Gradient Based Features [18]

Inspired by the success of Histogram of Oriented Gradient (HOG) features [11] in many vision tasks, we adapted them to the word recognition problem. To compute the adapted HOG features, we begin by applying the Canny edge operator on the image. Note that we do not expect a clean edge map from this result. We then compute the orientation of gradient at each edge pixel. The gradient orientations are accumulated into histograms over vertical (overlapping) strips extracted from the image. The histograms are weighted by the magnitude of the gradient. An illustration of the feature computation process in shown in Fig. 2.2. At the end of this step, we have a representation of the image in terms of a set of histograms.

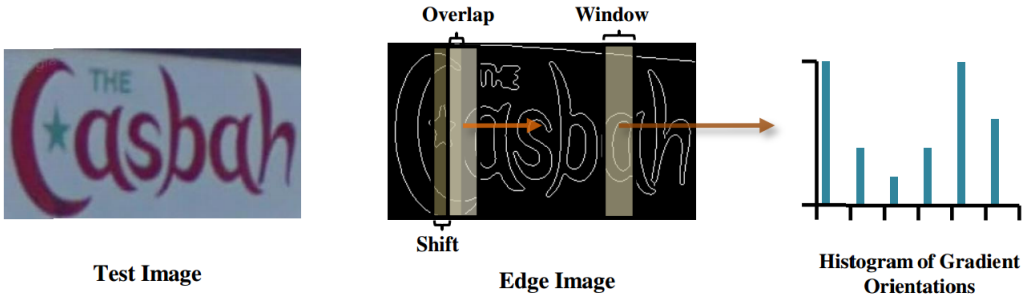


Figure 2.2 Histogram of Gradients computation by recording the gradient orientation at edges. Figure courtesy [18].

2.1.2 *k*-Means Clustering

k-means clustering is an unsupervised clustering algorithm, commonly used for constructing vocabularies for Bag of Words model. Given a set of N data points and number of clusters K, *k*-means partitions the data points into K clusters, such that each data point belongs to the cluster with the nearest mean. Figure 2.3 shows an example with some data points and the clusters formed with 3 clusters.

Let the N data points be $x_1, x_2, \ldots, x_n$, where $x_i \in \mathbb{R}^D$, and K be the number of clusters ($K <= N$). $S = S_1, S_2, \ldots, S_K$ be the set of clusters each consisting of some data points, $/mu_i$ be the mean of points in the set S_i.

$$S = \underset{S}{\operatorname{argmin}} \sum_{i=1}^{K} \sum_{x_j \in S_i} \|x_j - \mu_i\|^2 \tag{2.1}$$

The k-means algorithm uses an iterative refinement technique to solve the optimization problem. The iterative procedure is also referred to as Lloyds algorithm. The algorithm starts with randomly initializing the means $\mu_1, \mu_2, \ldots, \mu_k$. The algorithm proceeds by alternating between the following two EM steps:

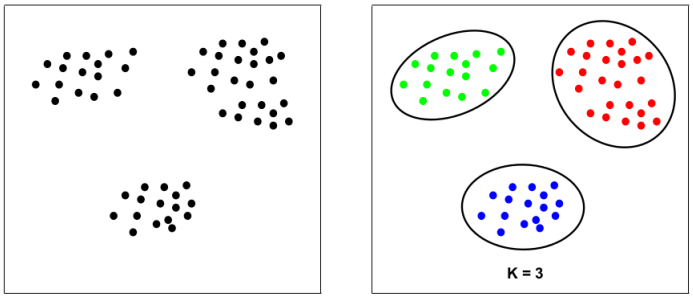
- Expectation Step. During the assignment step, each data point is assigned to the cluster whose mean is closest to that data point.

$$S_i = \{x_p : x_p - \mu_i \leq x_p - \mu_j \| \forall 1 \leq j \leq K\} \tag{2.2}$$

- Maximization Step. During the update step, the mean of each cluster is recomputed after the new assign- ments from the previous step.

$$\mu_i = \frac{\sum_{x_j \in S_i} x_j}{\sum_{x_j \in S_i} 1} \tag{2.3}$$

There is no guarantee that the algorithm will converge to the global optimum, and the result depends on the initialisation of the cluster means. One common practice is to randomly chose K points from



(a) Input data points (b) Clusters

Figure 2.3 *k*-Means Clustering. Example data points, and the clusters computed by *k*-means clustering. Figure courtesy [**?**].

the data points as the initial cluster means, and run it multiple times with different initialisations. There are other variants of initialisations in the literature, for example *k*-means++, which avoids the poor clusterings found by the standard k-means algorithm.

The only parameter involved in k-means clustering is K, the number of clusters. The value usually depends on the nature of data, and should be chosen appropriately by experiments.

2.1.3 Support Vector Machines (SVM)

In machine learning, classification task is a very common task. There are two types of classification tasks in machine learning:

- **Supervised Learning.** It is a task of inferring a function from *supervised* training training data. The training data consists of training examples which are pair of an input vector and a desired output label.
- **Unsupervised Learning.** This type of learning is used to draw inferences from the datasets consisting of input data without label information. For example, cluster analysis which is used to find the hidden patterns in data.

Support Vector Machines is a popular and powerful classification learning tool. It comes under supervised learning tasks. i.e. it learns from a given set of labeled training data and predicts the label for an unseen test datum. We will explain SVMs for two-class case, which is also called as a "binary" classifier. The basic idea behind a linear classifier is to separate the given D-dimensional data points with a $(D1)$ dimensional hyperplane. For a given set of points, there may exist multiple hyperplanes which can separate the data (Figure 2.6(a)). The best classifier of all these hyperplanes is the one which provides the maximum separation of the data points (Figure 2.6(b)). It essentially means that the best hyperplane should maximize the distance between the nearest points on both sides of the hyperplane (nearest points to the hyperplane from each class). This distance is defined as the "margin", and the SVM selects the hyperplane with the maximum margin. The hyperplane obtained is called maximum-margin hyperplane and the linear classifier is called maximum-margin classifier.

Given a set of n labelled training samples,

$$S = \{\{x_i; y_i\} \mid \quad x_i \in D, y_i \in \{1, 1\}\}_{i=1}^{n} \tag{2.4}$$

where, x_i is the D-dimensional data point, y_i represents the class to which the x_i belongs.
A separating hyperplane with w as the normal vector, can be written as

$$w^T x + b = 0 \tag{2.5}$$

Here, b is called the bias term, $\frac{b}{\|w\|}$ gives the perpendicular distance from the origin to the hyperplane. Our goal is to find the w and b, such that the "margin" is maximized. We can select two parallel hyperplanes which separate the data and are as far as possible (Figure 2.6). These hyperplanes can be written as follows:

$$w^T x + b = 1 w^T x + b = -1 \tag{2.6}$$

Now, the distance between the two parallel hyperplanes is $\frac{2}{\|w\|}$. Since the distance needs to be maximised, it translates to minimizing $\|w\|$. Since we do not want any data points falling in between the two parallel hyperplanes, the following constraints are added:

$$w^T x_i + b \geq 1 \qquad \forall x_i \quad s.t y_i = 1 \tag{2.7}$$

$$w^T x_i + b \leq 1 \qquad \forall x_i \quad s.t y_i = -1 \tag{2.8}$$

The two constraints can be combined and rewritten as:

$$y_i(w^T x_i + b) \geq 1 \qquad \forall x_i \tag{2.9}$$

We can substitute $\|w\|$ with $\frac{1}{2}\|w\|^2$, without changing the solution, this makes the optimization problem easier to solve. The optimization problem can now be written in primal form as:

$$\underset{w,b}{\operatorname{argmin}} \frac{1}{2}\|w\|^2 \qquad subject \quad to \qquad y_i(w^T x_i + b) \geq 1 \forall x_i \tag{2.10}$$

2.1.4 Bag of Words Method

The bag of words model in computer vision is inspired from the bag of words model in Natural Language Processing where documents are represented as an unordered collection of words. But in Bag-of-Visual-Words downplays the role of order of words and classifies based on a histogram of the frequency of visual words. The typical bag-of-visual words features consist following steps:

- Extracting local image features (*e.g.* SIFT, HOG).
- Visual Words Vocabulary generation. (*e.g.* by *k*-means clustering)

- Encoding and Spatial Histogram Generation
- Off-the-self classifier learning on the image descriptors (*e.g.* SVM)

2.1.4.1 Extracting Local Image Descriptors

The first step is to compute the local features, which captures the local characteristics of an image. In our work, we use SIFT descriptors. These descriptors are extracted at a dense grid of points. At each point on the dense grid, the SIFT descriptors are computed over multiple circular support patches. The multiple descriptors are computed at each point to allow for scale variation between images.

2.1.4.2 Generating a Codebook

Once the local image descriptors are computed, the next step is to cluster the local descriptors into a visual words' codebook. A codebook is also known as "vocabulary" of visual words, which analogous to the dictionary in text domain. The idea behind a codebook is that an image can be represented in terms of these visual words. *k*-means clustering is a popular method to construct a vocabulary of visual words. A set of random descriptors from a subset of the Training set images is used to construct the visual vocabulary. The number of visual words is a parameter that depends on the dataset, and is generally determined experimentally.

2.1.4.3 Histograms Creation

After learning the vocabulary above, each image is then represented as a bag-of-visual words. Each local descriptor x_i is encoded by the nearest visual word in the Euclidean space. An Euclidean space can be denoted by:

$$c_i = \underset{k}{\operatorname{argmin}} \|x_i - \mu_k\|^2 \tag{2.11}$$

After getting the encodings for all the local descriptors in an image, it is described by a vector (or histogram) that stores the distribution of all the visual words. The size of the histogram is equal to the vocabulary size, where each bin corresponds to a visual word. The final descriptor of an image is a concatenation of the encodings of different spatial regions into a single vector. Note that the histograms of each region are individually normalized before concatenation. The distance between the two vectors reflects the extent to which the images contain similar appearance and the extent to which the appearances correspond in their spatial layout.

2.1.4.4 Model Learning

After getting the image descriptors, the aim is to learn models (classifiers) for the different classes. SVM is a commonly used classifier in BoW steps. For each class, a separate SVM is learned which

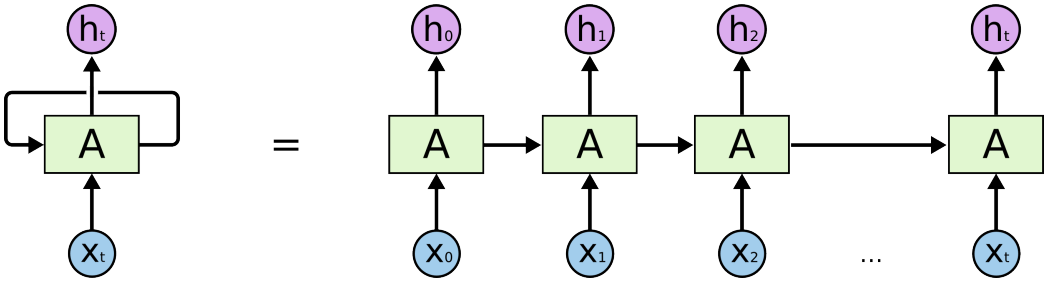


Figure 2.4 A recurrent neural network, unrolled.

can predict whether an unseen image belongs to that particular class or not. For training an SVM for a particular class, the descriptors of the images belonging to that class act as positive data points for the SVM, and descriptors of rest of the images act as negative data points. The set of these positive and negative images is also referred to as the Training Set, while the set of unseen images whose classes are to be predicted, is referred to as the Test Set.

2.2 Deep Learning and Recurrent Neural Networks

2.2.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are inspired from the way a human thinks. Everytime, humans don't start thinking from the scratch. They build upon the knowledge which persists across time. Traditional Neural Networks can't do this, for example a traditional neural network can't predict an event based on its past experiences. RNNs, however address this issue, which have loops in them, which allow the information to persist.

As shown in Figure 2.4, RNNs, when unrolled reveal a chain-like nature and are intimately related to sequences and list. One of the appeals of the RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame. Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in "the clouds are in the sky," we dont need any further context its pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that its needed is small, RNNs can learn to use the past information. But there are also cases where we need more context. Consider trying to predict the last word in the text "I grew up in France I speak fluent French." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. Its entirely possible for the gap between the relevant information and the point where it is needed to become very large.

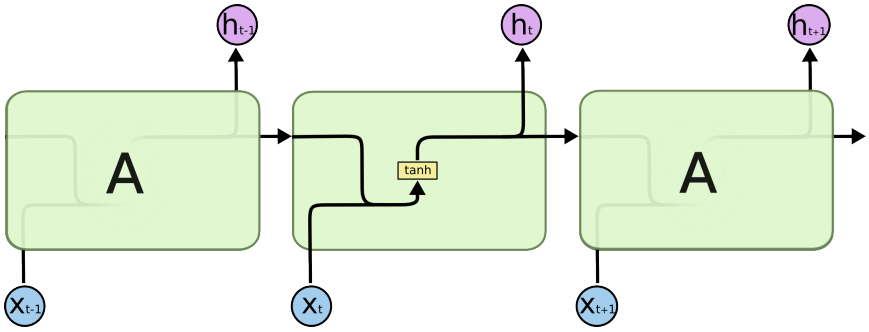


Figure 2.5 The repeating module in a standard RNN contains a single layer

Long Short Term Memory Networks. As we have seen above, RNN are capable of persisting an information over a small timesteps or gap but it can't do that over a longer timesteps. Hence, Hochreiter and Schmidhuber introduced Long Short Term Memory (LSTM) networks. These are special kind of RNNs which are capable of learning the long-term dependencies.

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single *tanh* layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way. Key to the LSTM is the cell state, the horizontal line running through the top of the diagram. The cell state is like a conveyor belt. It runs straight down the entire chain, with minor linear interactions, which helps the information to flow along it unchanged. The LSTM cell have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of sigmoid neutral net layer and a pointwise multiplication operation. These gates allows LSTM memory cells to store and access information over long period of time, thereby avoiding the vanishing gradient problem.

For example, as long as the input gate remains closed (i.e. has an activation close to 0), the activation of the cell will not be overwritten by the new inputs arriving in the network, and can therefore be made available to the net much later in the sequence, by opening the output gate. The preservation over time of gradient information by LSTM is illustrated in Figure 2.6.

LSTM in Action As it can be seen in the Figure 2.7, there are four interacting layers in a cell. We will go through each layer one by one. In Figure 2.7, (i) step, LSTM has to decide upon the information which needs to be thrown out from the cell state. This decision is made by a sigmoid layer called the "forget gate layer". A 1 represents "completely keep this" while 0 represents "completely get rid of this". The (ii) step decides what new information we're going to store in the cell states. This has two parts, a) a sigmoid layer called the "input gate layer" decides which values we'll update and b) a *tanh* layer creates a vector of new candidate values that could be added to the state. In (iii) step, we will combine these two to create an update to the state. In the (iv) and final step we decide what we're going to output. Then, we run a sigmoid layer which decides what parts of the cell state were going to output.

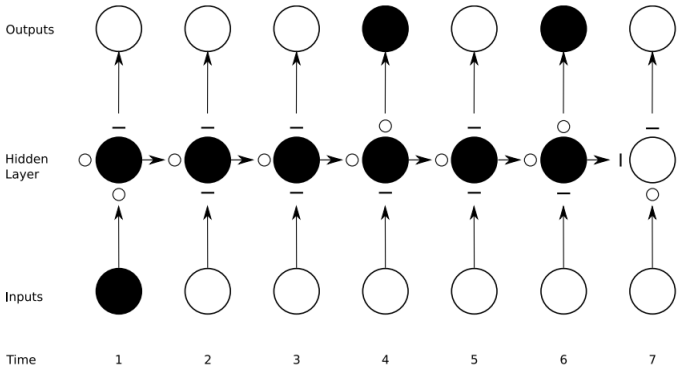


Figure 2.6 Preservation of gradient information by LSTM. The state of the input, forget, and output gate states are displayed below, to the left and above the hidden layer node, which corresponds to a single memory cell. For simplicity, the gates are either entirely open ('*O*') or closed (''). The memory cell 'remembers' the first input as long as the forget gate is open and the input gate is closed, and the sensitivity of the output layer can be switched on and off by the output gate without affecting the cell.

Then, we put the cell state through *tanh* (to push the values to be between 1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

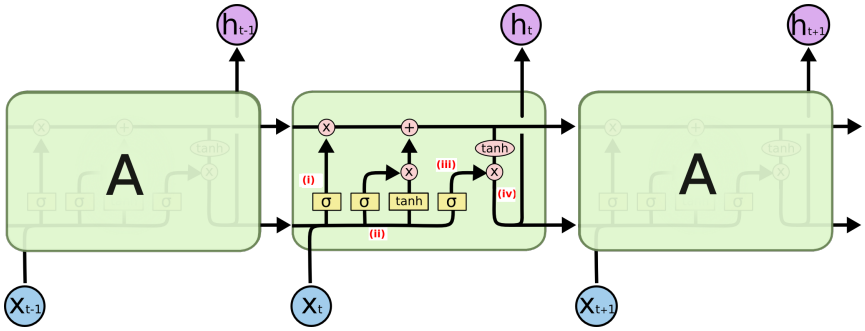


Figure 2.7 The repeating module in a standard RNN contains four interacting layers.

LSTM has been applied to various real-world problems, such as protein secondary structure prediction [21], music generation [13], reinforcement learning [5] and speech recognition and handwriting recognition. As would be expected, its advantages are most pronounced for problems requiring the use of long range contextual information.

Chapter 3

Script Identification in the Wild

We present an approach for automatically identifying the script of the text localized on the scene images. Our approach is inspired by the advancements in mid-level features. We represent the text images using mid-level strokes based features which are pooled from densely computed local features. Once text images are represented using the proposed bag-of-strokes representation, we use an off-the-shelf classifier to identify the script of the text image. Our approach is efficient and requires very less labeled data. We evaluate the performance of our method on a recently introduced CVSI dataset, demonstrating that the proposed approach can correctly identify script of 96.70% of the text images. In addition, we also introduce and benchmark a more challenging Indian language scene text dataset for evaluating the performance of our method.

3.1 Introduction

Reading text in scene images can provide useful information about the content of the image. In multilingual country like India, sign boards often contain text of regional languages along with English and Hindi. The first step of reading text in such images is to answer "what script is this?". The goal of this paper is to answer this question (see Figure 3.1). To this end we use off-the-shelf text localization method and propose a novel mid-level feature based representation which we call bag-of-strokes, for robust script identification. The proposed method achieves an accuracy of 96.70% on a recently introduced Video Script Identification (CVSI) dataset [39]. For comprehensive evaluation of our method we also introduce and benchmark a more challenging Indian Language Scene Text (ILST) dataset in this work. The code and data used in this work will be made available on our project website[1].

Script identification in printed and handwritten document images is a highly researched problem [17, 42, 9]. Contrary to the scanned and handwritten images, script identification in scene images poses many additional challenges such as, (i) lack of context. Scene text often appears as a single word or a group of words, and applying larger sentence or paragraph level context is hard. (ii) stylish fonts. Scene text images often contains stylish fonts to attract the viewers and do not easily generalize to the training data,

[1] http://cvit.iiit.ac.in/projects/SceneTextUnderstanding/

Figure 3.1 A typical example of a street scene image captured in a multilingual country, e.g. India. Our goal in this paper is to localize the text and answer "what script is this?" to facilitate the reading in scene images.

(iii) complex background. Scene text come with highly complex natural scene background, on the other hand document images often contain predominantly text. In case of scene images text localization and dealing with the false detection are few additional challenges. In this work we demonstrate the cropped word script identification as well as end-to-end script identification on scene images. To our knowledge end-to-end script identification on scene images have not been attempted in prior works.

There are many methods in the literature for script identification [39, 17, 35, 25, 34, 40] Texture based features such as Gabor filter [34], LBP [33] have been used for script identification. Joshi *et al.* [25] proposed multi-channel *log*-Gabor filter bank and hierarchical classification scheme for script identification in Indic language documents. Reader is encouraged to refer [17] for detailed survey of classical methods in this area. These classical methods, though achieve high performance on printed documents, are not very successful in our case (see Section 3.4). More recently in ICDAR 2015, competition for script identification on video text has been organized [39]. We compare our method with the entries for this competition and show that our method is comparable to the top performing methods in this competition. There have been few contemporary methods based on deep learning [40, 39] and RNN [42]. These methods achieve noticeable success on some of the selected benchmarks. However, these methods often require huge training data and computation resources.

In this paper, we propose a simple and effective solution for script identification in the wild. Our method is inspired by recent advancements made in mid-level features [14, 27, 6]. First, we densely compute the local features on the given image and then pool these local features to encode the larger context about the given image. In our case, these larger context encode the strokes of the scripts. We represent each training image using bag of these strokes and learn a classifier to identify the script of a test image. The advantages of our method are two fold, first, it is robust to variations and noise commonly present in the scene text. And second, the method is easily trainable and computationally efficient.

(a)

(b)

Figure 3.2 Few example images from thee ILST datset we introduce. (a) we provide ground truth text bounding box, script and text for the images. (b) Few cropped word images of our dataset. The dataset can be used for variety of problems including recognition, text localization etc.

The remainder of the paper is organized as follows. We discuss about datasets in Section 3.2. Here, we introduce Indian language scene text dataset for the problem. In Section 3.3, mid-level features and novel bag-of-strokes based feature representation for text images is introduced. Section 3.4 gives details of the evaluation protocols, and performance measures used in this work. Experimental settings, results, discussions, and comparisons with various techniques are also provided in this section, followed by conclusions.

3.2 Datasets

3.2.1 The ILST dataset

Scene text understanding has gained huge attention in last decade, and several benchmark datasets were introduced [38, 32]. Most of these datasets are for scene text localization and recognition in English. There are also few datasets [40, 39] of multiple scripts e.g., east Asian languages or Indian lan-

guage video text. In this work we introduce Indian Language Scene Text (ILST in short) dataset which is a comprehensive dataset for Indian language scene text containing six scripts commonly used in India, namely Telugu, Tamil, Malayalam, Kannada, Hindi and English. The dataset contains 500 scene images with more than 4000 words. It can be used for following tasks: text localization, recognition, script identification. In this work we use this dataset for two tasks- cropped word script identification and text localization with script identification (i.e., end-to-end pipeline).

Comparison with other related datasets To our knowledge ILST dataset is the largest scene text dataset for the Indian languages. Other related datasets such as CVSI [39], SIW [40] are only meant for script identification on cropped words whereas ours can be used for many other related tasks e.g., recognition and text localization. Also, the dataset is collected in a realistic setting and has wide variations in scale, font style, background and illumination.

Mode of collection. We have collected the images for this dataset by either capturing pictures in streets of various cities in India, harvesting images from Google image search or importing and providing annotation for few images from other existing datasets. These images contain signboards, billboards, posters mainly from urban part of the country. We have collected these images in an unconstrained manner, i.e., without considering much of the view angle. Further, various digital cameras with diverse camera settings were used while capturing images. These are intentionally done to create a realistic dataset.

Annotations. For annotations of the scene images, we use a publicly available web based tool [2]. All the annotations are provided in XML for each image separately describing global image features, bounding boxes of text and its special characteristics. The XML-Schema of LabelMe has been adapted and extended by tags for additional metadata.

Each text field (word) in the image is annotated with following attributes: (i) word bounding box. These bounding boxes are rectangular and parallel to the axes. (ii) ground truth text, (iii) the inherent script of the text, and additional information such as, (iii) illumination, (iv) blur, (v) occlusion, and (vi) 3D Effects.

Train-Test splits. We provide a standard train and test splits for this dataset. We use randomly chosen 30% images of the dataset for the training and the rest for testing. We will make this dataset publicly available on our project website. The details of dataset is provided in Table 3.1. We also show few example images of our dataset in Figure 3.2.

3.2.2 CVSI 2015 [39]

To show the genarality of our method, we also evaluate our method on the dataset which has been introduced in Video Script Identification Competition held at ICDAR 2015. The dataset is composed of images from news videos of various Indian languages. It contains 6412 training text images and 3207

Table 3.1 The ILST dataset: we introduce a ILST dataset which contains 578 scene images and 4036 cropped images from 5 major Indian languages.

Languages	# scene images	# word images	Mode of collection
Hindi	76	514	Authors, Google Images
Malayalam	121	515	Authors, Google Images
Kannada	115	534	Char74K [12]
Tamil	59	563	Authors
Telugu	79	510	Authors
English	128	850	Authors
total	578	4036	-

testing text images from 10 different scripts namely, English, Hindi, Bengali, Oriya, Gujarati, Punjabi, Kannada, Tamil, Telugu and Arabic, commonly used in India.

3.3 Methodology

In this section we introduce our bag-of-strokes based representation of text images for script identification task. First, we briefly give motivation and overview of our method, compare it with closely related works, and then give the details of how we obtain bag-of-strokes representation by pooling local low level features. We finally summarize the full pipeline of our approach.

3.3.1 Motivation and overview

Mid-level feature representation have gained huge attention in last few years. These features are potentially more distinctive than the traditional low-level local features constructed in a purely bottom-up fashion [14]. Mid-level features have achieved noticeable success in image classification and retrieval tasks [27, 6, 14]. Our method is inspired from these methods as we present a novel bag-of-strokes based features which are robust for the task of our interest, i.e., script identification in the wild.

Script identification in the wild is a challenging problem. The traditional low-level local features are not competent for this task. This is primarily due to the imaging, variation in scale, ambiguity, sharing of strokes in the scene text images. On the other hand strokes are the atomic units of scripts and collection of strokes are discriminative enough for the task of identifying the scripts (see figure 3.3). Our method is build on these intuitions. In this work, given a text image we first densely compute local visual features, and then pool these local features into frequently occurring larger patterns (or strokes) and each text image is represented using histogram of these larger patterns (or strokes).

Comparison with other related approaches: The mid-level features have outperformed the naïve bag-of-visual-words based features for image classification [27], because of their robustness and better discriminating power. Usually these mid-level features capture the larger context in the image as com-

Figure 3.3 Strokes are atomic units of scripts. We show some representative strokes of following scripts (top to bottom): Hindi, Kannada, Malayalam, Tamil and Telugu. Our method yields the strokes which are representative and discriminative enough for a cropped image.

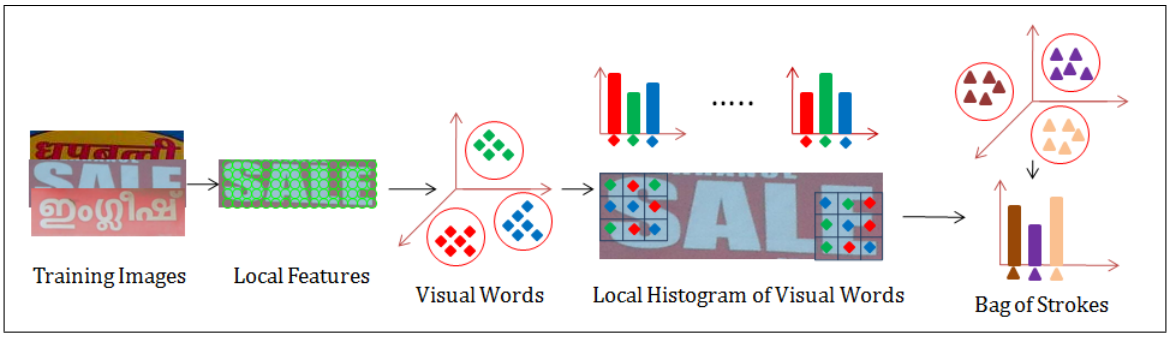


Figure 3.4 Method Overview: The figure depicts the feature computation process where, first we find the local features from the images, we cluster these feature to get the local histogram of visual words. Then we cluster the histogram of visual words to get the representation of words in form of strokes.

pared to the local or semi-local features. One alternative to use larger context is to simply cluster larger patches, as done for local feature computation. However such method are not effective in our case due to the large variability in scene text images. In context of supervision methods, mid-level features can be grouped into three categories: supervised [27], weakly-supervised [14] and unsupervised [43]. Our method falls in weakly supervised category where we only need the class information.

3.3.2 Bag-of-strokes based representation

We compute the bag-of-strokes representation of words from a labeled data. The overview of our method is illustrated in Figure 3.4. Given training text image I_i and its script s_j where $s_j \in \mathcal{S}$ (set of scripts), we follow the following steps.

- First, we densely compute the local features and represent each training image I_i as a set of descriptors (see Section 3.3.2.1 for details of feature computation).
- All the descriptors are then clustered to obtain visual words. Let $C = \{c_1, c_2, ..., c_m\}$ be the set of visual words with vocabulary size = m.
- We obtain assignment for every feature ϕ_k, i.e., obtain the feature-visual word pair (ϕ_k, c_l)
- In a $p \times q$ rectangular neighborhood around feature ϕ_k we obtain a local histogram of visual words H_k. These H_ks capture a larger context and are more discriminative patterns. Indeed they are strokes in our case.
- We again cluster local histograms H_k to obtain larger patterns which encode the strokes. Let $\psi = \{\psi_1, \psi_2, \cdots, \psi_n\}$ be the set of such clusters with stroke vocabulary size = n.
- Once H_k and ψ in hand, we assign every local histogram H_k to one of cluster from ψ. In other words, each image is represented as bag of strokes. We name this representation χ.

At the end of this process each word image in the training data is represented with bag-of-strokes χ. To only use the best strokes we prune few less informative strokes by using the method described in Section 3.3.2.2.

3.3.2.1 Feature computation

Given a text image we compute the SIFT descriptors at points on a regular grid with spacing of M pixels. At each point the descriptors are computed over four circular support patches with different radii, consequently each point is represented by four SIFT descriptors. We also learn the multiple descriptors to allow the scale variation between images. At each grid point the descriptors are computed over circular support patches with radii $r = 4, 6, 8$ and 10.

3.3.2.2 Finding the best strokes for the task

We wish to use the bag-of-strokes χ as a novel set of mid-level features to describe the text image. But not all strokes are are relevant for the task of script identification, e.g., a stroke commonly occurring in all the scripts may not carry useful information (not *discriminative*). Moreover, there are few strokes which can occur in most of the images in a script and are very good *representative* of that script. To measure discriminativity and representativity of a stroke $str \in \chi$ we compute following relevance score:

$$rel(str) = D(str) \times R(str), \tag{3.1}$$

where $D(str)$ and $R(str)$ are discriminativity and representativity scores respectively. To compute $D(str)$ and $R(str)$ we follow entropy based formula. We compute the entropy of stroke by considering (i) scripts as class (script specific entropy) (ii) individual images in scripts as class (image specific

Table 3.2 Results on ILST (cropped words script identification)

Method	***Accuracy*** *(%)*
Baseline Methods	
Gabor features [34]	59.25
Gradient features	47.74
Profile Features [16]	49.24
LBP [33]	78.08
Ours	**88.67**

entropy). We use these entropies to define $D(str)$ and $R(str)$ such that lower value for script specific entropy and higher value for image specific entropy results in higher values of $D(str)$ and $R(str)$. This ensures that those strokes which are found in certain script and almost all the images of that script are more relevant. We prune the bottom 20% less relevant strokes before representing images using bag-of-strokes.

3.3.3 Script identification: Full pipeline

Given a scene image our goal is to localize text and then identify its script. To this end we first obtain text localization using a method proposed in [30] and an open source OCR [1]. While the text localization technique we apply is rather standard, we adapt this for the multi-script dataset we use. Once the text is localized we represent it using bag-of-strokes representation which is learned from the training data (discussed in Section 3.3.2). Each localized text is now fed to a linear SVM classifier which is trained for the task to obtain the inherent script.

3.4 Experiments

Given a scene image containing text our goal is to localize the text and identify it's script. We show results in two settings, (i) end-to-end pipeline, and (ii) cropped word script identification on the datasets presented in Section 3.2. In this section we provide details of implementation, evaluation protocols and baseline methods, and evaluate the performance of our method and compare it with previously published works.

3.4.1 Implementation details and design choice

The proposed method is implemented on a system with 16 GB RAM and Intel® Core™ *i3*-2105 CPU @ 3.10GHz system. The proposed system takes approximately **0.4** ***millisecond*** to identify the script of a

Table 3.3 Results on ILST (End-to-End pipeline). We use [30] and tesseract [1] for text localization and evaluate our proposed method of script identification based on measure presented in Section 3.4.2

Script	*Precision*	*recall*	*f-score*
Telugu	0.47	0.54	0.51
Tamil	0.41	0.44	0.42
Malayalam	0.49	0.45	0.47
Kannada	0.39	0.47	0.42
Hindi	0.42	0.48	0.45
English	0.46	0.56	0.50

cropped word. The two important parameters visual word vocabulary size m and stroke vocabulary size n (refer Section 3.3.2) were empirically chosen as 4K and 3K respectively. The parameter C in SVM is obtained using grid search on independent validation set. We keep these parameters fixed for all our experiments.

3.4.2 Evaluation Protocols

End-to-end script identification. We evaluate our method on end-to-end pipeline of script identification. For this we first localize the text in scene images. We use a standard available text localization scheme for localizing the text. Obviously, this step misses some text regions and produces few false bounding boxes. We fed all the text candidate bounding boxes to script identifier.

It should be noted that end-to-end script identification is far more challenging than script identification in cropped words or document images. Since the final score of this reflects the error accumulated due to text localization and incorrect script identification.

To evaluate the end-to-end script identification we use standard measures, precision ($prec$), recall(rec) and $fscore$ computed for every script. For every script s we compute following terms: (i) number of correctly identified words (TP_s). A detected word is called correctly identified if the intersection by union overlap with the ground truth bounding box is more than 60% and it has the same script identified as the ground truth, (ii) total number of identified words (TI_s), and (iii) total number of ground truth words (TG_s)

Once these in hand we compute *precision*, *recall* and *f-score* for every script and every image. We then report mean of these score over all the images in the dataset.

$$prec_s = \frac{TP_s}{TI_s} \tag{3.2}$$

$$rec_s = \frac{TP_s}{TG_s} \tag{3.3}$$

$$fscore_s = 2\,\frac{prec \times rec}{prec + rec} \tag{3.4}$$

The ideal script identifier should achieve 1 for these measures for all the scripts.

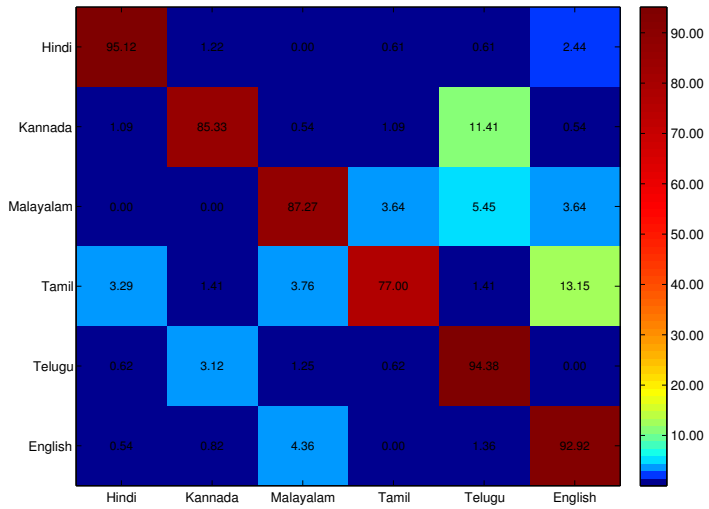


Figure 3.5 Confusion matrix on ILST cropped words. Our method achieve a 88.67% accuracy of script identification on the introduced dataset.

Cropped word script identification. We also evaluate our method on cropped words. For this we compute accuracy which defined as follows:

$$Accuracy = \frac{correctly\ identified\ words}{total\ number\ of\ words} \times 100. \tag{3.5}$$

Here a word is called correctly identified if the method identifies script same as the ground truth.

3.4.3 Baseline Methods

We compare our methods with popular features used for script identifications in document images namely LBP [33], Gabor features [34]. We also evaluate gradient based features and profile features [16] for script identification task and compare with our method. For comparison in CVSI dataset we compare our method with the best performing methods reported in [39].

3.4.4 Results on the ILST dataset

3.4.4.1 End-to-end script identification

We evaluate end-to-end script identification on ILST dataset. To this end we first use public implementation of [30] for text extraction and then fed it to an open source OCR [1] to obtain text boundaries. Once we get bounding boxes we perform script identification using our method and evaluate performance based on performance measures presented in Section 3.4.2. We summarize results of full pipeline in Table 3.3. We observe that our method achieves reasonably high *fscore* for this challenging task. The robustness of mid-level features we use can be attributed as factor for this success. It should be noted that text localization is still an open problem and its performance affects the overall score of

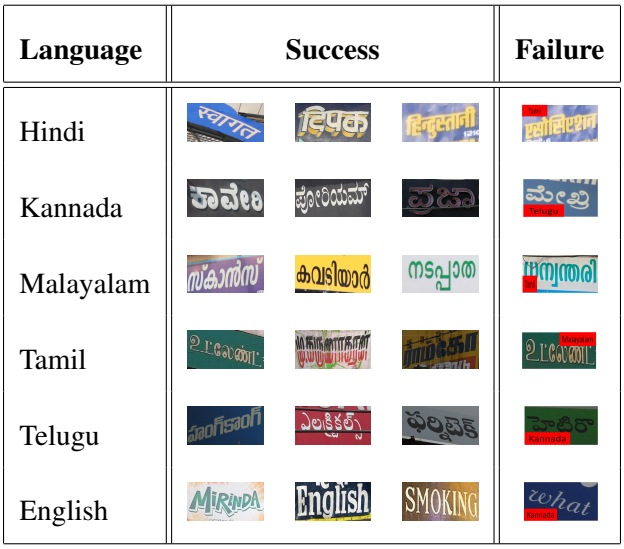


Figure 3.6 Success and Failure Cases. Despite high variations in the dataset, our method correctly identifies the script of scene text images. The "Success" columns depicts the correctly classified word images, and wrongly classified words are shown in "Failure" column along with recognized script in red boxes.

end-to-end script identification.

3.4.4.2 Cropped word Script Identification

We also show results on cropped words on ILST dataset. These results are summarized in Table 3.2. Despite many challenges in this dataset (see figure 3.2) our method achieves script identification accuracy of 88.67% which is significantly better than methods used in document image script identification domain such as [34, 33]. To study script wise confusion we illustrate confusion matrix of our method for ILST dataset in Figure 3.5.

3.4.5 Results on CVSI dataset

Following the protocols of ICDAR competition on video script identification [39] we evaluate our method following for four tasks: (i) Task-1: tri script identification (ii) Task-2: north Indian script identification (iii) Task-3: south Indian script identification, and (iv) Task-4: script identification in all the ten scripts.

We compare our method with top performing methods in this competition. These results are summarized in Table 3.4. Our method achieves 96.70% for Task-4, i.e., script identification in all the ten scripts and clearly outperform two methods in the competition namely, C-DAC and CUK. Moreover, our results are marginally superior to HUST, CVC-1, CVC-2 and comparable to the deep learning based best performing method by Google.

Table 3.4 Task specific evaluation on CVSI [39]. Here A: Arabic, B: Bengali. E: English, H: Hindi,G: Gujrati, K: Kannada, O: Oriya, P: Punjabi, Ta: Tamil, Te: Telugu. Hence AEH means where script identification of three class namely, Arabic, English and Hindi, is performed and so on. Further, Task-1, Task-2, Task-3 and Task-4 indicates tri-script, north Indian script, south Indian script, all script identification, respectively.

Task	Methods						
	C-DAC	CUK	HUST	CVC-1	CVC-2	Google	**Ours**
Task-1							
AEH	95.46	-	99.79	97.32	96.80	100.00	100.00
BEH	91.40	-	98.36	94.68	94.27	99.49	98.61
GEH	88.33	-	99.09	96.38	95.98	99.4	99.41
KEH	91.44	-	99.80	95.51	95.92	99.59	99.19
OEH	95.87	-	99.50	96.88	96.37	99.19	99.49
PEH	84.94	-	98.68	94.20	95.32	99.49	99.37
TaEH	92.71	-	99.39	95.95	96.66	99.70	99.61
TeEH	93.84	-	97.98	96.46	95.96	99.19	97.06
Task-2	96.79	79.50	97.69	95.73	95.91	99.19	97.99
Task-3	86.95	79.14	97.53	95.38	95.75	98.95	96.11
Task-4	84.66	74.06	96.69	95.88	96.00	98.91	96.70

3.4.6 Qualitative evaluation

We qualitatively evaluated our method in Figure 3.6 and Figure 3.7. We show results on end-to-end as well as cropped word script identification. We observe that despite high variations in images such as complex background, illumination change, low resolution our method is successful. Success and failure cases on the cropped image for six script are shown In Figure 3.6. In failure section, Kannada text is wrongly classified as Telugu due to similarity in inherent scripts of both the languages. Similarly, Malayalam text is wrongly classified as Tamil and vice versa. These scripts are visually very similar and often challenges script identifier. It is also very interesting that, an English word is classified as Kannada due to the writing style. Adding location information (i.e., where the image is captured), context (i.e., scripts of neighboring text) can help mitigating such errors. We plan to add such features in our method in future.

3.5 Conclusions

In this paper, we have addressed the problem of script identification in the wild. To this end we made following two important contributions: (i) we introduced a comprehensive dataset for Indian language scene text. This dataset will be useful for the community for many scene text related tasks

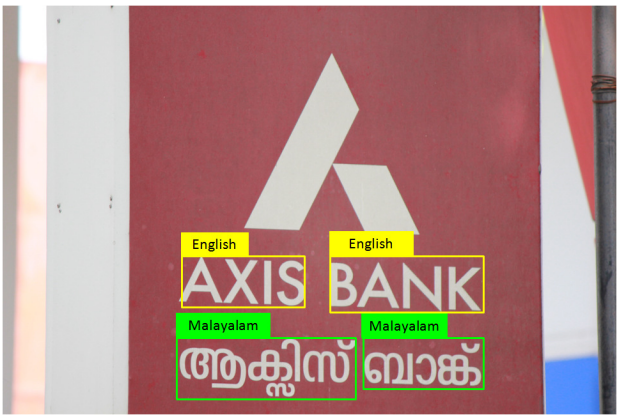


Figure 3.7 An example result of End-to-end script identification of our method. We localize the text boxes in images using method using [30] and [1]. Then we apply our method to find the inherent script in the text boxes.

in multilingual environment in the future. (ii) We have established a baseline for the end-to-end script identification pipeline for scene text and shown that simple mid-level features can achieve reasonably high performance for this task. As a future work we intend to extend our ILST dataset to 10 popular scripts used in India and explore the usage of multiple cues as aid to our script identifier, such as location of the image and neighboring texts.

Chapter 4

Script and Language Identification using Recurrent Neural Networks

In this work, we investigate the utility of Recurrent Neural Networks (RNNs) for script and langauge identification. Both these problems have been attempted in the past with representations computed from the distribution of connected components or characters (e.g. texture, n-gram). Often these features are computed from a larger segment (a paragraph or a page). We argue that one can predict the script or language with minimal evidence (e.g. given only a word or a line) very accurately with the help of a pre-trained RNN. We propose a simple and generic solution for the task of script and language identification which do not require any special tuning. Our method represents the word images as a sequence of feature vectors, and employ the RNNs for the identification. We verify the method on a large corpus of more than 15.03M words from 55K document images comprising 15 scripts and languages. We report an accurate script and language identification at word and line level.

4.1 Introduction

Recurrent Neural Networks (RNNs) have gained popularity in recent years for many recognition tasks such as Optical Character Recognition(OCR) [7, 28], handwriting recognition [20], word retrieval [23], and word spotting [16]. This architecture has started finding more and more applications in diverse areas of computer vision [29]. In this work, we investigate the utility of RNNs for script and language identification at the granularity of words and lines. Naturally, this investigation has its applications in multi-lingual settings, where one needs to decide the script or language before recognition or post-processing of an incoming document image. In addition, this work also throws light on how semantically richer tasks can be attempted without any explicit recognition by looking at the distribution of certain features. For example, can we find the topic model or classify the document into an appropriate category without an explicit textual representation? Such high-level tasks are often attempted based on the statistics and distribution of words or characters. In this work, we limit our attention to the identification of script and language at the word and line level as shown in Fig. 4.1. By designing an RNN that can learn the distribution of feature vectors, we reliably identify the scripts and language from the image itself. We empirically demonstrate the utility of RNNs for script and language identification with experiments on a

Figure 4.1 Figure depicts the script and language identified at word level in document snippets written in Roman-script (first row) based languages and Indic scripts (second row), respectively. In the first row, red, green and blue rectangles denote German, French and Spanish languages, respectively. In the second row, violet, orange and brown rectangles denote Hindi, Telugu and Malayalam scripts, respectively. Unlike the approaches in the past we propose a method to identify the script and language at word and line level by employing popular Recurrent Neural Network (RNNs).

corpus of nearly 15.03M words from 55K document images comprising 15 scripts and languages. We argue that RNNs can be considered as a strong contender for accurate script and language identification which can lead us to the integrated solutions for the recognition tasks in multilingual settings.

Many approaches proposed in the past for script and language identification often deal at page, line or word level. At page level, script is identified by looking at the texture and orientation of the image segments. Sptiz [44] analylzed the individual components for script identification in document images using attributes such as upward concavities, optical densities, character height densities and top and bottom profiles. The use of texture has also been extensively used in script identification. Tan [45] proposed to solve this problem using a multi-channel Gabor filter. Many later attempts used different variations of texture features computed from Gray-level Co-occurrence Matrix, Gabor Energy, Wavelet Energy, Local Binary Pattern [8, 15, 34, 10] for the identification purpose. In recent years, there has been an effort to use discriminative features learned using Convolutional Neural Networks(CNN) for multi-script recognition [36]. These features are automatically extracted and learned at connected component level of the document image.

When the inherent script of document images are same, visual features are hard to separate between different languages, especially when the identification is required at word level. There are many attempts in the textual domain to separate the languages. Often they use the statistics (*e.g.* n-gram probabilities of characters). In the image domain, language identification is attempted at page level or paragraph level in the past. A class of methods have been proposed which categorizes the characters based on a number of character shape features such as character ascenders and descenders. For example, [44] group the character images into a small set of categories first. Then, based on the classification results, each word image is converted into a word shape token. Latin-based languages are finally determined according to the frequency of a single word [44], word pair and word trigram. Shijian and Tan [41] combined the script and language identification using a document vectorization framework. They convert document

image into a vertical cut vector based on the number and positions of vertical cut to capture the shape of the word directly.

Our method is simple, efficient and accurate, without any special tuning for the scripts or languages of interest. We convert the word or line images into a sequence of feature vectors and train the RNNs to reliably separate the script or language. We report comparable, if not better results than the state-of-the-art [34]. Our method also leaves lots of scope for further improvement in performance with better features and special adjustments (e.g. hierarchical classification, special features for harder pairs). We believe this makes our method very generic and applicable in a wide range of settings. We perform the experiments on 12 Indic scripts: Hindi, Malayalam, Gurumukhi, Kannada, Tamil, Telugu, Bangla, Marathi, Gujarati, Assamese, Manipuri and Odiya, and 3 Roman script based languages: French, German and Spanish. We discuss the method in Section 4.2 and the experimental results in Section 4.3.

4.2 RNN for Script and Language Identification

In traditional feed-forward neural networks (FFNN), connections between the nodes do not form any cycles. If we relax this condition, and allow the cyclical connections among the nodes, we obtain the *recurrent neural networks* (RNNs). RNNs in the past have been used to handle sequential data. RNN is a powerful classification tool, as it allows a "memory" regarding previous inputs to persist in network's internal state, which can be later used to influence the network output. RNNs are not widely popular, as they often require a longer training process, because the error path integral decays exponentially along the sequence [22]. Our preference for RNNs is motivated by the fact that it has superior characteristics in several aspects. Unlike HMM which uses the current state of input to generate any observations, RNN uses the long-short term memory (LSTM [20]) structure to store the contextual information of previous states. Also it does not require any explicit labeling of all the vectors in the input feature sequences.

For the script and language identification, we use a RNN based Bidirectional Long Short Term Memory(BLSTM) network. These networks have been used in the past for printed text [28] and handwritten text recognition [20]. This network consists of two LSTM networks in which one network takes the input from beginning to end while other network takes the input from end to beginning. The individual output of both the LSTM networks is used to predict the final output. Hence, these networks have been known for remembering the long range of context over several timesteps. The Connectionist Temporal Classification (CTC) [19] is used at the output layer of RNN network to label the unsegmented data which uses a forward-backward algorithm. The CTC [19] layer directly outputs the probability distribution of desired label. The output layer of RNN network contains one node for each class label plus a special node, (ϵ), which indicates "No Label", i.e. no decision can be made about the incoming word/line at that position. Hence, there are $\mathcal{K} + 1$ nodes in the output layer, where $\mathcal{K}$ is the number of class labels. In our system, a training sample can be viewed as a pair of input sequential features and target script/language label

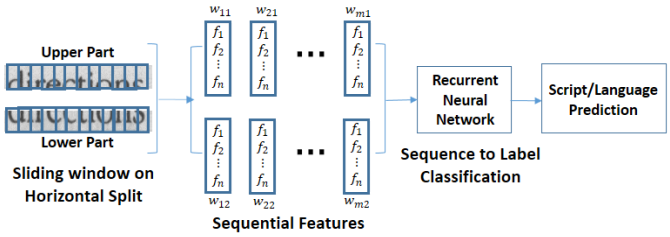


Figure 4.2 The architecture for RNN based script and language identification. From left to right, the segmented line and word from the document images are horizontally divided into two parts. Then, sequence features are calculated from sliding windows, w. Here, m is the number of sliding windows and n is the number of features , f, computed from a single window. These features are then given as input to the LSTM cell of RNN to identify the script and language of current line/word image.

(x, z). The objective function of RNN is then defined by:

$$\mathcal{O} = -\sum_{(x,z)\in\mathcal{S}} \ln p(z|x), \tag{4.1}$$

where $\mathcal{S}$ denotes the training set and $p(z|x)$ denotes the conditional probability of label z given a sequence of feature x. The main objective is to minimize $\mathcal{O}$, which is equivalent to maximization of conditional probability $p(z|x)$. For script and language identification, our method only uses the script/language level annotation.

We also analyzed the network performance on various parameter settings for our identification task. A RNN is characterized by the number of nodes in hidden layer it uses, number of hidden layers and the stopping criteria used for training. We generally stop the RNN training once the training error rate ceased to reduce below a certain threshold. We have observed, experimentally, that increasing the number of hidden layers until 3 gave better results. The best results are obtained with the LSTM size of 50 with 3 hidden layers.

4.2.1 Representation of Words and Lines

In order to use the RNN, the input word and line images are needed to be converted into sequential features. For this, we use the popular profile features [28, 37], which can be used to represent the lines and words as a feature sequence. In this work, we calculate six profile features from every word and image. These features are calculated using the sliding windows of size 20 pixels with an overlap of 75%. For each window, scanning is done from top to bottom and following four features are computed: (F1) vertical profile(i.e. the number of ink pixels in each column), (F2) location of uppermost ink pixel, (F3) location of lowermost ink pixel and (F4) number of ink to background transitions. The profile features are calculated on binarized word/line images obtained using the Otsu thresholding algorithm. We also use the gray level information of the image to extract two features: (F5) mean value and (F6)

standard deviation of gray pixel values. All features are normalized with respect to the image height to [0,1]. These features are made more robust by horizontally dividing the image into two regions and then computing the aforementioned features for each region. Hence, we extract a total of twelve features. The splitting of the image into two parts may seem insignificant, but it helps in differentiating similar symbols which appear in different areas. Fig. 4.2 shows the full pipeline for script and language identification, depicting the various stages of identification framework from feature representation to identification using RNN.

4.2.2 Implementation and Evaluation

The script and language of a line or word image is identified by presenting the corresponding sequential features to the RNN. For this, we train integrated neural networks for both scripts and languages for identification at word and line level. For training the RNN, the initial parameters, number of hidden nodes, number of hidden layers are obtained by cross-validation. Number of input nodes in network is equal to the number of features presented to it (12 in our case) and number of output nodes is same as the number of target labels (in our case 12 nodes for script and 3 nodes for language). For all the experiments, we have used a LSTM size of 50 and number of hidden layers is set at 3. All these experiments were conducted on a mid-level desktop PC having 16GB RAM and a 2.3GHz processor. On an average, training was conducted for 50 epochs for all the experiments mentioned below. The implementation details specific to script and language identification are explained in detail in sections 4.3.1 and 4.3.2, respectively.

4.3 Results and Discussions

In this section, we validate our method on a spectrum of scripts and languages. We present the experimental results of our proposed script and language identification method at both word and line levels.

4.3.1 Script identification

We have tested the proposed method on 12 different scripts of Indian multilingual dataset. For this evaluation we have taken around as many as 5000 pages and as few as 2800 pages from each script, amounting to 50K pages and a total of 12.84M words. This dataset has emerged as a challenging benchmark data (D1) [24] within Indian OCR research community. Almost all of these scripts and languages have their own unique way of representing the character symbols. For example, the scripts of the languages such as Hindi, Bangla and Gurumukhi uses *shirorekha* (headline) over its words while the languages such as Malayalam, Tamil, Telugu and Kannada are more curved in nature. Table 4.1 shows the details of the printed dataset which we have used for our experiments.

Scripts/Languages	D1[24]				Accuracy (in %)			
	Books	Pages	Lines	Words	D1-[24]		D2-[34]	
					Line	Word	Ours	Pati[34]
Hindi	34	5K	133K	1.66M	96.6	85.8	92.3	96.2
Malayalam	31	5K	93K	0.96M	99.2	99.0	96.2	93.3
Gurumukhi	33	5K	125K	1.62M	97.9	93.2	92.8	93.6
Kannada	27	3.8K	90K	0.72M	98.0	93.8	93	93.8
Tamil	23	4.8K	88K	0.64M	98.5	98.1	95.9	95.2
Telugu	28	5K	102K	0.83M	98.4	96.0	91.5	92.3
Bangla	14	2.8K	50K	0.95M	98.6	98.5	94.3	96.2
Marathi	20	5K	127K	1.44M	97.6	95.8	-	-
Gujarati	26	5.2K	124K	1.25M	98.6	98.4	94.5	95.5
Assamese	19	3.5K	73K	0.59M	95.3	93.3	-	-
Manipuri	25	3.6K	69K	0.72M	98.2	71.4	-	-
Odiya	17	5K	109K	1.44M	99.5	97.2	96.4	94

Table 4.1 Table depicts the details of dataset (D1) [24] used for script and language identification. It depicts the performance of our method on the D1 at word and line level. It also shows the comparison of our method against Gabor features with SVM classifier on D2 [34]. Since, D2 [34] didn't show any results on Marathi, Assamese and Manipuri scripts, we are not comparing on these languages.

To train the RNN for word level script identification we use 960K words and 240K words for validation from all the scripts. Training the network for word level script identification took an average of 3.75 hours per epoch. The trained network is then tested on 11.64M words. It took *0.1 ms* to identify the script of a word. At line level too, a separate network (with same architecture) is trained with 120K lines followed by validation with 60K lines from all the scripts . Training time in this task took an average of 4.11 hours per epoch. The trained network is then tested on 1.003M lines. Script identification of a single unseen line took *0.5 ms*. Note that as the average length of input sequence increases, training the network becomes costly with respect to time

We have performed the experiments at line and word level on reported dataset (D1). Table 4.1 shows the accuracy of our script identification method at line and word level for all the scripts. At word level script identification, our method achieves an average accuracy of 93.96% and at line level, we report an average accuracy of 97.90%. At word level, we report a maximum accuracy of 99% for

Malayalam script. And for Manipuri, report a minimum accuracy of 71.4% due to presence of visually similar characters in the script from Assamese script. Similarly, at line level we are getting a maximum accuracy of 99.5% for Odiya and a minimum accuracy of 95.3% for Assamese.

In order to validate the generality of the work, we compare it with the method proposed in [34] on the reported dataset (D2). Their word image dataset (D2) contains about 220K words from eleven different Indian scripts. The method in [34] uses Gabor features with SVM as classifier to identify the scripts of the incoming word images. We train the RNN with 7K words and test it with remaining 13K words from each script. In Table 4.1 we report the accuracy of both these methods on this dataset (D2). Both the method yield comparable results (i.e., 94.59% of our method against 94.8% of [34]). As can be seen, our method which uses naive features yield results that are comparable to those that are evolved over years of research. (Note that wide variety of texture features based on gabor and wavelets are tried in the past [34, 8, 26] and this was one of the top performing descriptors in this class.) In addition, our method uses a simple multiclass classifier and not a hierarchical handcrafted classifier architecture as in [34]. It can also be seen from the Table 4.1, our method on D2 gives an accuracy 94.10% whereas [34] reported an accuracy of 94.44%. One may also note that D2 does not contain scripts (such as Manipuri, Assamese and Marathi) which can get confused with others present in the dataset. On this subset of scripts, we report an average performance of 95.55%, which is better than 94.44% as reported by [34]. Using multilayer perceptrons (MLP) for script identification at word level, yields an average baseline accuracy of 74.67% against our method's 93.96%.

Scripts in Indic languages share some minor or major similarities with each other. For example, Hindi, Bangla and Gurumukhi have *shirorekha* at top of their wordset. Therefore there is a probability that a Hindi word can be confused with a Gurumukhi or a Bangla word, which also holds true for Gurumukhi and Bangla words. Also, there are some characters in these scripts which are visually similar. Fig. 4.4 shows the confusion matrix at word level for all the 12 Indian multilingual scripts. It is evident that aforementioned observation holds true as Hindi is confused with Bangla and Gurumukhi 1.46% and 0.91% of words, respectively. Similarly, Gurumukhi words are identified as Hindi as much as 2% of times and Bangla 1.7%. In confusion matrix table it can also be seen that 2% of Kannada words are being confused with Telugu words and 1.81% of Telugu words are confused as Kannada words. This is due to the fact that Kannada and Telugu alphabets are essentially the regional calligraphic variants of a single script. Assamese, Oriya and Bangla also look similar as they all originated from an ancient Siddhông script. Typographical differences between these scripts are used to identify the alphabets and their script. Hence, it can be seen in the confusion matrix that Assamese words are identified as Bangla 0.51% of the times, and as Oriya 0.54% times. Similarly, Oriya words are identified as Assamese 0.95% times. Some failure cases in script identification at word level has been discussed in Fig. 4.3, where we show the effect of observations made above, on identification at word level.

For line level script identification, we report better results than the identification at word level. We observed through our experiments that longer sequential features, even at word level, gives a good accuracy. Hence, it is natural that the accuracy at line level will be better than the word level accuracy

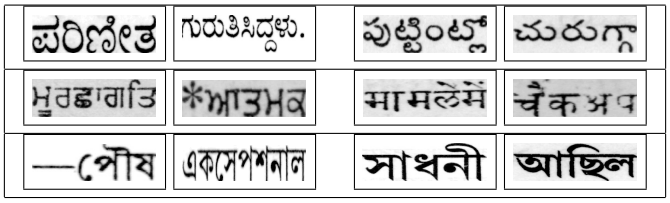


Figure 4.3 Script identification Results: Some failure cases in script identification at word level. First row, first column shows Kannada words identified as Telugu and the second column in same row shows Telugu words identified as Kannada words. In second row, first column shows the Gurumukhi words as Hindi and in second column of the same row, Hindi words identified as Gurumukhi. Similarly in the third row of the figure, first column shows Bangla words identified as Assamese and vice versa in second column.

as RNN becomes more confident with longer sequences. Also, we find that the assumptions which we made above, at word level hold true for the line level too. Although, the accuracies of Hindi, Bangla and Gurumukhi has increased at line level, it is observed that there are still some confusions, albeit low, among these due to their textual properties. For Kannada and Telugu too, there are some confusion due to similarity of the scripts they are written in.

4.3.2 Language Identification

Encouraged by the performance of the method on script identification, we also did experiments to identify the inherent language of a document image at line as well as word level. For this we use three Roman script based languages: French, German and Spanish; two Devanagari script based languages: Hindi and Marathi; two Bangla script based languages: Assamese and Manipuri which also happens to share some vocabulary. Table 4.2 shows the printed dataset details for Roman-based languages. We have used around 2000 pages for each language, amounting to 2.19M words and 154K lines.

For language identification at word level, we train the RNN (with the same architecture as mentioned in section 4.3.1) with 600K words followed by validation with 150K words from all Roman-script based languages. Training took approximately 2 hours per epoch. For testing, around 1M words were used. To identify the language of a word, it took an average of *0.1 ms*. For language identification at line level, we train and validate a different network with same architecture with 30K lines 15K lines, respectively, from all the languages. Training took an average of 0.8 hours per epoch. Trained network is then tested with 100K lines.

For language identification, we are showing the accuracy of all the Roman-script based languages in Table 4.2. We achieve an average accuracy of 93.39% at word level on our dataset. In Table 4.2 we also show the confusion matrix for language identification for the languages at word level. For language

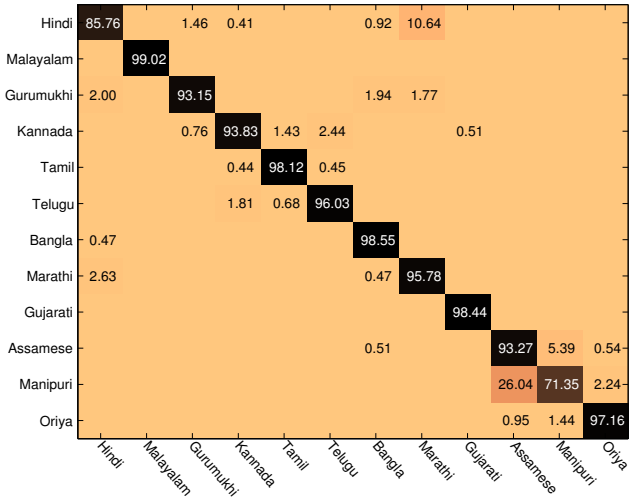


Figure 4.4 Confusion Matrix for the script identification at word level. The blank spaces in the graph denotes predictions that are less than 0.40%.

identification at line level, the average line level accuracy is shown in also shown in Table 4.2. Using RNN, we achieve an average accuracy of 95.25% for language identification at line level.

We also perform language identification at word and line level on some Indian languages that share script and vocabulary. We achieve a fairly good accuracy for all these languages. As it can be seen in Table 4.1, Hindi and Marathi, which share Devanagiri script, obtain an accuracy of 85% and 95.8% respectively. Assamese and Manipuri, which share both script and vocabulary, obtain an accuracy of 93.27% and 71.4% respectively. At the line level too, the Indian languages perform better than that at word level due to longer sequential features. Table 4.1 shows the accuracy at line level for the Indian languages sharing the script as well as some vocabulary.

Lexical similarity is a measure of the degree to which the word sets of two given languages are similar. A lexical similarity of 1 means the complete overlap between vocabularies whereas 0 means no overlap. Lexical similarity of French and German is 0.29, hence, there are 29% common words in French and German language, similarly French and Spanish has 75% of vocabulary overlapping (more information can be found at [3]). Therefore, it is evident in Table 4.2 that 3.63% of Spanish words are confused with French and 3.21% of French words as Spanish. Similarly, 3.47% of French words are confused as German and 5.44% of German words as French. In Indian languages, Hindi and Marathi share a common script of Devanagiri. Hence, it can be seen in the confusion matrix in Fig. 4.4 that 2.64% of Marathi language words are confused with Hindi words. And 10% of Hindi words are confused as Marathi. In Fig. 4.4, it can also be seen that 26% of Manipuri words are confused as Assamese words, and 5.7% of Assamese words are confused as Manipuri words. The failure cases in language identification at word level for both Indian languages and Roman-script based languages are shown in Fig. 4.5.

Language	Dataset				Confusion Matrix (%)			Accuracy (%)	
	Books	Pages	Lines	Words	French	German	Spanish	Line	Word
French	6	1.9K	51K	0.71M	93.32	3.47	3.21	94.51	93.32
German	4	2.1K	55K	0.74M	5.44	92.19	2.37	94.77	92.19
Spanish	5	1.9K	48K	0.63M	3.63	1.70	94.67	96.47	94.67

Table 4.2 Table depicts the Roman script-based dataset used for language identification. It shows the confusion matrix for language identification for Roman-script dataset. It also depicts the performance of our method on the reported dataset at word and line level.

Figure 4.5 Language Identification Results: Some failure cases for language identification at word level for both the Indian and Roman-script based dataset. In the first row, the first column shows the French words identified as Spanish and the second column shows Spanish words identified as French. In the second row, the first column shows the German words identified as French and the second ones shows French words identified as German. For the third row, the first column shows the Marathi words identified as Hindi, and vice versa in second column. In the fourth row, the first column shows the Assamese words identified as Manipuri and vice versa in the second column.

4.4 Conclusion

Script and language identification in multilingual setting is very important in optical character recognition tasks. In this work, we present a simple, efficient and accurate method to predict the inherent script or language at word and line level with minimal evidence, using a pre-trained RNN. This work comprises of two important components. First component computes the sequential feature from an unsegmented word image. The second component, LSTM is used to classify the incoming word image into its corresponding scripts and languages. Also, experiments on a public dataset show that even with naive features, RNNs achieves good if not better results than the state-of-the-art method. We also observe that, RNNs are able to characterize the statistical distribution of features computed over vertical segments. We hope that this can help in other forms of recognition free tasks in document image understanding.

Acknowledgements. This work is supported by Ministry of Communication and Information Technology, Government of India, New Delhi.

Chapter 5

Conclusions

Conclusion goes here ....

Related Publications

Bibliography

[1] Tesseract OCR, http://code.google.com/p/tesseract-ocr/.

[2] LabelMe - The Open Annotation Tool. `http://labelme.csail.mit.edu/`.

[3] Ethnologue - webpage. `https://www.ethnologue.com/`.

[4] Authors. Frobnication tutorial. 2006. Supplied as additional material `tr.pdf`.

[5] B. Bakker. Reinforcement learning with long short-term memory. In *In NIPS*, 2002.

[6] Y. Boureau, F. R. Bach, Y. LeCun, and J. Ponce. Learning Mid-Level Features for Recognition. In *CVPR*, 2010.

[7] T. Breuel, A. Ul-Hasan, M. Al-Azawi, and F. Shafait. High-performance ocr for printed english and fraktur using lstm networks. In *ICDAR*, 2013.

[8] A. Busch, W. Boles, and S. Sridharan. Texture for script identification. *PAMI*, 2005.

[9] S. Chanda, S. Pal, K. Franke, and U. Pal. Two-Stage Approach for Word-wise Script Identification. In *ICDAR*, 2009.

[10] S. Chanda, S. Pal, K. Franke, and U. Pal. Two-stage approach for word-wise script identification. In *ICDAR*, 2009.

[11] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *In CVPR*, pages 886–893, 2005.

[12] T. E. de Campos, B. R. Babu, and M. Varma. Character recognition in natural images. In *VISAPP*, 2009.

[13] D. Eck and J. Schmidhuber. Finding temporal structure in music: blues improvisation with lstm recurrent networks. In *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on*, 2002.

[14] B. Fernando, É. Fromont, and T. Tuytelaars. Mining Mid-level Features for Image Classification. *IJCV*, 2014.

[15] M. Ferrer, A. Morales, and U. Pal. Lbp based line-wise script identification. In *ICDAR*, 2013.

[16] V. Frinken, A. Fischer, R. Manmatha, and H. Bunke. A novel word spotting method based on recurrent neural networks. *PAMI*, 2012.

[17] D. Ghosh, T. Dube, and A. P. Shivaprasad. Script Recognition - A Review. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2010.

[18] V. Goel, A. Mishra, K. Alahari, and C. Jawahar. Whole is greater than sum of parts: Recognizing scene text words. In *ICDAR*, Aug 2013.

[19] A. Graves, S. Fernández, F. J. Gomez, and J. Schmidhuber. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In *ICML*, 2006.

[20] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *PAMI*, 2009.

[21] S. Hochreiter, M. Heusel, and K. Obermayer. Fast model-based protein homology detection without alignment. *Bioinformatics*, 2007.
[22] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computing*, 1997.
[23] R. Jain, V. Frinken, C. V. Jawahar, and R. Manmatha. Blstm neural network based word retrieval for hindi documents. In *ICDAR*, 2011.
[24] C. V. Jawahar and A. Kumar. Content-level Annotation of Large Collection of Printed Document Images. In *ICDAR*, 2007.
[25] G. D. Joshi, S. Garg, and J. Sivaswamy. A Generalised Framework for Script Identification. *IJDAR*, 2007.
[26] G. D. Joshi, S. Garg, and J. Sivaswamy. A generalised framework for script identification. *IJDAR*, 2007.
[27] M. Juneja, A. Vedaldi, C. V. Jawahar, and A. Zisserman. Blocks That Shout: Distinctive Parts for Scene Classification. In *CVPR*, 2013.
[28] P. Krishnan, N. Sankaran, A. K. Singh, and C. V. Jawahar. Towards a robust ocr system for indic scripts. In *DAS*, 2014.
[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*. 2012.
[30] D. K. Lluis Gomez. A Fast Hierarchical Method for Multi-script and Arbitrary Oriented Scene Text Extraction. In *arXiv:1407.7504*, 2014.
[31] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.
[32] A. Mishra, K. Alahari, and C. V. Jawahar. Scene Text Recognition using Higher Order Language Priors. In *BMVC*, 2012.
[33] T. Ojala, M. Pietikainen, and T. Maenpaa. Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns. *TPAMI*, 2002.
[34] P. B. Pati and A. G. Ramakrishnan. Word Level Multi-script Identification. *PR Letters*, 2008.
[35] T. Q. Phan, P. Shivakumara, Z. Ding, S. Lu, and C. L. Tan. Video Script Identification Based on Text Lines. In *ICDAR*, 2011.
[36] S. Rashid, F. Shafait, and T. Breuel. Discriminative learning for script recognition. In *ICIP*, Sept 2010.
[37] T. M. Rath and R. Manmatha. Features for word spotting in historical manuscripts. In *ICDAR*, 2003.
[38] A. Shahab, F. Shafait, and A. Dengel. ICDAR 2011 Robust Reading Competition Challenge 2: Reading Text in Scene Images. In *ICDAR*, 2011.
[39] N. Sharma, R. Mandal, R. Sharma, U. Pal, and M. Blumenstein. ICDAR2015 Competition on Video Script Identification(CVSI 2015). In *ICDAR*, 2015.
[40] B. Shi, C. Yao, C. Zhang, X. Guo, F. Huang, and X. Bai. Automatic Script Identification in the Wild. In *ICDAR*, 2015.
[41] L. Shijian and C. Tan. Script and language identification in noisy and degraded document images. *PAMI*, 2008.
[42] A. K. Singh and C. V. Jawahar. Can RNNs Reliably Separate Script and Language at Word and Line Level? In *ICDAR*, 2015.
[43] S. Singh, A. Gupta, and A. A. Efros. Unsupervised Discovery of Mid-Level Discriminative Patches. In *ECCV*, 2012.
[44] A. Spitz. Determination of the script and language content of document images. *PAMI*, 1997.

[45] T. Tan. Rotation invariant texture features and their use in automatic script identification. *PAMI*, 1998.