Optical Character Recognition as Sequence Mapping

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

MS by Research
in
Computer Science

by

Devendra Kumar Sahu
201250923
devendra.sahu@research.iiit.ac.in

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

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Optical Character Recognition as Sequence Mapping" by Devendra Kumar Sahu, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Adviser: Prof. C. V. Jawahar

To my family

Acknowledgments

I would like to show my deepest gratitude towards my adviser, Dr. C. V. Jawahar, for giving me the opportunity to explore this exciting field, for his guidance and support. He kept me in track and helped me address deep questions at personal and professional level. I would also like to wholeheartedly thank my family, mummy, papa and my brother, for their love and support in all my ventures. I also thank my friends Vivek, Dhiraj, Utsav for their trust in me and always being there for me. I am specially thankful to Pritish, Aniket, Udit and Viresh for their support and enthusiasm which made learning and sharing ideas fun. I am thankful to all my lab mates and friends at IIIT for all the fruitful(as well as not so fruitful) discussions, Anand, Pritish, Viresh, Aniket, Mohak, Udit, Vijay, Swagatika, Saurabh, Nataraj, Suriya, Koustav, Ajeet, Vidyadhar, Tejaswinee, Praveen, Arunava, Sanchit, Yashaswi, Rajvi, Pramod, Sandeep, Nagender, Abhinav, Joshi, Mittal, Vishakh, Bipin. I would also like to thanks Dr. Vishesh for giving me new perspectives on variety of things. A special thanks to all the faculty members at CVIT, my adviser Dr. Jawahar as well as Dr. P. J. Narayanan, Dr. Anoop Namboodiri, Dr. Jayanthi Sivaswamy for creating a wonderful environment in CVIT to do research, and CVIT administration Satya, Phani, Rajan and Nandini for helping me on numerous occasions.

Abstract

Digitization can provide a means of preserving the content of the materials by creating an accessible facsimile of the object in order to put less strain on already fragile originals such as out of print books. The document analysis community formed to address this by digitizing the content thus, making it easily shareable over Internet, making it searchable and, enabling language translation on them. In this thesis, we have tried to see optical character recognition as a mapping problem. We proposed extensions to two method and reduced its limitations. We proposed an application for sequence to sequence learning architecture which removed two limitations of previous state of art method based of connectionist temporal classification output layer. This method also gives representations which can be used for efficient retrieval. We also proposed an extension to profile features which enabled us to use same idea but by learning features from data.

In first work, we propose an application of sequence to sequence learning approach for printed text Optical Character Recognition. In contrast to present day existing state-of-art OCR solution which uses Connectionist Temporal Classification (CTC) output layer our approach makes minimalistic assumptions on the structure and length of the sequence. We use a two step encoder-decoder approach – (a) A recurrent encoder reads a variable length printed text word image and encodes it to a fixed dimensional embedding. (b) This fixed dimensional embedding is subsequently comprehended by decoder structure which converts it into a variable length text output. The learnt deep word image embedding from encoder can be used for printed text based retrieval systems. The expressive *fixed* dimensional embedding for any variable length input expedites the task of retrieval and makes it more *efficient* which is not possible with other recurrent neural network architectures. Thus single model can do predictions and features learnt with supervision can be used for efficient retrieval.

In the second work, we investigate the possibility of learning an appropriate set of features for designing OCR for a specific language. We learn the language specific features from the data with no supervision. In this work, we learn features using a unsupervised feature learning and use it with the RNN based recognition solution. We learn features using a stacked Restricted Boltzman Machines (RBM). These features can be interpreted as deep extension of projection profiles. These features can be used as a plug and play features to get improvements where profile feature are used. We validate these features on five different languages. In addition, these novel features also resulted in better convergence rate of the RNNs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Digital preservation in its most basic form is a series of activities maintaining access to digital materials over time. Digitization in this sense is a means of creating digital surrogates of analog materials such as books, newspapers, microfilm etc. Digitization can provide a means of preserving the content of the materials by creating an accessible facsimile of the object in order to put less strain on already fragile originals. Many of these content are printed on paper and might be out of print for years. Such content even though designed to reach a larger community could stay at few rare spots. The document analysis community formed to address the these challenges by digitizing the content thus, making it easily shareable over Internet, making it searchable and, enabling language translation on them.

The prevalent brittle books issue facing libraries across the world is being addressed with a digital solution for long term book preservation. For centuries, books were printed on wood-pulp paper, which turns acidic as it decays. Deterioration may advance to a point where a book is completely unusable. In theory, if these widely circulated titles are not treated with de-acidification processes, the materials upon those acid pages will be lost forever. As digital technology evolves, it is increasingly preferred as a method of preserving these materials, mainly because it can provide easier access points and significantly reduce the need for physical storage space.

For last two decades there has been a global effort in digitizing and archiving large collection of printed books. Digital Library of India (DLI) is also progressed with same goals. The aim of digitization for easier preservation and making document freely accessible and easily searchable.

1.1 Optical Character Recognition

Optical character recognition (OCR) is the mechanical or electronic conversion of images of typed, handwritten or printed text into machine-encoded text. It is widely used as a form of data entry from printed paper data records, whether passport documents, invoices, bank statements, computerized receipts, business cards, mail, printouts of static-data, or any suitable documentation. It is a common method of digitizing printed texts so that it can be electronically edited, searched, stored more compactly, displayed on-line, and used in machine processes such as machine translation, text-to-speech,

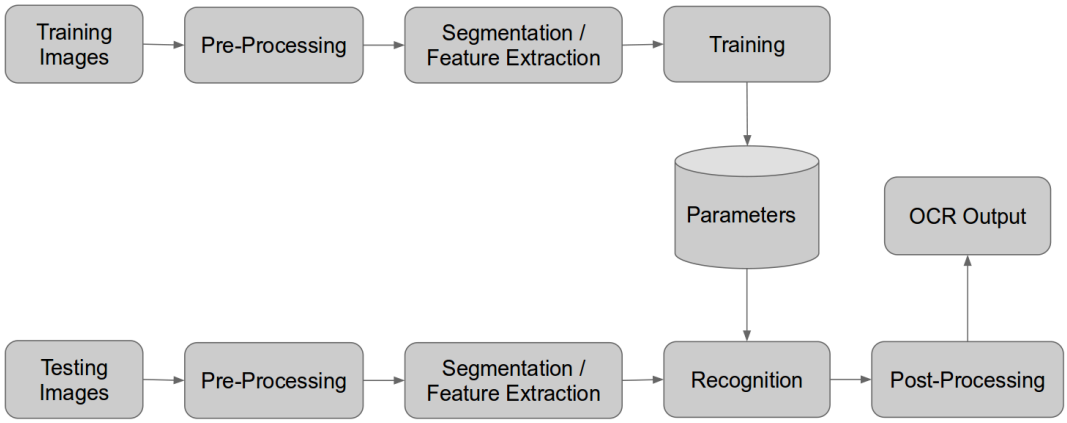


Figure 1.1: Figure shows a pipeline of a typical OCR system. During training phase we perform pre-processing step and then segmentation and feature extraction which is described in 1.2.1. The a chosen prediction model is trained. Using parameters we get from training the prediction model will be used while recognition at test time. The test images go through same preprocessing step and segmentation and feature extraction. Optionally output can go through post-processing module.

key data and text mining. OCR is a field of research in pattern recognition, artificial intelligence and computer vision.

OCR engines have been developed into many kinds of OCR applications. They can be used for:

- Automatic number plate recognition.
- Automatic insurance documents key information extraction.
- Extracting business card information into a contact list.
- Assistive technology for blind and visually impaired users.
- Data entry for business documents, e.g. check, passport, invoice, bank statement and receipt.
- More quickly make textual versions of printed documents, e.g. book scanning for DLI.
- Make electronic images of printed documents searchable e.g. Google Books.
- Defeating CAPTCHA anti-bot systems, though these are specifically designed to prevent OCR.

1.2 Optical Character Recognition Architectures

1.2.1 Architecture of a Typical OCR

The process of Optical Character Recognition typically begins with a document image. They are obtained typically by scanning or by camera capture. Documents captured using such ways will contain

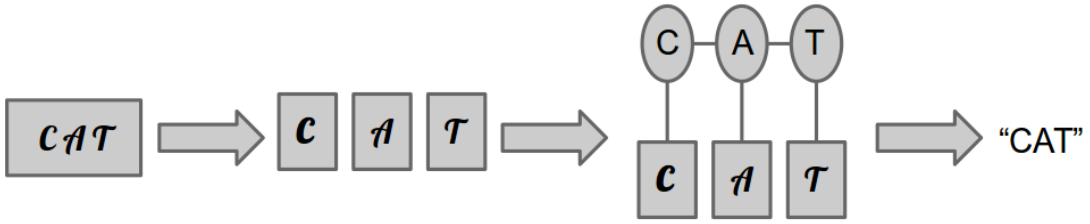


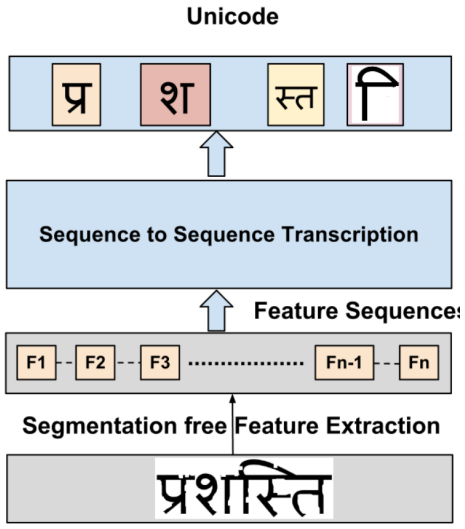
Figure 1.2: Segmentation based OCR which classify output sequence as a whole. In this figure prediction is done given segmented characters it will find best sequence of 3 letters using graphical models. After pre-processing the word image is segmented into characters and we build a graphical model based on the segmented characters and do inference to find the most probably word for given input sequence.

lots of variations and requires a set of pre-processing operations to be performed on them before they can be OCRed (See 1.2.3 for different types of categorized preprocessing). Preprocessing typically includes binarization and noise cleaning followed by skew detection and correction. Binarization refers to the task of converting the original image, which could be in color or grey, to binary format. This process is required as most of the subsequent processes works best with binary images. Popular binarization techniques include Otsu method, Morphology based, Sauvola method etc.. This is followed by correcting the skew of the image as typical scanned input will have a small degree of skewness present in them. There are different methods for correcting the skew and most of them are language specific. For eg: typical skew correction mechanism for Indian language of Devanagari take advantage of the fact that Devanagari words have a head-line (shiro-rekha) connecting all the components.

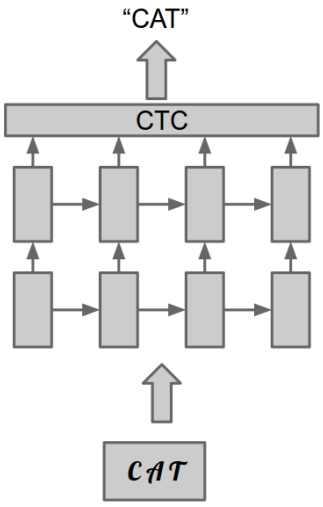
Once the document is processed, a layout engine is used to extract the structure of the document. Typical documents include texts, images, tables etc. in them and it is very important to identify these components. The text content might be split into several columns and layout analysis is required to identify the blocks which needs to be OCRed and also to reconstruct the layout once we generate the OCR output. Several algorithms exist which can be used to extract the lines and words from the text blocks.

Words are then segmented to symbols often by a connected component analysis. Each of these symbols are then recognized/classified. Extracting discriminative features is a key step in any classification problem. Several local and global features can be used in this task. However, using high dimensional features can make the pattern classification process more complicated. Some commonly used features include HoG, SIFT, profile based features, appearance based features, PCA of raw-pixels etc. For any model, feature representation is one of the most important thing to gain performance. A good feature makes the task of classifiers simple and a good classifier makes task of feature extractor simple.

Once we have the features extracted, we use a classifier to recognize them. The purpose of a classifier is to classify the input features into one of the possible output labels. If the input features can be linearly separated, then use of linear Support Vector Machines (SVM) can come in handy. Else, there exists several other classifiers like Neural Networks, HMMs, Random Forests etc. which can perform



(a) OCR schematic [41].



(b) OCR based on recurrent network.

Figure 1.3: Figure 1.3a and 1.3b show segmentation free OCR using recurrent neural network with CTC output layer.

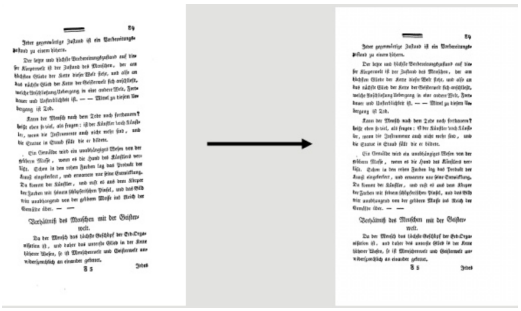
the classification. One main requirement for any machine learning technique is the availability of considerable amount of training data. Lack of training data can often generate a classifier model which may not perform as expected. The classifier model generated after training is stored so that it can be used during testing. There are also several approaches wherein the classifier continues to improve itself in a semi-supervised way. One possible prediction method is shown in figure 1.2.

The labels generated by the classifier needs to be converted to corresponding output format. For documents, it will usually be in UNICODE. However, enough care should be taken when performing this conversion as UNICODE requires you to re-order the output in many cases so that it could be rendered correctly by various text editors. Also, there could be cases where multiple connected components needs to be combined together and assigned a single label. A post-processor module is used to correct any errors which the classifier might have made. They use dictionaries and other language models to perform this task.

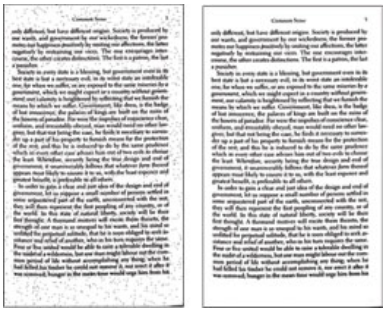
Figure 1.1 shows a typical OCR system. Although what comes under preprocessing is subjective and each step in preprocessing can have its own module. We abstract out preprocessing to focus on recognition system.

1.2.2 Segmentation Free OCRs

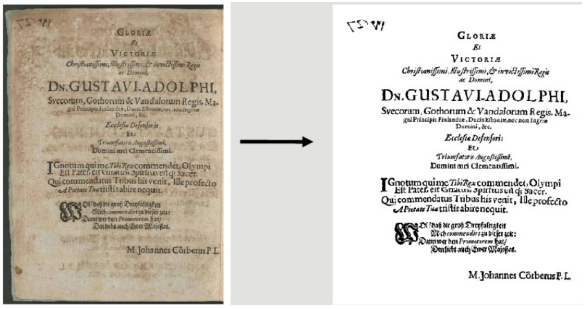
One of performance bottlenecks of traditional OCR architecture is character segmentation. Failure of character segmentation module guarantees the failure of whole system. This decade there has been attempts to perform optical character recognition on whole word without character segmentation mod-



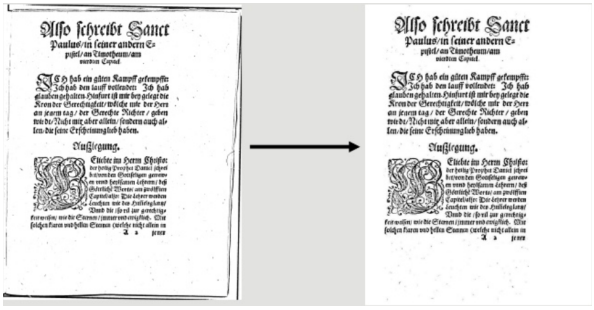
(a) Example of Geometric Correction.



(b) Example of speckle removal.



(c) Example of Binarization.



(d) Example of Border and Line Removal.

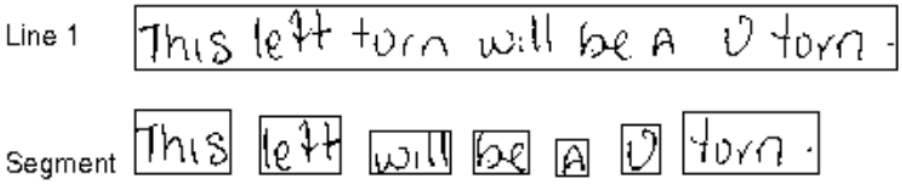
Figure 1.4: Figure 1.4a 1.4b 1.4c 1.4d shows an example of Geometric Correction, Speckle Removal, Binarization and Border / line removal respectively.

ule. Some of them are based on Hidden Markov Models (HMM) and some are based on recurrent neural networks (RNN).

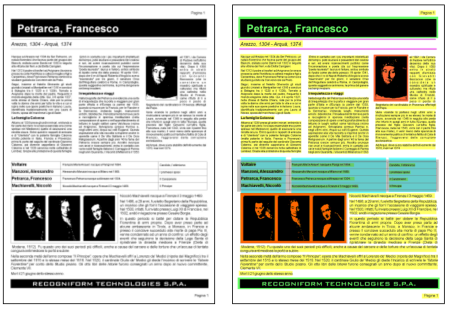
[45] proposed a segmentation free OCRs based on HMMs. A segmentation-free strategy based on Hidden Markov Models (HMMs) is used for offline recognition of unconstrained handwriting. Handwritten textlines are converted to observation sequence by sliding windows and character segmentation stage is avoided prior to recognition. Following that, embedded Baum-Welch algorithm is adopted to train character HMMs. Finally, best character string maximizing the aposteriori is located through Viterbi algorithm. [41, 25] are some work done in same direction using recurrent neural networks. They propose a recognition scheme for the Indian scripts. Their solution uses a Recurrent Neural Network known as Bidirectional Long-Short Term Memory (BLSTM). This approach does not require word to character segmentation. They show high performance on Indic scripts due to availability of labeled data and success of recurrent networks. [34] proposed multilingual OCR using recurrent neural networks.

1.2.3 Preprocessing Steps

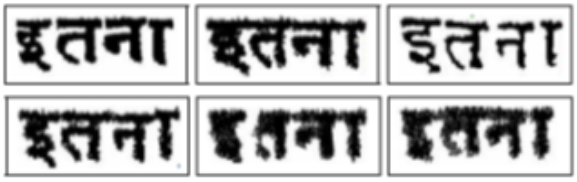
We need to do a lot of preprocessing to get into a state when prediction models are applicable. A prediction model generally required segmented series of characters or whole word or line image. Table 1.1 shows detailed schema of printed text document system. A summary is presented here as pre-processing.



(a) Example of Line and Word Detection.



(b) Example of Layout Analysis. yellow(text), orange(images), green(inverted text), pink(lines), blue(column), gray(table)



(c) Example of different level of noise.

Figure 1.5: Figure 1.5a and 1.5b 1.5c shows an example of Line / Word Detection, Layout Analysis and different level of noise respectively.

- **De-Skewing** - If the document was not aligned properly when scanned, it may need to be tilted a few degrees clockwise or counterclockwise in order to make lines of text perfectly horizontal or vertical. An example is shown in figure 1.4a.
- **Speckle removal** - Remove positive and negative spots, smoothing edges. An example is shown in figure 1.4b.
- **Binarization** - Convert an image from color or greyscale to black-and-white (called a ”binary image” because there are two colours). The task of binarization is performed as a simple way of separating the text (or any other desired image component) from the background. The task of binarization itself is necessary since most commercial recognition algorithms work only on binary images since it proves to be simpler to do so. In addition, the effectiveness of the binarization step influences to a significant extent the quality of the character recognition stage and the careful decisions are made in the choice of the binarization employed for a given input image type; since the quality of the binarization method employed to obtain the binary result depends on the type of the input image (scanned document, scene text image, historical degraded document etc.). An example is shown in figure 1.4c.
- **Line removal** - Cleans up non-glyph boxes and lines. An example is shown in figure 1.4d.
- **Layout analysis** - Identifies columns, paragraphs, captions, etc. as distinct blocks. Especially important in multi-column layouts and tables. An example is shown in figure 1.5b.

- **Line and word detection** - Establishes baseline for word and character shapes, separates words if necessary. An example is shown in figure 1.5a.
- **Script recognition** - In multilingual documents, the script may change at the level of the words and hence, identification of the script is necessary, before the right OCR can be invoked to handle the specific script.
- **Character segmentation** - For per-character OCR, multiple characters that are connected due to image artifacts must be separated; single characters that are broken into multiple pieces due to artifacts must be connected.
- **Normalization** – Normalize aspect ratio and scale.

1.2.4 Recognition and Retrival

There are lot of challenges at preprocessing stage before doing recognition and retrieval (figure 1.4a - 1.4d and figure 1.5a - 1.5c). For recognition, once we reach this stage our task is to convert a word/line image to text. While recognition we also understand the performance bottlenecks at pre-processing stage. For instance, character segmentation is a performance bottleneck and it is possible to do recognition without this module. In this thesis we do full word recognition without character segmentation as mapping a sequence of pixels to a sequence to symbols i.e. text. Output from a recognition system could be fed to language translation system hence reducing language barrier. Retrieval task is defined as given a textual or image query we need to retrieve similar documents. Recognition module may be part of retrieval system. A good retrieval system makes huge corpus of documents searchable.

1.3 Literature Overview

There are various challenge in designing full OCR system due to huge variation in data. The document analysis community dealt with problems specific to digitizing printed text content [13, 30, 35]. Many subproblems were explored by the community. Documents once scanned into images, required to be binarized to separate text from its usually simple background. Methods like Otsu's global thresholding method [37] to local thresholding methods of Sauvola [42] were developed. After binarization, the layout of the text was identified from the foreground and lines and words were segmented out [33, 36]. The words were now recognized by either recognizing character one by one or using a holistic approach [40]. Methods targeting different languages like arabic [31, 4] to various indic scripts [38, 15] were addressed. Offline and online handwritten documents possessed a different set of challenges which were addressed too [40, 39]. Optical Character Recognition (OCR) systems were built which clubbed all parts of document analysis and presented an end to end solution. Some notable systems are Tesseract [43, 3] and ABBYY [1] which can recognize text from scanned document images in real time with

support for multiple languages. There has also been attempts for character segmentation free recognition systems. [45] proposed a segmentation free OCRs based on HMMs. [41, 25] are some work done in same direction using recurrent neural networks. They propose a recognition scheme for the Indian scripts. Their solution uses a Recurrent Neural Network known as Bidirectional Long-Short Term Memory (BLSTM). There has also been attempts to do segmentation free script and language identification for OCRs. [34] proposed multilingual OCR using recurrent neural networks. In this work, an end to end RNN based architecture which can detect the script and recognize the text in a segmentation free manner is proposed for multilingual OCR.

1.4 Schema of Document Analysis

Table 1.1 shows a detailed schema of document analysis. There are different levels of processing in document analysis. Different level of processing moving from lower to higher are as follows: pixels, primitives, structures, documents and corpus.

At pixel level of processing, lot of pre-processing steps happen. Noise reduction is one of the initial steps of pre-processing when noisy documents are converted to its cleaner versions. Then noise reduced grayscale documents are binarized. These document image are binary in nature (foreground and background) hence the name. If the document was not aligned properly when scanned, it may need to be tilted a few degrees clockwise or counterclockwise in order to make lines of text perfectly horizontal or vertical. This is called skew correction. At pixel level processing character segmentation, character scaling, script recognition, language and font recognition can also be done.

A glyph is an elemental symbol within an agreed set of symbols, intended to represent a readable character for the purposes of writing. At next level of processing one could deal with higher level glyph structures like connected components, strokes, symbols and words. These structure can be found, corrected, combined in order to do glyph recognition at different levels like strokes, symbols and words.

Text recognition comes at next level of processing. Word segmentation, line segmentation and content recognition comes under this level of processing. Content recognition uses morphological and lexical context. Morphological context deals with structures of symbols and lexical context deals with words or vocabulary of a language, especially as distinguished from its grammatical and syntactical aspects.

Page layout analysis comes at document level of processing. Page layout analysis identifies columns, paragraphs, captions, text non-text regions etc. as distinct blocks. In document with complex layout like mix of multi-column layouts, images and tables.

Next level of processing is corpus level processing. Retrieval comes at this level of processing. Retrieval task is defined as given a textual or image query we need to retrieve similar documents. Indexing and search are major components of retrieval system. Security, authentication and privacy are higher level issues which also needs to be taken care of in real implementation of information retrieval systems.

1.5 Contribution and Thesis Outline

In the detailed schema of document analysis (in section 1.4), we deal with two thing in the schema. We focus on word prediction and image retrieval problems. The main contributions of this thesis are:

1. **Problem**: *Tackle the limitations of previous state of art* OCR *solutions which used* CTC *(Connectionist Temporal Classification) output layer [16] with recurrent neural networks.*

   There were two major limitations of previous Recurrent Neural Network (RNN) architecture for OCRs. Firstly, it made assumptions on structure of input and output sequence (input sequence length more than output sequence length). Secondly, Once network is trained, its hidden representations cannot be used efficiently for other tasks such as retrieval. We propose a novel application of sequence to sequence learning architecture applied to Optical Character Recognition. We do recognition and supervised feature learning with a single model while tacking limitations of previously used architectures. These learnt features are fixed dimensional representations which can be used to do efficient retrieval using approximate nearest neighbor.

2. **Problem**: *Tackle the limitations of previously used successful profile features which were hand-engineered by learning features from data.*

   Major limitations of designing hand-engineered features is the amount of time involved throughout the design and experimentation cycle. Still this required lot of domain knowledge and has limitation on number of features based no domain experience. Still there is a limitation to how many features our limited domain understand could help us propose. Learning features from data helps us speedup the process of feature design and one could ask many features as required.

In this thesis, we see both problems as a sequence mapping problem. For the first problem, We propose an end-to-end recurrent encoder-decoder based sequence learning approach for printed text Optical Character Recognition (OCR). In contrast to present day existing state-of-art OCR solution which used CTC output layer [16] our approach makes minimalistic assumptions on the structure and length of the sequence(i.e. length of input sequence is more than length of output sequence). We use a two step encoder-decoder approach – (a) A recurrent encoder reads a variable length printed text word image and encodes it to a fixed dimensional embedding. (b) This fixed dimensional embedding is subsequently comprehended by decoder structure which converts it into a variable length text output. Our architecture gives competitive performance relative to Connectionist Temporal Classification (CTC) [16]) output layer while being executed in more natural settings. The learnt deep word image embedding from encoder can be used for printed text based retrieval systems. The expressive *fixed* dimensional embedding for any variable length input expedites the task of retrieval and makes it more *efficient* which is not possible with other recurrent neural network architectures. The utility of the proposed architecture for printed text is demonstrated by quantitative and qualitative evaluation of two tasks – word prediction and retrieval. This is described in chapter 3. For second problem, we extend projection profiles using deep belief networks / stacked Restricted Boltzman Machines. Here, we investigate the possibility of

learning an appropriate set of features for designing OCR for a specific language. We learn the language specific features from the data with no supervision. This enables the seamless adaptation of the architecture across languages. In this work, we learn features using a stacked Restricted Boltzman Machines (RBM) and use it with the RNN based recognition solution. This is described in chapter 4. In chapter 2, we cover technical background useful for further chapters. In chapter 5 we conclude with possible future directions.

Table 1.1: Detailed Schema of Document Analysis [35].

Level of Processing (Low to High)	Task
Pixels	**Preprocessing**
	Representation Noise Reduction Binarization Skew Detection Zoning Character Segmentation Script, Language and Font Recognition Character Scaling
Primitives	**Glyph Recognition**
	Connected Components Strokes Symbols Words
Structures	**Text Recognition**
	Word Segmentation Text Line Reconstruction Table Analysis Morphological Context Lexical Context Syntax and Semantics
Documents	**Page Layout Analysis**
	Text vs Non-Text Physical Component Analysis Logical Component Analysis Functional Components Compression
Corpus	**Information Retrieval**
	Document Classification and Indexing Search Security, Authentication and Privacy

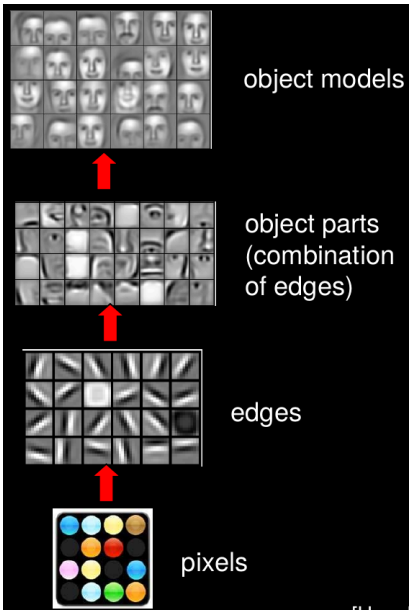
Chapter 2

Background

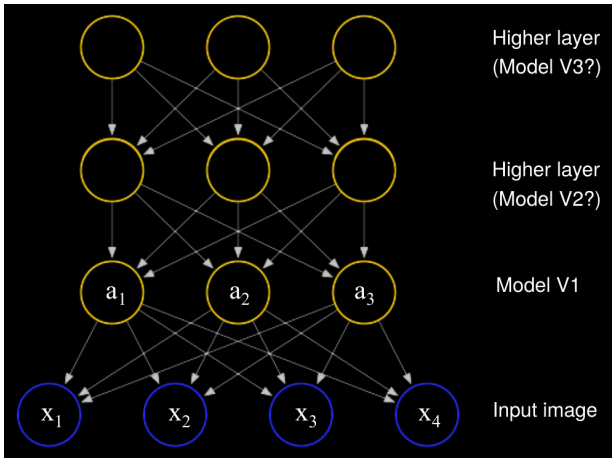
Machine learning is a subfield of computer science that evolved from the study of pattern recognition and computational learning theory in artificial intelligence. It explores the study and construction of algorithms that can learn from and make predictions on data. Such algorithms operate by building a model from example inputs in order to make data-driven predictions or decisions, rather than following strictly static program instructions. It is employed in a range of computing tasks where designing and programming explicit algorithms is infeasible. Machine Learning Technology powers many aspects of modern society, web searches to content filtering on social platforms to recommendations on e-commerce sites. This technology is even present in consumer products like cameras, digital bands and smart phones even without us knowing them. Example applications of machine learning technology include spam filtering, optical character recognition (OCR), search engines, object recognition, transcribing speech to text, drug discovery, genomics etc. Machine learning tasks are typically classified into three broad categories, depending on the nature of the learning "signal" or "feedback" available to a learning system. These are:

- **Supervised learning**: The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs.
- **Unsupervised learning**: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end.
- **Reinforcement learning**: A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle), without a teacher explicitly telling it whether it has come close to its goal or not. Another example is learning to play a game by playing against an opponent.

This thesis deals with two categories(Supervised and Unsupervised learning). Increasingly machine learning technology is using a class of techniques called Deep Learning [5]. All models used in this thesis comes under the umbrella of Deep Learning. In the following sections, we describe Deep Learning, Recurrent Neural Networks (RNN), Restricted Boltzman Machines (RBM).



(a) Example of Heirarchical representation [22]



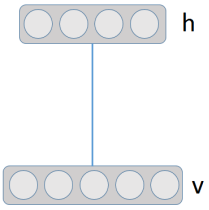
(b) Motivation of heirarcichal representation.

Figure 2.1: Figure 2.1a shows an example of representation in layered architechture. Figure 2.1b shows an example of layered achitecture in neural network.

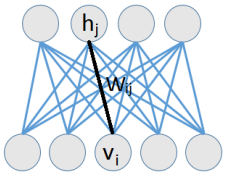
2.1 Deep Learning

Increasingly machine learning technology is using a class of techniques called *Deep Learning*. Conventional machine learning was limited to their ability to process data in their raw form or heavily relied on features which needed careful hand-engineering which took years to build. For a long time, most machine learning technology was based on carefully hand-engineered features which needed considerable domain expertise in order to design feature extractor which transformed the raw data to a suitable feature vector which a learning system can work on. The choice of learning system depends on the task at hand like object recognition, sequence classification or regression tasks on complicated objects like trees or graph.

Representation Learning is a class of methods which enables us to feed raw data into the system and automatically discover the representations needed for a particular task at hand (e.g. classification). *Deep Learning* methods are representation learning methods with multiple levels of representations, obtained by composing simple but non-linear operations that transform the representation at one level into the representation at a higher and slightly more abstract level as shown in figure 2.1a. With composition of enough transformations a complication function can be learned. For classification tasks, higher level of representation amplify the aspects of input which are relevant for discrimination and suppresses irrelevant aspects. An example of different levels of representation is, level 0 being raw input image $\rightarrow$ level 1 presence or absence of edges (assemble pixels into edge group) $\rightarrow$ level 2 presence or absence of an arrangement of edges (assemble edges to form corners or small parts of object) $\rightarrow$ level 3 presence



(a) Restricted Boltzman Machine



(b) RBM showing all the connections

Figure 2.2: Models for representation learning

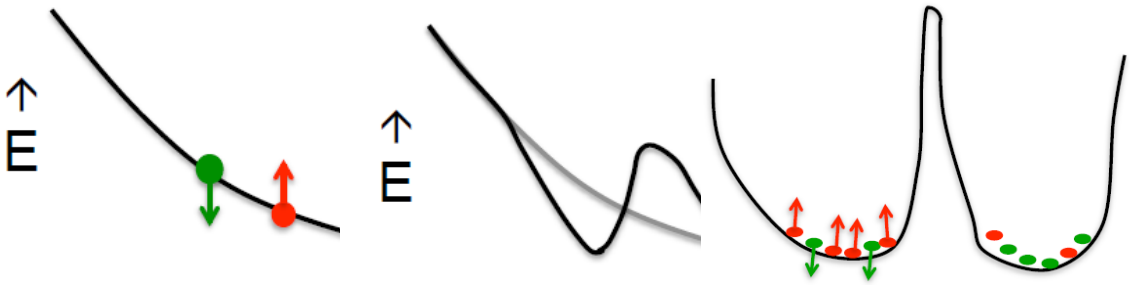
or absence of arrangement of group at level 2 which could be combination that correspond to parts of familiar objects as shown in figure 2.1a. Although deep leaning is very *weakly* connected to brain but still has some parallels with brain at an abstract level. Figure 2.1b shows a layered network with analogy to human brain. Our visual cortex is a layered structure with possibly different layers like V_1, V_2 and V_3. This is obviously an oversimplification but the main idea is being layered structure and possibly this abstract idea could be taken and one can build on this idea with could resemble with a neural network.

Despite significant success of supervised learning today is still severely limited. Specifically, most applications of it still require that we manually specify the input features given to the algorithm. Once a good feature representation is given, a supervised learning algorithm can do well. But in domains as computer vision, audio processing, and natural language processing, there are now hundreds or perhaps thousands of researchers who have spent years of their lives slowly and laboriously hand-engineering vision, audio or text features. While much of this feature-engineering work is extremely clever, one has to wonder if we can do better. Certainly this labor-intensive hand-engineering approach does not scale well to new problems; further, ideally we would like to have algorithms that can automatically learn even better feature representations than the hand-engineered ones. Next section describes an unsupervised feature learning model called Restricted Boltzman Machines.

2.2 Restricted Boltzman Machines

Here we briefly review RBM following [5] [19] [48]. RBM's are energy based models and generally models joint probability distribution of the observed variables and hidden variables, the posterior over latent variable factorizes and thus making inference easy unlike in belief nets. In our presentation we use binary logistic units for both visible and hidden variables.

An RBM defines a distribution over $(v, h) \in \{0, 1\}^{N_v \times N_h}$ v being visible variables and h being hidden variables. RBM models a distribution over visible variables by introducing a set of stochastic hidden features. RBM jointly model the visible and hidden variables. The graphical model has a layer of visible units v and a layer of hidden units h; there are undirected connections between layers but no connections within a layer as shown in figure 2.2b. Figure 2.2a is an equivalent representation of figure 2.2b. So RBM is a two layered network, complete bipartite graph that have a layer of input/visible



(a) Energy and particles before update.

(b) Energy after update.

(c) General snapshot during learning.

Figure 2.3: Energy during contrastive divergence learning.

variables and a layer of hidden random variables. The energy function associated with RBM is described in equation 2.1. Equation 2.2 is probability distribution associated with RBM.

$$E(v,h) = -v^T W h - b^T v - c^T h \tag{2.1}$$

$$P(v,h) = \frac{e^{-E(v,h)}}{\sum_{v,h} e^{-E(v,h)}} \tag{2.2}$$

Here $W \in \mathcal{R}^{N_v \times N_h}$ N_v, N_h are number of visible and number of hidden nodes respectively. $W_{i,j}$ is symmetric weights between v_i and h_j. b and c are biases of hidden units and visible units respectively.

The main advantage of using this undirected, energy-based model rather than a directed belief nets is that inference is very easy because the hidden units become conditionally independent when the states of the visible units are observed. The condition distribution of $P(H|V)$ and $P(V|H)$ is shown in Equation 2.3 and 2.4 respectively.

$$P(h_j = 1|x) = sigmoid(W^T v + c)_j \tag{2.3}$$

$$P(v_i = 1|h) = sigmoid(Wh + b)_i \tag{2.4}$$

Maximum likelihood learning is slow in an RBM but learning still works well if we approximately follow the gradient of another function called the contrastive divergence[19, 9].

2.2.1 Contrastive Divergence Learning

The derivative of the average log likelihood L with respect to the parameters is given by the following equations:

$$\Delta W_{ij} \propto < v_i h_j >_{P(H|V)\tilde{P}(V)} - < v_i h_j >_{P(V,H)} \tag{2.5}$$

$$\Delta c_j \propto < h_j >_{P(H|V)\tilde{P}(V)} - < h_j >_{P(H)} \tag{2.6}$$

$$\Delta b_i \propto < v_i >_{\tilde{P}(V)} - < v_i >_{P(V)} \tag{2.7}$$

where $< \ . \ >_{Q^*}$ is the expectation under Q^*, $\tilde{P}(V)$ denotes the empirical data distribution and $P(H|V)\tilde{P}(V)$ is the RBM distribution that arises when V is set by the data distribution. Maximum Likelihood Estimation is difficult due to the need to compute expectations with respect to the models distribution $<>_P \ (v, h)$. A natural answer to compute these expectations to do monte carlo approximation by using alternating gibbs sampling. We can start from an arbitrary distribution, we alternate between updating all the hidden units using equation 2.3 and updating all the visible units using equation 2.4. After a sufficient number of iterations, this method gives unbiased samples from the distribution P(V,H). It is generally much better than brute-force calculation of the expectation which takes exponential time in the size of the RBM, but it is still slow in practice, since the Markov chain needs to be run for each iteration of the learning algorithm.

The drawback of previous obvious method can be handled by another parameter estimation framework named contrastive divergence(CD). It follows the approximate gradient of an objective function that is the difference of two Kullback-Liebler divergence. CD is much more efficient than maximum likelihood learning and it works well in practice RBMs learned with CD produce high-quality generative models. The weight updates for CD are given by the following equations:

$$\Delta W_{ij} \propto < v_i h_j >_{P(H|V)\tilde{P}(V)} - < v_i h_j >_{P_k(V,H)} \tag{2.8}$$

$$\Delta c_j \propto < h_j >_{P(H|V)\tilde{P}(V)} - < h_j >_{P_k(H)} \tag{2.9}$$

$$\Delta b_i \propto < v_i >_{\tilde{P}(V)} - < v_i >_{P_k(V)} \tag{2.10}$$

where a sample from $P_k()$ is obtained by running gibbs sampling for k-steps, initializing (V,H) by sample from $P(H|V)\tilde{P}(V)$. To be clear, we sample V by $\tilde{P}(V)$. The hidden variables H from $P(H|V)$. Then sample the visibles and then the hiddens once more to get a sample from P_1. We can continue sampling for more k-1 steps to get a sample from $P_k()$. k=1 works well in practice. The positive phase terms lowers the free energy of the data and the negative phase term increases the free energy of the samples which the model can generate so as to prevent the directions of distorted samples in future. If we look at figure 2.3a, green particles correspond to positive phase terms and red particles correspond to negative phase term. So to summarize, data dependent green particle lowers the energy where there exists data and red particles increase the energy in regions where model thinks data exists. Energy after update changes from figure 2.3a to 2.3b. Figure 2.3c shows are general setting where negative particles keep increasing energy till they escape.

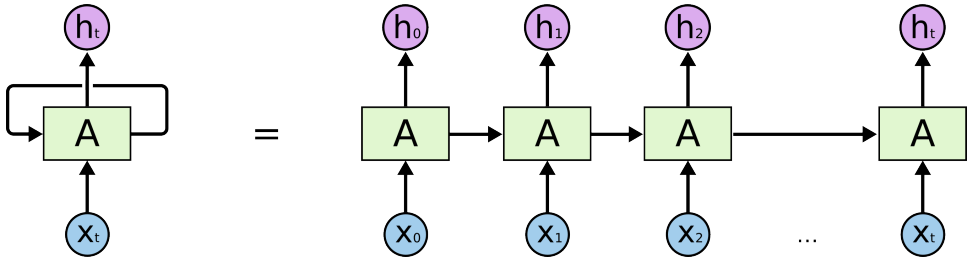


Figure 2.4: Figure shows an a network with self loop (RNN) and how it can be unrolled in time to form a linear architecture. [2]

2.3 Recurrent Neural Networks

A recurrent neural network is a neural network which can have self loops. A very simple example of a network with self loop is shown in figure 2.4. Figure 2.4 also shows that a network with self loop can be equivalently unrolled in time so that network equivalently accepts sequential structured inputs.

More formally, a recurrent neural network (RNN) is a neural network which can handle variable length sequences $X = \{x_1, ..., x_T\}$ and optionally a corresponding variable length output sequence $y = \{y_1, \ldots, y_T\}$. It maps X to Y using intermediate hidden representations h_t. RNN sequentially reads each time step x_t of input sequence X and updates its internal hidden representations h_t according to equation 2.11.

$$h_t = A_\theta(h_{t-1}, x_t), \tag{2.11}$$

where A_θ is a non-linear activation function parameterized by θ. When target sequence y is given, the RNN can be trained to sequentially make a prediction $\tilde{y}_t$ of actual output y_t at each time step t:

$$\tilde{y}_t = g_\theta(h_t, x_t), \tag{2.12}$$

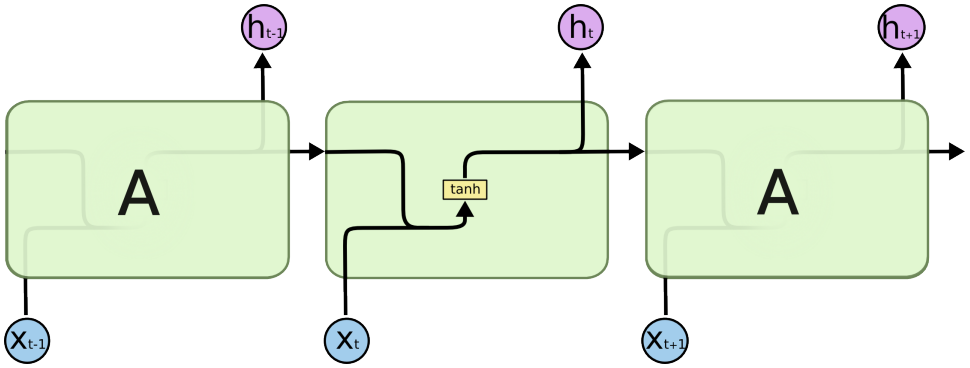
where $g_\theta(h_t, x_t)$ may be an arbitary, parametric function that is learned jointly as a part of whole network.

The recurrent activation function A in 2.11 maybe as simple as an affine transformation followed by an element-wise non-linearity A like tanh or sigmoid for example. To be precise equation 2.11 can be expanded as shown in 2.13-2.14.

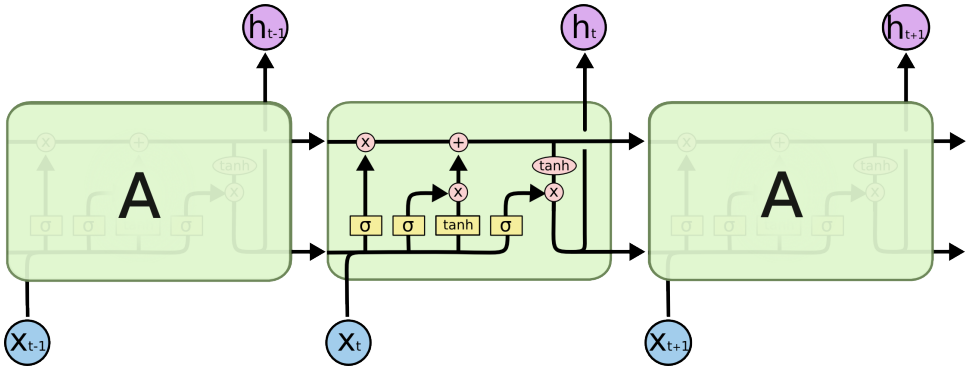
$$h_t^1 = A(W_{ih^1} x_t + W_{h^1 h^1} h_{t-1}^1 + b_h^1) \tag{2.13}$$

$$h_t^n = A(W_{ih^n} x_t + W_{h^{n-1} h^n} h_t^{n-1} + W_{h^n h^n} h_{t-1}^n + b_h^n) \tag{2.14}$$

where $W_{ih^1}, W_{h^1 h^1}, b_h^1, W_{ih^n}, W_{h^{n-1} h^n}, W_{h^n h^n}, b_h^n$ are weight matrices to be learnt.



(a) Vanilla RNN



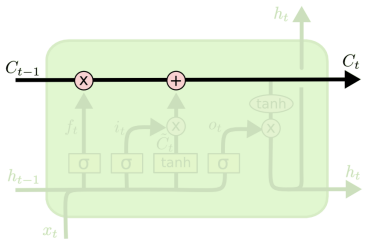
(b) RNN based on LSTM block

Figure 2.5: Figure 2.5a shows vanilla RNN architecture. Figure 2.5b shows LSTM architecture. [2]

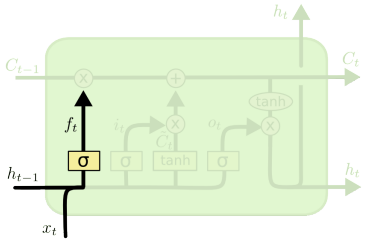
Recently more sophisticated recurrent activation functions like LSTM [21] or GRU [11] [12] have become more common and has also achieve state of art results for various applications. In next section we briefly describe LSTMs. Vanilla RNNs and LSTMs are shown in figure 2.5a and 2.5b respectively. More details about LSTM and why it works better than vanilla RNN is described in section 2.3.1.

2.3.1 Long Short Term Memory (LSTM) block

Long Short-Term Memory (Hochreiter and Schmidhuber, 1997) [21] is an RNN architecture that elegantly addresses the vanishing gradients problem using memory units. These linear units have a self-connection of strength 1 and a pair of auxiliary gating units that control the flow of information to and from the unit. When the gating units are shut, the gradients can flow through the memory unit without alteration for an indefinite amount of time, thus overcoming the vanishing gradients problem. While the gates never isolate the memory unit in practice, this reasoning shows that the LSTM addresses the vanishing gradients problem in at least some situations, and indeed, the LSTM easily solves a number of synthetic problems with pathological long-range temporal dependencies that were previously believed to be unsolvable by standard RNNs. Figure 2.5b shows a variant of LSTM proposed by Hochreiter and Schmidhuber [21] without peepholes. Equations 3.1-3.5 governs LSTM blocks.

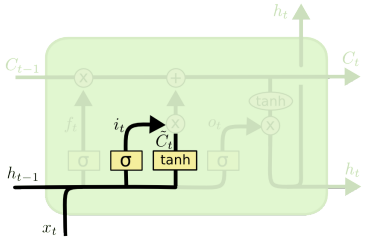


(a) Figure showing memory passing unattenuated.



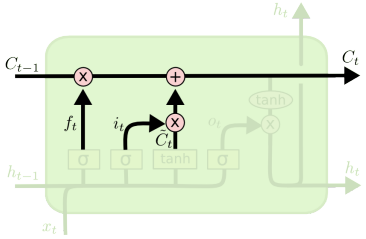
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

(b) Forget gate activation



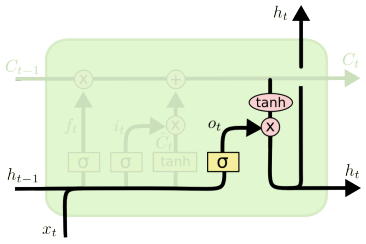
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

(c) Input gate activation



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

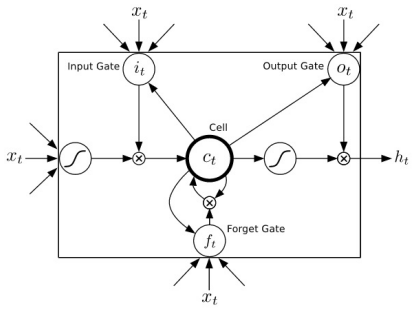
(d) Memory update



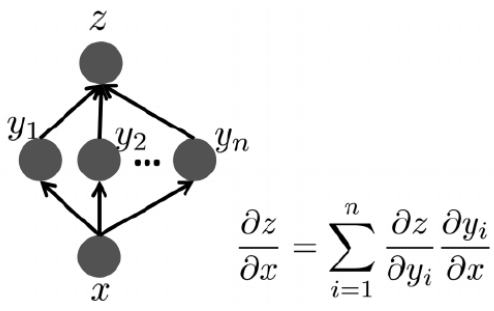
$$o_t = \sigma\left(W_o \; [\, h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

(e) Output

Figure 2.6: Figure 2.6a shows long term memory when forget gate is 1 and 0 input is added. Figure 2.6b - 2.6e shows LSTM block is composed and updated. [2]



(a) Long Short Term Memory



(b) Chain Rule with n nodes. [7]

Figure 2.7: 2.7a Long Short Term Memory [21] and 2.7b Chain rule on simple computational flow graph.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + w_{ci} \odot c_{t-1} + b_i) \tag{2.15}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + w_{cf} \odot c_{t-1} + b_f) \tag{2.16}$$

$$c_t = f_t c_{t-1} + i_t tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{2.17}$$

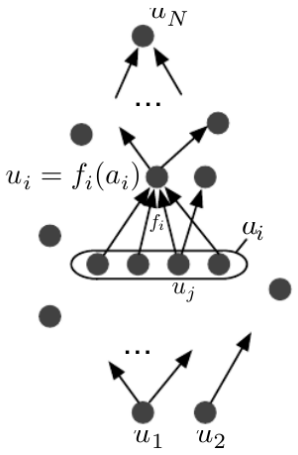
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + w_{co} \odot c_{t-1} + b_o) \tag{2.18}$$

$$h_t = o_t tanh(c_t) \tag{2.19}$$

Here, $W_{xi}, W_{hi}, w_{ci}, b_i$ are parameters associated with input gate. $W_{xo}, W_{ho}, w_{co}, b_o$ are parameters associated with output gate. $W_{xf}, W_{hf}, w_{cf}, b_f$ are parameters associated with forget gate. W_{xc}, W_{hc}, b_c are parameters associated with input which will directly modify the memory cells. The product $\odot$ denotes element-wise multiplication. The gating units are implemented by multiplication, so it is natural to restrict their domain to $[0, 1]^N$, which corresponds to the sigmoid nonlinearity. The other units do not have this restriction, so the tanh nonlinearity is more appropriate. The variant in figure 2.5b sets w_{ci}, w_{cf}, w_{co} to vector of zeros.

2.3.2 Why it works?

Why long term memory in LSTM works is a bit subtle. Figure 2.7a shows an LSTM block (originally proposed by [21]) which does not show directly how long term memory works. Figure 2.5b shows a different organization of same figure(but without peepholes). Figure 2.5b is broken into non-overlapping subsets to clearly show the dynamics of how it works(as shown in figure 2.6). Figure 2.6a clearly shows the existance of an unattenuated path in LSTM network hence giving it a capbility of long term memory in theory. Figure 2.6b shows how forget gate activations are computed using current input and previous hidden state. These activations are in [0,1] range and hence decays memory from previous time step. Figure 2.6c shows how new memory can be computed using input gate which will be combined with old memory. Figure 2.6d and 2.6e show how new hidden states and memory are computed.



Algorithm 1: Forward Computation

–initialization;
for $i = 1 : M$ **do**
  $u_i = x_i$;
end
–Forward Computation;
for $i = M + 1 : N$ **do**
  $a_i = (u_j)_{j \in parent(i)}$;
  $u_i = f_i(a_i)$;
end
return u_N;

Figure 2.8: Forward computational flow graph on a network.[7]

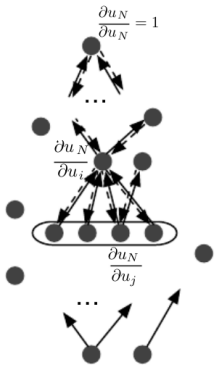
2.3.3 Computational Graph Implementation

Following [7], we describe computational flow graph and how a stack can be used to realize it with a python class(figure 2.11 2.10 at end of this chapter). Figure 2.7b shows chain rule of derivatives which is one of the fundamental ideas in learning of deep learning models. This could be extended to general graphs as showing in 2.9.

Every model is associated with a cost function (say J). Let the inputs/parameters θ to the model be $u_1, u_2, \ldots u_M$. Consider a general case when node $\{u_j\}_{j=1}^N$ form a directed acyclic graph that has J as its final node u_N, that depends of all the other nodes u_j. The back-propagation algorithm exploits the chain rule (figure 2.7b) for derivatives to compute $\frac{\partial J}{\partial u_j}$ when $\frac{\partial J}{\partial u_i}$ has already been computed for successors u_i of u_j in the graph. This recursion can be initialized by noting that $\frac{\partial J}{\partial u_N} = \frac{\partial J}{\partial J} = 1$ and at each step only requires to use the partial derivatives associated with each arc of the graph, $\frac{\partial u_i}{\partial u_j}$, when u_i is a successor of u_j. In general, $J(\theta)$ can be decomposed into simpler computations and a corresponding computational flow graph can be formed. Figure 2.8 summarized how to compute u_N. Figure 2.9 summarized how to compute the derivatives $(\frac{\partial u_N}{\partial u_i})_{i=1}^M$. Figure 2.10 show a simple usage of computational graph. Figure 2.11 shows an equivalent computational graph implementation using stack of derivative objects.

2.4 Summary

In this chapter, we gave a short overview of background required for further chapters. Feature representation of one of the important components in any prediction system. RBM is used to do unsupervised feature learning for OCR from data. RNN is used for sequence transcription in chapter 3 and 4. In chapter 3 we use a specific type of architecture of RNN for sequence transcription called sequence to sequence architecture.



Algorithm 2: Backward Computation

$\frac{\partial u_N}{\partial u_N} = 1$;
for $j = N - 1 : 1$ **do**
  $a_i = (u_j)_{j \in parent(i)}$;
  $\frac{\partial u_N}{\partial u_j} = \sum_{i:j \in parent(i)} \frac{\partial u_N}{\partial u_i} \frac{\partial f_i(a_i)}{\partial a_{i,\Pi(i,j)}}$;
  –where $\Pi(i, j)$ is the index of u_j as an argument to f_i;
end
return $(\frac{\partial u_N}{\partial u_i})_{i=1}^{M}$;

Figure 2.9: Backpropagation on computational flow graph on a network.[7]

```
#Use of computational flow graph for a simple function

model = {}
model['W'] = RandMat(10, 4, 0.08) # weights Mat
model['b'] = RandMat(10, 1, 0.08) # random input Mat
x = RandMat(4, 1, 0.08) # bias vector

# matrix multiply followed by bias offset. h is a Mat
G = Graph()
h = G.add(G.mul(model['W'], x), model['b']);

# the Graph structure keeps track of the connectivities between Mats
# we can now set the loss on h
h.dw[0] = 1.0; # say we want the first value to be lower

# propagate all gradients backwards through the graph
# starting with h, all the way down to W,x,b
# i.e. this sets .dw field for W,x,b with the gradients
G.backward();

# do a parameter update on W,b:
solver = Solver() # the Solver uses RMSProp
# update W and b, use learning rate of 0.01,
# regularization strength of 0.0001 and clip gradient magnitudes at 5.0
# for some model
solver.step(model, 0.01, 0.0001, 5.0)
```

Figure 2.10: A simple example of use of computational graph.

```
class Graph(object):
    """
    Inputs:
    -------
    needs_backprop boolean (optional) : whether ops should keep their
                                        derivative steps for backprop.

    """
    __slots__ = ["backprop", "needs_backprop"]

    def __init__(self, needs_backprop=True):
        self.needs_backprop = needs_backprop
        self.backprop = []

    def backward(self):
        """
        Travel graph backwards, applying the backpropagation
        steps.

        """
        for step in reversed(self.backprop):
            step()


    def tanh(self, matrix):
        """
        Hyperbolic tangent activation element wise on a matrix:

            > y = tanh( x )

        """
        out = Mat(matrix.n, matrix.d, np.tanh(matrix.w))

        if self.needs_backprop:
            def backward():
                matrix.dw += (1. - (out.w ** 2)) * out.dw
            self.backprop.append(backward)

        return out
    def mul(self, matrix1, matrix2):
        """
        Matrix dot product
        """
        assert(matrix1.d == matrix2.n), "matmul dimensions misaligned"
        out = Mat(matrix1.n, matrix2.d, matrix1.w.dot(matrix2.w))

        if self.needs_backprop:
            def backward():
                matrix1.dw += out.dw.dot(matrix2.w.T)
                matrix2.dw += matrix1.w.T.dot(out.dw)
            self.backprop.append(backward)
        return out

    def add(self, matrix1, matrix2):
        """
        Element add two matrices
        """
        assert(matrix1.n == matrix2.n and matrix1.d == matrix2.d)
        out = Mat(matrix1.n, matrix1.d, matrix1.w + matrix2.w)

        if self.needs_backprop:
            def backward():
                matrix1.dw += out.dw
                matrix2.dw += out.dw
            self.backprop.append(backward)
        return out

    def eltmul(self, matrix1, matrix2):
        """
        Element multply two matrices
        """
        assert(matrix1.n == matrix2.n and matrix1.d == matrix2.d)
        out = Mat(matrix1.n, matrix1.d, matrix1.w * matrix2.w)
        if self.needs_backprop:
            def backward():
                matrix1.dw += matrix2.w * out.dw
                matrix2.dw += matrix1.w * out.dw
            self.backprop.append(backward)
        return out
```

Figure 2.11: class structure for computational graph implementation using stack.

Chapter 3

Sequence to Sequence Learning for Optical Character Recognition

3.1 Introduction

Deep Neural Nets (DNNs) have become present day de-facto standard for any modern machine learning task. The flexibility and power of such structures have made them outperform other methods in solving some really complex problems of speech [20] and object [26] recognition. We exploit the power of such structures in an OCR based application for word prediction and retrieval with a single model. Optical character recognition (OCR) is the task of converting images of typed, handwritten or printed text into machine-encoded text. It is a method of digitizing printed texts so that it can be electronically edited, searched, stored more compactly, displayed on-line and used in machine processes such as machine translation, text-to-speech and text mining.

From character recognition to word prediction, OCRs in recent years have gained much awaited traction in mainstream applications. With its usage spanning across handwriting recognition, print text identification, language identification etc. OCRs have humongous untapped potential. In our present work we showcase an end-to-end, deep neural net, based architecture for word prediction and retrieval. We conceptualize the problem as that of a sequence to sequence learning and use RNN based architecture to first encode input to a fixed dimension feature and later decode it to variable length output. Recurrent Neural Networks (RNN) architecture has an innate ability to learn data with sequential or temporal structure. This makes them suitable for our application. Encoder LSTM network reads the input sequence one step at a time and converts it to an expressive fixed-dimensional vector representation. Decoder LSTM network in turn converts this fixed-dimensional vector (Figure 3.2b) to the text output.

Encoder-Decoder framework has been applied to many applications recently. Sutskever et al. [46] used recurrent encoder-decoder for character-level language modeling task where they predict the next character given the past predictions. It has also been used for language translation [47] where a complete sentence is given as input in one language and the decoder predicts a complete sentence in another language. Vinyals et al. [53] presented a model based on a deep recurrent architecture that can be used to generate natural sentences describing an image. They used a convolutional neural network as encoder and a recurrent decoder to describe images in natural language. Zaremba et al. [54] used

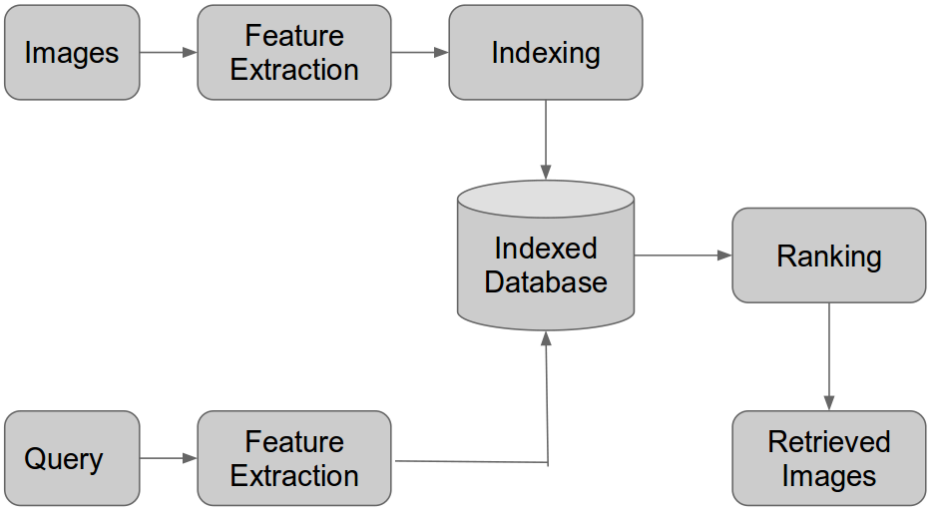
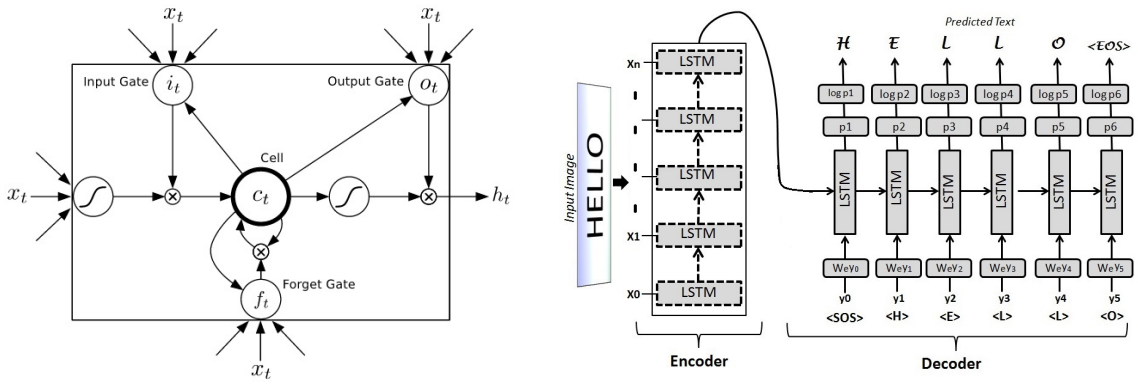


Figure 3.1: A typical retrieval system.

sequence to sequence learning by training them to evaluate short computer programs, a domain that have been seen as too complex in past. Vinyals et al. [52] proposed neural conversational networks based of sequence to sequence learning framework which converses by predicting the next sentence given the previous sentence or sentences in a conversation. Our work seeks close resemblance to previous works of [53, 47]. We formulate the OCR problem as a sequence to sequence mapping problem to convert an input (text) image to its corresponding text.

In this chapter, we investigate the expressiveness and learnability of LSTMs in sequence to sequence learning regime for printed text OCR. We demonstrate that sequence to sequence learning is suitable for word prediction task in a segmentation free setting. We even showcase the expressiveness of the learnt deep word image embeddings (from Encoder network of prediction) on image retrieval task. Converting variable length samples to fixed dimensional representation gives us access to fast and efficient methods for retrieval in fixed dimensional regime – approximate nearest neighbour. In (majority of) cases where standard LSTM models do not convert a variable length input to a fixed dimensional output, we are required to use Dynamic Time Warping (DTW) for retrieval which tends to be computationally expensive and slow.

Figure 3.1 shows a typical retrieval system. The performance of retrieval system is very sensitive to choice of feature extraction module. Traditionally we would have used one of hand engineered features like SIFT. In this work, we are interested to learn fixed dimensional feature extractor which captures the important features in the word images. Since we use an encoder-decoder framework for prediction, we can observe that the encoder will encoder the word image to a fixed dimensional representation which is used by decoder for prediction. We learn the parameters for encoder and decoder to maximize prediction performance. So inherently the encoder learns an intermediate representation which is useful for prediction. We also hope that these features are useful for retrieval. We also want to point out that



(a) LSTM block

(b) Encoder Decoder Framerwork for OCR

Figure 3.2: Figure 3.2a is the LSTM block comprised of input, output and forget gates. Figure 3.2b showcases the proposed recurrent encoder decoder framework for Optical Character Recognition. The Encoder LSTM section reads the input image and converts it to a fixed-dimensional vector representation. Decoder LSTM in turn generates the text output corresponding to fixed-dimensional vector representation.

(for example) the ranking module could be realized by approximate nearest neighbour if dimension of encoded representation is fixed across samples. This can be done by the architecture used in this chapter. Additionally we should also note that if we would have used standard RNN architecture for prediction and could have stripped off the output layer to use the learnt representation but the dimensionality of representation would not have been fixed across samples and we could not use efficient methods such as approximate nearest neighbour. In this setting, we would move to a less efficient mothods like DTW for sequence comparison. Hence, this work does help in prediction as well as efficient retrieval with same network in recurrent neural network regime. To summarize, feature extraction can be done by encoder and approximate nearest neighbour can be used for efficient ranking module in fixed dimensional representation setting.

3.2 Sequence learning

A recurrent neural network (RNN) is a neural network with cyclic connections between its units. These cycles create a concept of 'internal memory' in network and thus differentiate RNNs from other feed forward networks. The internal memory of RNN can be used to process arbitrary sequences of inputs – given a variable length input sequences X we can generate corresponding variable length output sequence Y. This is done by sequentially reading each time-step x_t of input sequence X and updating its internal hidden representations h_t. More sophisticated recurrent activation functions like LSTM [21] and GRU [11, 12] have become more common in recent days. They perform better when compared to other vanilla RNN implementations.

Long Short-Term Memory [21] is a RNN architecture that elegantly addresses the vanishing gradients problem using 'memory units'. These linear units have a self-connection of strength 1 and a pair of auxiliary 'gating units' that control the flow of information to and from the unit. Equations 3.1-3.5 describe LSTM blocks.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + w_{ci} \odot c_{t-1} + b_i) \tag{3.1}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + w_{cf} \odot c_{t-1} + b_f) \tag{3.2}$$

$$c_t = f_t c_{t-1} + i_t tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{3.3}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + w_{co} \odot c_{t-1} + b_o) \tag{3.4}$$

$$h_t = o_t tanh(c_t) \tag{3.5}$$

Here, $W_{xi}, W_{hi}, w_{ci}, b_i$ are input gate parameters. W_{xo}, W_{ho}, w_{co}, b_o are output gate parameters. W_{xf}, W_{hf}, w_{cf}, b_f are forget gate parameters. W_{xc}, W_{hc}, b_c are parameters associated with input which directly modify the memory cells. The product $\odot$ denotes element-wise multiplication. The gating units are implemented by multiplication, so it is natural to restrict their domain to $[0, 1]^N$, which corresponds to the sigmoid non-linearity. The other units do not have this restriction, so the tanh non-linearity is more appropriate. We use collection of such units (Figure 3.2a) to describe an encoder-decoder framework for the OCR task. We formulate the task of OCR prediction as a mapping problem between structured input (image) and structured output (text).

Let, $\{I_i, Y_i\}_{i=1}^N$ define our dataset with I_i being image and Y_i be the corresponding text label. Image I_i lies in $\mathbb{R}^{H \times T_i}$, where H is word image height (common for all images) and T_i is the width of i^{th} word image. We represent both image and label as a sequence – Image $I_i = \{x_1, x_2, ..., x_{T_i}\}$ is sequence of T_i vectors lying in $\{0, 1\}^H$ (x_i is a i_{th} pixel column) and $Y_i = \{y_1, y_2, ..., y_{M_i}\}$ is corresponding label which is a sequence of M_i unicode characters. We learn a mapping from image to text in two steps – (a) $f_{encoder} : I_i \rightarrow z_i$, maps an image I_i to a latent fixed dimensional representation z_i (b) $f_{decoder} : z_i \rightarrow Y_i$ maps it to the output text sequence Y_i. Unlike CTC layer based sequence prediction [16] we don't have any constraint on length of sequences I_i and Y_i. Equations 3.6-3.7 formally describe the idea. The choice of $f_{encoder}$ and $f_{decoder}$ depends on type of input and output respectively. Both input and output correspond to a sequence in our case, hence we use recurrent encoder and recurrent decoder formulation.

$$z_i = f_{encoder}(I_i) \tag{3.6}$$

$$P(Y_i|I_i) = f_{decoder}(z_i) \tag{3.7}$$

3.2.1 Encoder: LSTM based Word Image Reader

To describe the formulation we use vanilla RNNs with L hidden layer and no output layer. The encoder reads $I_i = \{x_1, x_2, ..., x_{T_i}\}$ one step at a time from x_1 to x_{T_i}. Hidden state h_t^n is updated using

equations 3.8-3.9 using current input x_t and previous hidden state $\{h_{t-1}^n\}_{n=1}^L$ where L is number of hidden layers in RNN.

$$h_t^1 = relu(W_{ih^1}x_t + W_{h^1h^1}h_{t-1}^1 + b_h^1) \tag{3.8}$$
$$h_t^n = relu(W_{h^{n-1}h^n}h_t^{n-1} + W_{h^nh^n}h_{t-1}^n + b_h^n) \tag{3.9}$$

where $W_{ih^1}, W_{h^1h^1}, b_h^1, W_{ih^n}, W_{h^{n-1}h^n}, W_{h^nh^n}, b_h^n$ are parameters to be learned.

To obtain our fixed dimensional latent representation z_i we use the final hidden states $\{h_{T_i}^n\}_{n=1}^L$.

$$z_i = \{h_{T_i}^n\}_{n=1}^L \tag{3.10}$$

It should be noted that we have used LSTM networks instead of vanilla RNNs and no output layer for encoder is needed. The hidden states of last step T_i are used as initial state of decoder network.

3.2.2 Decoder: LSTM based Word Predictor

Similar to encoder, we describe the idea using vanilla RNNs with L hidden layers and softmax output layer. The goal of word predictor is to estimate the conditional probability $p(Y_i|I_i)$ as shown in equation 3.11-3.12, where I_i is image input sequence and Y_i is output sequence.

$$p(Y_i|I_i) = p(Y_i = \{y_1, \ldots, y_{T'}\}|I_i = \{x_1, \ldots, x_{T_i}\}) \tag{3.11}$$
$$= \prod_{t=1}^{T'} p(y_t|I_i, y_1, \ldots, y_{t-1}) \tag{3.12}$$

The updates for single step for RNN is described in equations 3.13-3.16. The hidden state h_t^n is updated using equations 3.13 - 3.14 using current input x_t and previous hidden state $\{h_{t-1}^n\}_{n=1}^L$, where L is number of hidden layers in RNN. The hidden activations, h_{t-1}^L are used to predict the output at step t using equations 3.15-3.16. x_t is the embedding of the most probable state in previous step $t-1$ shown in equation 3.18.

$$h_t^1 = relu(W_{ih^1}x_t + W_{h^1h^1}h_{t-1}^1 + b_h^1) \tag{3.13}$$
$$h_t^n = relu(W_{h^{n-1}h^n}h_t^{n-1} + W_{h^nh^n}h_{t-1}^n + b_h^n) \tag{3.14}$$
$$o_t = W_{h^Lo}h_t^L + b_o \tag{3.15}$$
$$p_t = sofmax(o_t) \tag{3.16}$$

where $W_{ih^1}, W_{h^1h^1}, b_h^1, W_{ih^n}, W_{h^{n-1}h^n}, W_{h^nh^n}, b_h^n, W_{h^Lo}, b_o$ are parameters to be learned.

Decoding begins at $t = 0$ with $\langle SOS \rangle$ marker (Start Of Sequence). The state $t = -1$ is initialized with the final state z_i of encoder as shown in equation 3.17. The 1_{st} character is predicted using the em-

bedding for $\tilde{y}_0 = \langle SOS \rangle$ as input $x_0 = W_e \tilde{y}_0$ and output $p_1(y_1|I_i, y_0)$ where W_e is the embedding matrix for characters. Every t_{th} character is predicted using the embedding for $\tilde{y}_t = \arg\max p_t(y_t|I_i, y_{<t})$ as input, as shown in equation 3.18 and output $p_t(y_t|I_i, y_{<t})$. This is iterated till $t = T'$ where $y_{T'} = \langle EOS \rangle$ (End Of Sequence). T' is not known in priori, $\langle EOS \rangle$ marker instantiates the value of T'.

$$h_{-1} = f_{encoder}(I) \tag{3.17}$$

$$x_t = W_e \tilde{Y}_t \tag{3.18}$$

$$p_{t+1} = f_{decoder}(x_t) \tag{3.19}$$

W_e in equation 3.18 is the embedding matrix for characters. It should be (again) noted that we use LSTM networks (equations 3.1 - 3.5) instead of vanilla RNNs (equations 3.13, 3.14).

3.2.3 Training

The model described in section 3.2.1 and 3.2.2 is trained to predict characters of the input word (image) sequence. The input at time t of decoder is an embedding of the output of time $t-1$ and loss L for a sample (I, Y) is described by equation 3.20.

$$L(I, Y) = -\log p(Y|I; \theta) \tag{3.20}$$

here, $\log p(Y|I;\theta) = \sum_{t=0}^{M} \log p(y_t|I, y_0, \ldots, y_{t-1}; \theta)$ is the log probability of correct symbol at each step. We search for the parameters θ^* which minimize the expected loss over true distribution $P(I, Y)$ given in equation 3.21. This distribution is unknown and can be approximated with empirical distribution $\tilde{P}(I, Y)$ given in equation 3.22-3.23.

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{P(I,Y)} L(I, Y; \theta) \tag{3.21}$$

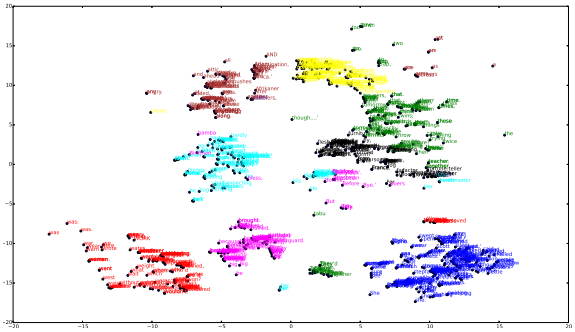
$$\approx \arg\min_{\theta} \mathbb{E}_{\tilde{P}(I,Y)} L(I, Y; \theta) \tag{3.22}$$

$$= \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L(I_i, Y_i; \theta) \tag{3.23}$$

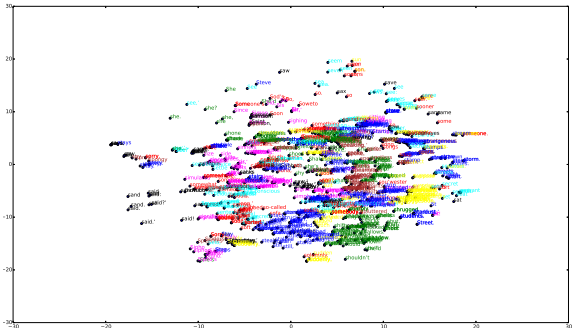
We use RMS prop [50] with momentum to optimize (minimize) the above loss with respect to θ.

3.3 Implementation Details

Keeping the aspect ratio of input images intact we resize them to height of 30 pixels. The binarized images are then used as input to the two hidden layer LSTM encoder-decoder architecture. We choose



(a) TSNE plots for feature representation of unique words starting with top eight most frequent alphabets(S, T, W, H, B, C, F, A including lowercase). (Color based on first alphabet of words)



(b) TSNE plots for feature representation of unique words starting with S and top eight *second* alphabets (t, h, o, e, i, u, a, p) in word with respect to population in S. (Color based on second alphabet of words starting with S.)

Figure 3.3: TSNE plots characterizing the quality of feature representations computed by encoder. Each word has a unique high dimensional feature representation which is then embedded in a two dimensional space (using TSNE) and is visualized using scatter plots. Similar words are grouped together (shown with various colours) and dissimilar words tend to get far away. As shown in the figure, (a) words beginning with same alphabets belong to same clusters and rest are in other clusters (b) words beginning with 'S' and having same second alphabet belong to same clusters and rest are in other clusters. (Readers are requested to magnify the graphs to look into the intricate details of clusters)

embedding size of 25 for our experiments. The dimensionality of output layer in decoder is equal to number of unique symbols in dataset. We use a RMS prop [50] with step size of 0.0001 and momentum of 0.99 without any regularization. We use gradient clipping threshold of 5. The parameters values are verified and set using a validation set. We use Python's numpy library to implement LSTM based architecture. The network is built using computational graphs.

3.4 Experiments

We demonstrate the utility of the proposed recurrent encoder-decoder framework by two related but independent tasks. Both prediction and retrieval experiments are independently baselined.

Prediction: We use 295K annotated English word images from seven books for our experiments. We perform three way data split for all our experiments – 60% training, 20% validation and remaining 20% for testing. Results are reported by computing 'label error rate'. Label error rate is defined as ratio of sum of insertions, deletions and substitutions relative to length of ground truth over dataset. We compare the results of our pipeline with state-of-art LSTM-CTC [16], an open-source OCR TESSERACT [3] and a commercial OCR ABBYY [1].

[1] Original images are used as input.
[2] Images are padded along boundary pixels for better results.
[3] Augmented Profiles [28]

Features	Dim	mAP-100	mAP-5000
BOW	400	0.5503	0.33
BOW	2000	0.6321	-
AUGMENTED PROFILES[3]	247	0.7371	0.6189
LSTM-ENCODER	400	0.7402 $(h1 - h2)$ 0.8521 $(c1 - c2)$	0.8521
OCR - TESSERACT	-	0.6594	0.7095
OCR - ABBYY	-	0.8583	0.872

Table 3.1: mAP computed for various methods: mAP-n stands for mean average precision computed over top n retrievals. hi is hidden representation of layer-i at last timestep of input sequence. ci is memory for layer-i at last timestep of input sequence. A-B is the concatenation of representation A and B. For example, $h1 - h2$ represents concatenation of both $h1$ and $h2$.

Model	Label error(%)
ABBYY[2]	1.84
TESSERACT[1]	35.80
TESSERACT[2]	16.95
RNN ENCODER-DECODER	35.57
LSTM-CTC [16]	0.84
LSTM ENCODER-DECODER	0.84

Feature	mAP-100
h1-h2	0.7239
c1-c2	0.8548
h1-h2-c1-c2 L1	0.8078
h1-h2-c1-c2 L2	0.7834
h1-h2-c1-c2	0.8545

Table 3.2: **Left:** Label Error Rate comparison of RNN-CTC and Recurrent encoder-decoder. **Right:** Effect of different concatenation and normalization on features from LSTM-Encoder. L1 and L2 represent normalization scheme.

Table 3.3: Qualitative results for retrieval: Comparison of retrieval scheme using simple Bag of Words (BoW) and proposed Deep Word Image Embeddings (DWIE). We use text labels (for more clarity in presentation) of both query and retrieved images to illustrate difference in performance of two approaches. The proposed approach is able to learn representations useful for retrieval task.

	DWIE	**BoW**	**DWIE**	**BoW**	**DWIE**	**BoW**	**DWIE**	**BoW**
Query (→) / Retrival (↓)	A.	A.	following	following	returned	returned	For	For
R1	A.	”A	following	long	returned	turned	For	For
R2	A.	A	following	long	returned	read	For	For
R3	A.	A	following	long	returned	refused	For	For
R4	A.	At	following	following	returned	retorted	For	For
R5	A.	Ah	following	long	returned	lifted	For	For
R6	A	A	following	flowing	returned	ceased	For	For
R7	A	A	following	long	returned	rolled	For	For
R8	A	A	folding	long	returned	and	For	For
R9	A	A	follow	long	returned	carried	For	for
R10	A	A	followed,	following	returned	red	For	for
R11	A	A	foolishly	long	returned	Head	For	for
R12	A	A	fellow,	along	returned	raised	For	for
R13	A	A	foolscap	long	returned	caused	For	for
R14	A	A	fellow,	following	returned	turned	For	for
R15	A	A	foliage.	long	returned	turned	For	for
R16	A	A	fellow,	long	returned.	road	For	for
R17	A	A	fellow,	long	retired	and	For	for
R18	A	A	fellow,	long	retorted	returned	Fer-	For
R19	A	As	falling	long	return.”	God	Fer-	for
R20	A	Af-	follow,”	closing	return	acted	Fer-	For
# relevant matches in corpus	5	5	7	7	15	15	17	17

Retrieval: We use 108K annotated word images from book titled ‘Adventures of Sherlock Holmes’ for retrieval experiments. In all 43K word images are used for querying the retrieval system. We compare retrieval results with SIFT [32] based bag of words representation, augmented profiles [28]and commercial OCR ABBYY.

3.4.1 Results and Discussion

Table 3.2 exhibits prediction baseline. We observe that LSTM ENCODER-DECODER outperforms vanilla RNN ENCODER-DECODER by significant margin. It is competitive LSTM with CTC output layer and ABBYY. When compared to CTC layer based LSTM networks, our network requires more memory space. The strength of our network is ‘fixed length representation’ for variable length input which enables us to perform better and fast retrieval.

Table 3.4: Qualitative results for prediction: We illustrate some of the success and failure cases of our word prediction output. The figure highlights the cases where both commercial and open-source OCRs fail

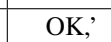
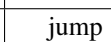
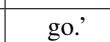
Query Image(→)				
True Label	mom	OK,'	jump	go.'
Tessaract [2]	IDOITI	ox,'	iump	80.
Abbyy [2]	UJOUJ	ok;	duinl	g-'
LSTM-ED	mom	OK,'	jump	go.'

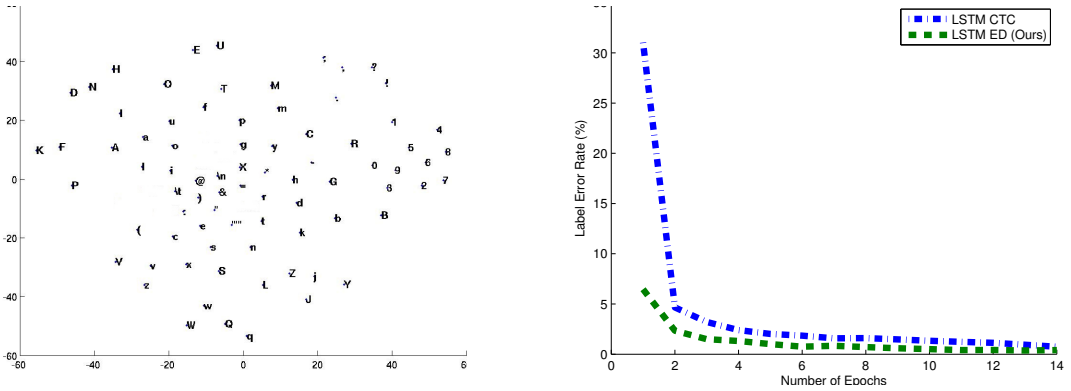
Table 3.1 depicts retrieval baseline. Features from LSTM encoder (refereed as deep word image embedding (DWIE)) are used for comparisons with other state-of-art results. We observe that DWIE features significantly outperform SIFT [32] based bag of words (BOW) and augmented profiles[28]. When compared to ABBYY, DWIE features perform notch better for top 5000 retrieval but perform similar for top 100 retrieval. The memory states at last position of each sequence are used as DWIE features. Various normalization (L1 and L2) and augmentations with hidden states were tried out as shown in Table 3.2.

Table 3.3 demonstrates the qualitative performance of retrieval system using both deep word image embedding (DWIE) and bag of words (BOW) models. The table illustrates top 20 retrievals using both the methods. We observe the proposed embeddings to be better than naive (BOW) in such settings. In majority of the cases we find all relevant(exact) matches at top in case of deep embeddings, which is not the case with (BOW) model. DWIE seems highly sensitive to small images components like '.' (for query 'A.') which is not the case with BOW model. Simple BOW fails to recover any relevant samples for query 'A.' in top 20 retrievals.

Figure 3.3 is culmination of T-SNE [51] plots of word image encodings. We show two levels of visualization along with groupings in context of word image representation. It's clear from the figure 3.3a that representation is dominated by first character of the word in word image. Sequence of correct encodings play a major role in full word prediction – a wrong letter prediction in early stages would result in overall invalid word prediction.

Figure 3.4a is a plot of learnt embeddings which shows relative similarities of characters. The similarities are both due to structure and language of characters – (i) all the numbers (0-9) are clustered together (ii) punctuations are clustered at top right of graph (iii) upper case and lower case characters tend to cluster together, viz. (m,M), (v,V), (a,A) etc. As embeddings are learnt jointly while minimizing cost for correct predictions, they tend to have relative similarities of characters based jointly on structure in image space and language in output space. Figure 3.4b illustrates training label error rate for various learning models – LSTM with CTC output layer and LSTM encoder-decoder.

[4]Standard LSTM-CTC implementation [16].



(a) Learnt embedding of characters using tSNE.

(b) Convergence of LSTM-CTC [4] and LSTM-ED.

Figure 3.4: Plot showing character embedding and convergence of LSTM-CTC [4] and LSTM-ED.

3.5 Summary

We demonstrate the applicability of sequence to sequence learning for word prediction in printed text OCR. Fixed length representation for variable length input using a recurrent encoder-decoder architecture sets us apart from present day state of the art. We believe with enough memory space availability, sequence to sequence regime could be a nice and efficient alternative for CTC based networks. The network could well be extended for other deep recurrent architectures with variable length inputs, e.g. attention based model to describe the image contents etc.

Chapter 4

Deep Profiles

4.1 Introduction

Learning appropriate representation for characterizing images have gained lots of attention in the recent years with deep and rich representations [27, 6]. In this work, we investigate the possibility of learning features for OCRs. We are especially interested in the recognition solutions based on Recurrent Neural Networks (RNN) which has emerged as the state of the art for recognizing printed [8], handwritten [17] and natural scene text [44]. This is especially relevant since the RNN based recognition schemes have been using mostly simple features such as profile representations. In this work, we learn a set of features and demonstrate more than 50% reduction in error rates for five different languages.

Conventional solutions to the OCR problem often demands a set of handcrafted features based on the shape of the characters and properties of the characters in the languages. Characterizing the shape or appearance with the help of statistical or structural features is very popular in literature. However, engineering the appropriate set of features have remained as more of an art than science. Handcrafting features for OCRs is a script-dependent process, which can be tedious. On the other hand, RNNs suffer from the burden of simplistic features. RNNs are a multilayer architecture where each layer "abstracts" out useful information from the previous layer. Thus, when presented with simplistic features, extracting complex concepts from them require the use of multiple layers. This leads to two problems with their training. First, having multiple layers increases the amount of data required to train RNNs, since naturally more data is required to get better estimates of huge number of parameters. Second, each iteration of the training is longer since information has to be propagated through a bigger network. In this work, we try to bridge these two different attempts by learning a set of features that are compatible with the word prediction architectures based on RNNs. To alleviate problems with both these approaches, we explore creating a representation that is not handcrafted but *learnt* from data. Such an approach simultaneously increases the portability of OCR architectures to multiple languages while reducing the time required for training the word recognition engine.

We use Restricted Boltzman Machines (RBMs) for learning features in an unsupervised manner. We start with a simple representation and learn the language specific features by finding the compact low-

dimensional representation using RBMs. We learn a hierarchical representation by stacking RBMs with compressive architecture. The choice of RBM is supported by its attractive properties. First, it is an unsupervised feature learner and has an expressive representation. Second, it can be trained efficiently using contrastive divergence [19, 49]. It is also possible to stack them so that they can learn hierarchical organization of explanatory factors in the data. Stacked RBM fits in our setting because we do not have well segmented labeled character windows for word prediction in OCR problem.

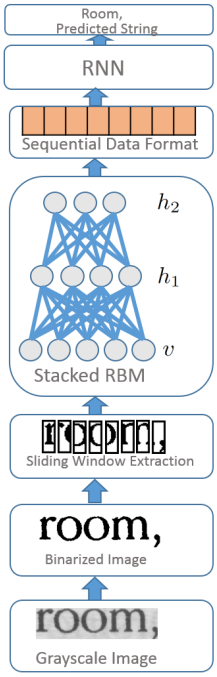
There have been many previous attempts in learning the feature representations for OCRs. Use of Principal Component Analysis (PCA) for creating a set of data specific basis vectors is a well known technique in OCRs. PCAs are linear and do not encode the nonlinear relationships in the data. RBMs, on the other hand, have the inherent capability to capture the nonlinear relationships. For the word prediction problem, an alternative to RNNs have been Hidden Markov Models (HMM). HMMs, which are popular for speech recognition had also seen the benefits of feature learning in offline handwriting recognition [18]. Chandra *et al.* [10] proposed a mixture model K-RBM, to learn feature representations on image patches and demonstrated its performance on object recognition.

In this work, we demonstrate our method on five different languages: English, Marathi, Telugu, Kannada and Malayalam. In all these cases, our feature learning method consistently improves the accuracy over profile features [25] which is a typical example of a hand crafted feature for the word recognition. Other advantages include that the learnt features will be of lower dimension and therefore are compact, efficient and takes smaller training time. We also demonstrate that, this results in faster convergence of RNN, which implies that the learnt features are more appropriate for the problem of interest.

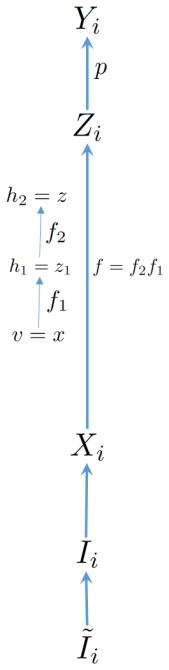
4.2 Feature Learning for word prediction

We propose a framework for word prediction for printed text where instead of using features like profile [25], we give learned representations to the predictor. Our proposed pipeline is a cascade of two models namely a stacked RBM for learning representations followed by a RNN which is used as a word predictor (Figure 4.1a). The aim is to give compact, rich and expressive features to the word predictor which is learnt from the data in an unsupervised manner. Our representations are fairly adaptable to change in languages (shown in section 4.3) which can have varying structural and statistical properties in their symbol set. Stacked RBMs also give compact and expressive representations because of being multi-layered distributed representations [5].

We start with a printed text image with variable width across samples. We first convert the image into sequential data by extracting a sequence of windows from the image by a sliding a window of fixed width and vectorize each window in that sequence by stacking all the pixels. Here each window might contain a partial, full or combination of multiple characters. The dimensionality of this data is reduced using stacked RBMs. Stacked RBM are unsupervised feature learners and need no supervision while learning. This is important because we do not have segmented characters. We also preferred an unsupervised



(a) Word prediction pipeline



(b) Symbolic Pipeline

Figure 4.1: Word Prediction Pipeline. (4.1a) Pictorial pipeline and (4.1b) Symbolic pipeline. Input image $\tilde{I}_i$ is converted to binarized image I_i, which is converted into a sequence by running a sliding window. Each vector in the sequence goes through a non-linear transformation $f = f_2 f_1$ using stacked RBM. These encoded windows collected in transformed sequence Z_i which is then given to BLSTM predictor p which outputs string Y_i.

setting as labels for partial character windows is difficult to obtain and are not unique. It also gives explicit non-linear projections which is desirable due to that fact that once learning is done we only need the learnt parameters to describe the model unlike kernel projection methods. Once parameters of RBM is learnt, we only need to do a matrix vector multiplication followed by a non-linearity for each layer to obtain our non-linear projection which can be done efficiently.

Let $\{\tilde{I}_i, Y_i\}_{i=1}^N$ be our dataset and image $\tilde{I}_i$ lies in $\mathbb{R}^{H \times W_i}$ where H is word image height common for all images and W_i is the width of i^{th} word image, $Y_i = \{y_1, y_2, ..., y_{M_i}\}$ be the corresponding label which is a sequence of M_i unicode characters. Let $\{I_i, Y_i\}_{i=1}^N$ be the set obtained by binarizing all images $\{\tilde{I}_i\}_{i=1}^N$. We represent I_i as a sequence $X_i = \{x_1, x_2, ..., x_{L_i}\}$ of L_i vectors lying in $\{0,1\}^{d_v}$ by running a sliding window of dimensions $w_h \times w_w$ with a step size of s and unrolling the windows obtained into a vector of length $d_v = w_h \times w_w$. Here, w_h is the height of the sliding window and is equal to the image height H, w_w is the width of the sliding window and d_v is the length of vectors in sequences after unrolling the window into a vector.

4.2.1 Feature Learning using Stacked RBM

Let $\{X_i\}_{i=1}^N$ be the unlabeled dataset used for training stacked RBM and X_i be a vector sequence of i^{th} printed word image obtained by sliding a window over corresponding I_i. Unsupervised feature learning in stacked RBM can be seen as learning a non-linear projection $f : x \rightarrow z \mid x \in \{0,1\}^{d_v}, z \in \mathbb{R}^{d_h}$. Therefore, we project our dataset $\{X_i = \{x_1, x_2, ..., x_{L_i}\}\}_{i=1}^N \xrightarrow{f} \{Z_i = \{z_1, z_2, ..., z_{L_i}\}\}_{i=1}^N$ using the learned non-linear projection f. Here, Z_i's are vector sequences where each vector in the sequence is a learnt compact and expressive representation. In the following paragraphs we focus on non-linear projection f using stacked RBM and RBM is trained on a sampled subset from collection of all vectors in all sequences X_i.

Now we briefly review Restricted Boltzman Machines (RBM) which is the core component used in our feature learning step. An RBM defines a distribution over $(\mathbf{v}, \mathbf{h}) \in \{0,1\}^{n_v \times n_h}$, $\mathbf{v}$ being visible variables and $\mathbf{h}$ being hidden variables. Here, $\mathbf{v} \in \{0,1\}^{n_v}$ corresponds to each vector $x_j \mid j \in \{1, 2, ..., L_i\}$ in each sequence X_i and $\mathbf{h} \in \{0,1\}^{n_h}$ corresponds to learned representation from corresponding RBM in stack. It can be observed in stacked RBM block in Figure 4.1a, any two consecutive layers constitute an RBM, like $(\mathbf{v}, \mathbf{h}_1)$ and $(\mathbf{h}_1, \mathbf{h}_2)$ are two RBMs stacked to form a two layer stacked RBM. Equation 4.1, 4.2, 4.3 and 4.4 describe the energy function, probability distribution, and the conditional distribution of random variables of one layer conditioned on the other and vice versa respectively.

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^T W \mathbf{h} - b^T \mathbf{v} - c^T \mathbf{h} \tag{4.1}$$

$$P(\mathbf{v}, \mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} \tag{4.2}$$

$$P(h_j = 1|\mathbf{v}) = sigmoid(W^T \mathbf{v} + c)_j \tag{4.3}$$

$$P(v_i = 1|\mathbf{h}) = sigmoid(W\mathbf{h} + b)_i \tag{4.4}$$

Here $W \in \mathbb{R}^{n_v \times n_h}$ n_v, n_h are number of visible and number of hidden nodes respectively in the RBM. $W_{i,j}$ is symmetric weights between v_i and h_j. W, b, c are the parameters of the model and needs to be learned from the data by maximizing the log likelihood of the data with respect to parameters. The learning can be efficiently done by contrastive divergence (CD)[19]. Lets consider a two layer stacked RBM as shown in stacked RBM block of Figure 4.1a, Here for first RBM, $n_v = d_v$ and $n_h = n_{h_1}$ and for last RBM in stack $n_v = n_{h_1}$ and $n_h = d_h$.

Now we stack RBMs to build our projection f. We achieve this by cascading a series of RBMs and learning a series of latent intermediate representations, for example: $x \xrightarrow{f1} z_1 \xrightarrow{f2} z_2 \xrightarrow{f3} z$ in 3 layer stacked RBM. Here, f can be seen as cascade of three projections $f = f_3 f_2 f_1$ and each f_1, f_2, f_3 can be learned by an RBM which gives f some characteristics like efficient, compact, non-linear and hierarchical representation [6]. We learn the intermediate latent representation z_1 using our dataset $\{X_i\}_{i=1}^N$ as input to RBM and the intermediate projection of dataset $\{X_i\}_{i=1}^N$ be $\{Z_{1i}\}_{i=1}^N$. We now get next intermediate

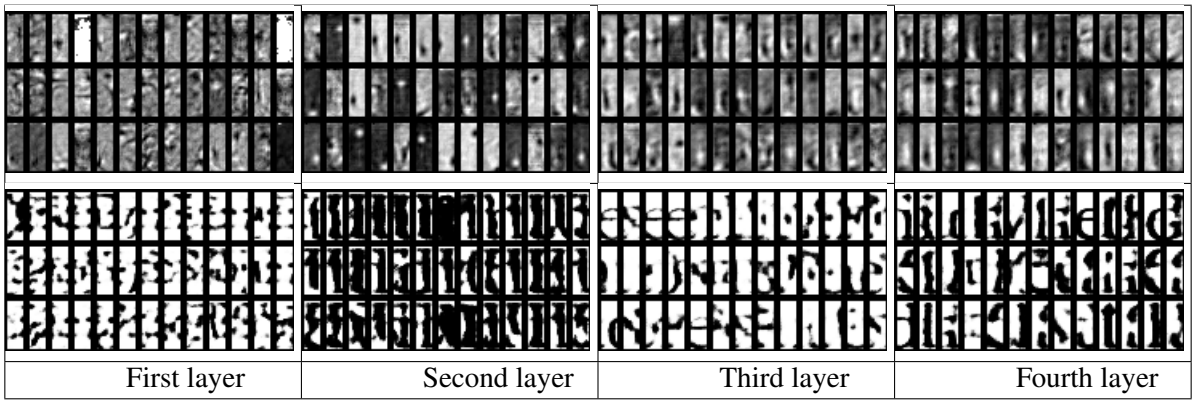


Figure 4.2: Visualizing hidden layers in original image space using two methods of visualization, (First Row) Linear combination of previous layers (Second Row) Sampling.

representation z_2 by using $\{Z_{1i}\}_{i=1}^{N}$ as dataset for next RBM in cascade and keep repeating this till we get our final representation z. These representations can be used to represent the whole dataset $\{X_i\}_{i=1}^{N}$ as $\{Z_i\}_{i=1}^{N}$ which is input to Recurrent Neural Network(RNN) as discussed below.

4.2.2 Recurrent Neural Network(RNN)

We obtain a dataset $\{Z_i, Y_i\}_{i=1}^{N}$ from projecting $\{X_i, Y_i\}_{i=1}^{N}$ using stacked RBM as discussed earlier. Z_i and X_i are corresponding vector sequences of dimensionality d_h and d_v respectively, given sequence length remains same. We want to learn a mapping $p : Z_i \rightarrow Y_i$ from the data $\{Z_i, Y_i\}_{i=1}^{N}$. The predictor p can be seen as a mapping which tries to map each $Z_i \xrightarrow{p} Y_i$. Our choice for this predictor p is Bidirectional LSTM(BLSTM) which is a Recurrent Neural Network(RNN).

LSTM networks can remember long range context over several timesteps in the sequence. The output layer of our preferred network is Connectionist Temporal Classification (CTC) layer[16] which allows the network to handle a sequence of unsegmented features and perfectly match our requirement i.e. we learn features on set acquired from sliding windows which are not well segmented symbols. This enables us to use segmentation free model. LSTM networks are also easily trainable unlike general recurrent neural networks.

A BLSTM network contains two LSTM networks in which one network takes input from beginning to the end while other network takes the input from end to beginning. The activations from both networks are then used to predict the final output which is done using Connectionist Temporal Classification (CTC)[16]. CTC layer directly outputs the probability distribution over label sequences. The CTC objective function is defined as the negative log probability of the network correctly labeling the entire training set. Details for BLSTM and CTC algorithm can be found in [17, 16]. The measure of performance for word prediction is label error rate and sequence error rate. Label error rate is ratio of sum

of insertions, deletions and substitutions relative to length of ground truth and sequence error rate is the ratio of number of words misclassified to total number of words.

4.2.3 Visualization

In Figure 4.2, we show qualitative interpretation of high level features represented by stacked RBM. Here, the goal is to visualize hidden nodes in arbitrary layer of a deep network in image space. We use two visualization methods as also discussed in [14], namely by sampling and linear combination of previous units. These visualizations also help to appreciate hierarchical representations from deep networks and help us understand what models have learned. It is not surprising that the visualization associated with hidden nodes in first hidden layer are the weights itself but we want to visualize higher layers, tools for which is described in [14]. We use two visualization techniques, namely sampling and linear combination of previous units as described below. Our observations also suggest that sampling methods produce better visualization than linear combination method for this application as shown in Figure 4.2.

4.2.3.1 Linear combination of previous units

Simplest possible way to visualize hidden units of stacked RBM is by linearly combining of features of hidden units of previous layers. In order to visualize n_l hidden units of l_{th} layer, we plot feature maps formed by $F = W_1 W_2 \cdots W_l \in \mathbb{R}^{d_v \times n_l}$ where each column of F can be formed into $w_h \times w_w$ feature maps to be visualized. First row in Figure 4.2 shows visualization of hidden units of different layers. This method is simple but looses non-linearly and is inferior to other alternatives to visualization as mentioned in [14].

4.2.3.2 Sampling

RBMs can be stacked to form a Deep Belief Network (DBN), as these models are associated with generative process, we can use visualization to get an insight as to what individual hidden units represent in image space. In order to visualize higher layers, consider a DBN with j layers. In particular, layers $j-1$ and j form an RBM from which we can sample using block Gibbs sampling, which successively samples from $P(\mathbf{h}_{j-1} \mid \mathbf{h}_j)$ and $P(\mathbf{h}_j \mid \mathbf{h}_{j-1})$. Here, the conditionals are same as Equations 4.3 and 4.4 with $\mathbf{v}$ replaced with $\mathbf{h}_{j-1}$. $\mathbf{h}_j$ is binary vector of units from layer j. In order to visualize the hidden node h_{ij}, we clamp this unit to 1 and perform gibbs sampling for layer j and $j-1$. Then we perform ancestral top down sampling in DBN from layer $j-1$ to input layer. This produces a distribution $p_j(\mathbf{v} \mid h_{ij} = 1)$ which is used to characterize h_{ij}. Here, $p_j(\mathbf{v} \mid h_{ij} = 1)$ is the distribution over random variable $\mathbf{v}$ when i_{th} hidden unit in layer j is clamped to 1. This can be done by either by sampling from this distribution or by summarizing the information by computing the expectation $E[\mathbf{v} \mid h_{ij} = 1]$. Second row in Figure 4.2 shows visualization of hidden nodes for each layer using sampling method.

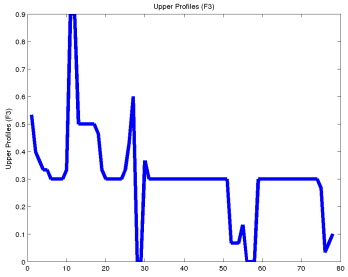


Figure 4.3: Upper Profile.

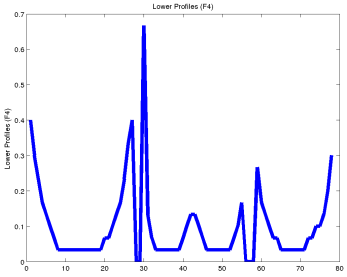


Figure 4.4: Lower Profile.

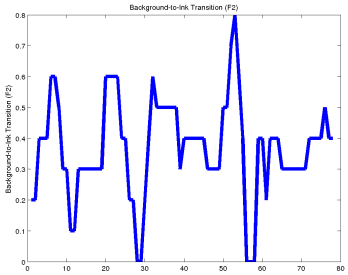


Figure 4.5: Ink Transition Profile.

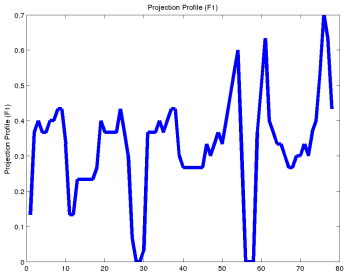


Figure 4.6: Projection Profile.

Figure 4.7: Profiles feature proposed by Rath and Manmatha [40]

We can observe that as we move from layer 1 toward layer 4 the visualization are more abstract and each node has more sense of the structure in the data. We can also observe in Figure 4.2 second row fourth layer, the visualization looks like parts of characters of English. We can see the higher layers has more sense of global structure of data.

4.2.4 Discussions

Given a dataset $\{X_i, Y_i\}_{i=1}^{N}$ where X_i is the input sequence and Y_i is the ground truth output string. We are trying to achieve the mapping $m = pf : X_i \xrightarrow{f} Z_i \xrightarrow{p} Y_i$ by learning intermediate compact representation Z_i in hope that it would give compact and expressive representation to sequence predictor p. The mapping $m = pf$ can be seen as getting the output word prediction by transforming to multiple intermediate representations in hope that the feature presented to the predictor p would be compact, expressive and hierarchical in representation. We can also interpret learning f as learning a non-linear operator $\{f : x \to z \mid x \in \{0,1\}^{d_v}, z \in \mathbb{R}^{d_h}\}$ through a combination of series of non-linear operators $\{f = f_3 f_2 f_1 : x \to z_1 \to z_2 \to z\}$, each f_1, f_2, f_3 can be learned by an RBM which gives f some characteristics like efficient, compact, non-linear and hierarchical representation [6].

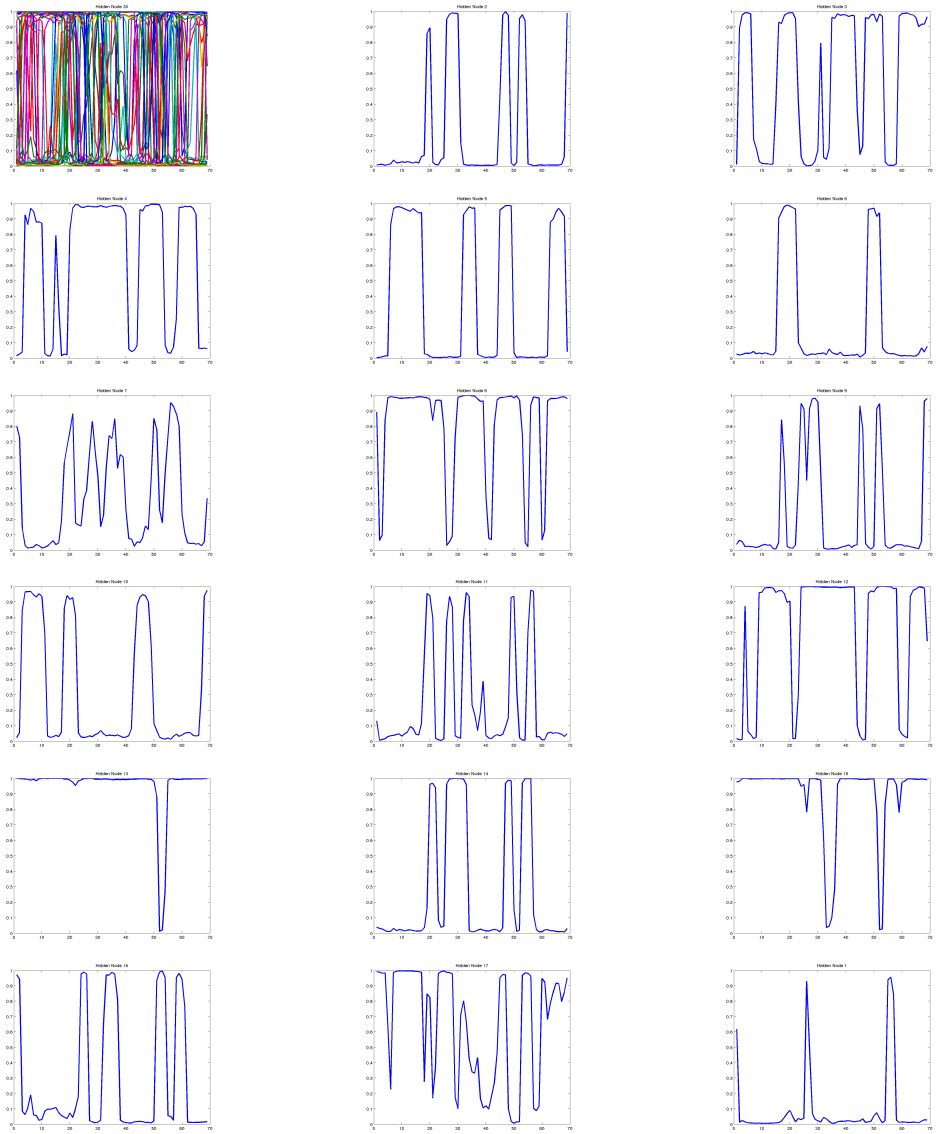


Figure 4.8: Deep Profiles: (Top Left) Image showing coverage of learnt profiles. It can be seen that every part of image is densely covered. (Following Top Left) Following images show few deep projection profiles learnt which is sensitive to specific pattern as shown in figure 4.2 .

Table 4.1: Comparison of RBM and Profile features of OCR Corpus for RBM(160-90) & BLSTM(50-50-50). The numbers indicate number of hidden nodes in each layer.

Languages	LabelError(%) Profile	LabelError(%) RBM	SeqError(%) Profile	SeqError(%) RBM
English	0.84	**0.22**	2.84	**0.65**
Kannada	4.82	**2.82**	18.23	**10.81**
Malayalam	1.38	**0.66**	9.11	**3.78**
Marathi	3.79	**1.43**	13.40	**5.68**
Telugu	5.78	**2.27**	22.67	**10.02**

4.2.5 Interpretation as Deep Profiles

Rath and Manmatha [40] proposed four profile features, namely Upper Profile, Lower Profile, Ink Transition Profile and Projection Profile shown in figure 4.7. Assuming foreground pixels are of value 1, projection profiles can be seen as a 2D convolution operation over the image as shown in equation 4.5.

$$F_{1\times w} = \frac{1}{h} 1_{h\times 1} * I_{h\times w} \tag{4.5}$$

Our proposed features can be seen as a natural extension of projection profiles. Instead of a vector $\frac{1}{h}1_{h\times 1}$, we use $\mathsf{N}_{h\times w_w}$ and convolve with the image $I_{h\times w}$. So instead of four features we could learn many features from data which turns out to have dense coverage as shown in figure **??**. Figure **??** shows overlayed profiles of 60 profiles learnt from data which would have taken long time of design and experimentation cycle (We can see that some RBM profiles has covered every part of word image unlike profile features [40] in figure 4.8 top left image). We have a network formed by stacking RBM and each hidden unit in last layer is sensitive to a pattern as visualized in figure 4.2. Hence, that hidden unit carves out a profile. So each hidden unit in last layer carves out a different profile which has been learn from the data (Few shown in figure 4.8).

4.3 Experiments & Discussion

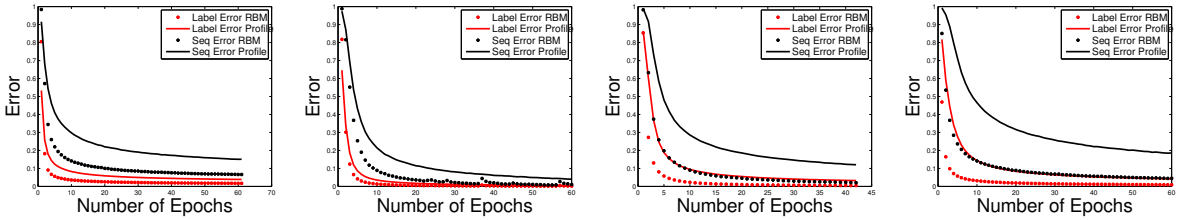
4.3.1 Dataset and Implementation Details

We choose five languages for comparison, namely English Kannada, Malayalam, Marathi and Telugu with 295K, 171K, 65K, 135K and 137K samples respectively. Four Indian languages namely Kannada, Malayalam, Marathi and Telugu are taken from OCR corpus [23] and English is taken from nine books of our internal dataset.

All images are binarized using *otsu* thresholding [37]. We resize all word images to a height $H = w_h = 30$ pixels while maintaining the aspect ratio. We extract sequence of vectors for each word image by running a sliding window of width $w_w = 10$ pixels and step $s = 3$ pixels and unrolling it into a vector. BLSTM classifier has a constraint that number of windows in an image should be greater than ground

Table 4.2: Measure of statistical significance of performance gain using paired t-test.

Languages	Mean Performance Gain(%)	Standard Deviation(%)	t statistics	p value
English	0.75	0.0603	25.05	$1.39e{-}04$
Kannada	2.62	0.4340	12.08	$1.20e{-}03$
Malayalam	0.80	0.1433	11.16	$1.50e{-}03$
Marathi	2.98	0.5289	11.27	$1.50e{-}03$
Telugu	3.28	0.4542	14.45	$7.17e{-}04$



(a) Kannada (b) Malayalam (c) Marathi (d) Telugu

Figure 4.9: Error convergence comparison of RBM (160-90) vs Profile feature for different languages. This figure demonstrates that stacked RBM features makes learning stage of BLSTM predictor converge faster in comparison to profile features. So stacked RBM features show better convergence rates.

truth sequence length and our chosen window width and step size does satisfy this for all languages. Then we perform unsupervised feature learning using a stacked RBM. We use around 200K windows for each language to train stacked RBM. All the vector sequences are encoded with stacked RBM and a BLSTM model is learned on these encoded vector sequences. Learning rate and momentum for RBM was set to 0.1 and 0.9 respectively. For BLSTM, learning rate and momentum was set to 0.0001 and 0.9 respectively. For fair comparison between profile features and RBM features, everything is kept fixed which includes word image height, windows size, step size for sliding window, BLSTM architecture. We tuned the hyper-parameters by measuring performance on validation set. In our experience momentum value of 0.9 or 0.95 works well for many different applications but training is very dependent on learning rate.

4.3.2 Results

The main purpose of presentation is comparison between hand engineered profile features[25] and features learned from data using stacked RBM. We show that features learned by using stacked RBM are compact, expressive and performs better than profile feature[25] for many languages. For all experiments we perform validation by three way data split, 60% for training, 20% for validation and 20% for testing. Figure 4.2 shows visualization of feature maps learned at first four layers. Each feature map is associated with a hidden unit. We visualize the higher layer as linear combination of feature maps of previous layers or by sampling particles in visible space given a specific hidden nodes being activated in

Table 4.3: Additional Comparison Results showing effect of number of layers.

Features	English	Kannada	Malayalam	Marathi	Telugu
Profile[25]	0.84	4.82	1.38	3.79	5.78
RAW	0.65	**2.38**	0.67	**0.70**	1.81
2L-RBM step=3	**0.22**	2.82	0.66	1.43	2.27
3L-RBM step=1	0.25	2.70	**0.61**	0.90	1.74
4L-RBM step=1	-	2.79	-	1.15	**1.70**

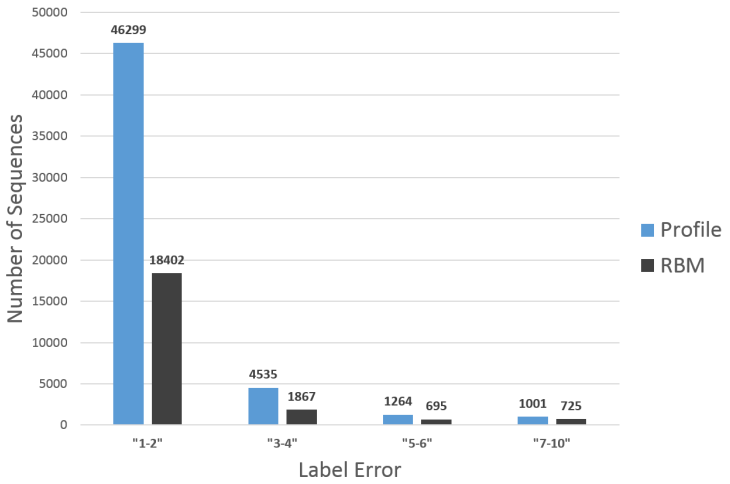


Figure 4.10: Number of Sequences v/s Edit Distance. The plot shows a comparison of number of sequences having edit distance in specific bin. We can see lower errors for RBM.

a DBN. We observe that the learnt feature maps look like region detectors to partial alphabets in English language as we move from first layer towards fourth layer. Also all windows in data can be represented by a cascade of linear combination of feature maps followed by non-linearity.

We performed statistical testing of performance gain using paired t-test. We set our critical value(significance level) $\alpha = 0.05$. For each language we create experiments by randomly sampling 1.0, 0.9, 0.8 and 0.6 of training population ratio and perform paired t-test on these generated populations. Table 4.2 shows mean performance gain, standard deviation, t-statistic and p-value for each language. p-value is less than α for all languages. Hence, stacked RBMs give statistically significant performance gain as compared to profile features.

Table 4.1 shows comparison of profile features[25] and stacked RBM features. We can clearly notice that RBM performs significantly better than profile features[25] in all five languages with respect to label error rate and sequence error rate. Table 4.10 shows the distribution of label errors into four bins. We can see that performance of stacked RBM features is much better as compared to profile features and it makes very few label error even in bin 1-2 with respect to profile features which shows that the learned representations as expressive. The data in bar graph is composed of four languages.

The convergence comparison is done in Figure 4.9 which highlights that with respect to number of epochs stacked RBM features show better convergence rates compared to profile features. Stacked RBM supports the BLSTM network in converging faster in comparison to profile features and can be observed in Figure 4.9.

4.3.3 Discussions

Stacked RBMs are feature learners which disentangle hidden factors of variations. For example, for natural scenes first layer can learn edge detectors, next layer can learn composition of features learnt from previous layers i.e. next layer can learn corner detectors. We can observe in second row of Figure 4.2 that features learnt at fourth layer look like partial characters of English. Also input image can be reconstructed by combination of those feature maps. Therefore, as we visualize higher layers we reach toward more abstract concepts like moving from pixels to small regions. Another advantage of using stacked RBM with bottleneck architecture is that it reduces the dimensionality of input and can speed up computation compare to raw features. We can also see that even if dimensionality of stacked RBM features have increased compared to profile features. This could be compensated to some extent by faster convergence rates and better performance but time taken per epoch is higher than profile features which is not surprising as profile features have much lower dimension. We also observe that with same architecture our proposed method performs better than RAW features for English but further experimentation is required for other languages.

4.4 Summary

We proposed a framework for word prediction for printed text where learned feature representation are obtained by stacked RBM and a sequence predictor BLSTM. The framework is segmentation free and does not depend on hand engineered features. We investigate the expressiveness of the stacked RBM features which were learned in an unsupervised fashion. We demonstrate better results compared to hand engineered profile features which had worked well in similar settings in past. Stacked RBM features were expressive across many languages which has variation in structural and statistical properties of symbols. Our experiments show that stacked RBMs can learn feature representations from data and perform better than profile features in generic settings. It also have better empirical convergence rates. Unsupervised feature learning is applicable to script identification and script specific learning problems.

Chapter 5

Conclusions and Future Directions

In this thesis, we have tried to see optical character recognition as a mapping problem (Equivalently projection of one sequence in image domain into other sequence in text domain). We proposed extensions to two method and reduced its limitations. We proposed an application for sequence to sequence learning architecture which removed two limitations of previous state of art method based of connectionist temporal classification output layer. We also proposed an extension to profile features (Rath and Manmatha) which enabled us to use same idea but by learning features from data.

Sequence to Sequence Learning
We proposed an end-to-end recurrent encoder-decoder based sequence learning approach for printed text Optical Character Recognition (OCR). In contrast to present day existing state-of-art OCR solution which uses CTC output layer [16] our approach makes minimalistic assumptions on the structure and length of the sequence. We use a two step encoder-decoder approach – (a) A recurrent encoder reads a variable length printed text word image and encodes it to a fixed dimensional embedding. (b) This fixed dimensional embedding is subsequently comprehended by decoder structure which converts it into a variable length text output.
Our architecture gives competitive performance relative to Connectionist Temporal Classification (CTC) [16]) output layer while being executed in more natural settings. The learnt deep word image embedding from encoder can be used for printed text based retrieval systems. The expressive *fixed* dimensional embedding for any variable length input expedites the task of retrieval and makes it more *efficient* which is not possible with other recurrent neural network architectures. We empirically investigated the expressiveness and the learnability of long short term memory (LSTMs) in the sequence to sequence learning regime by training our network for prediction tasks in segmentation free printed text OCRs. The utility of the proposed architecture for printed text is demonstrated by quantitative and qualitative evaluation of two tasks – word prediction and retrieval.

Deep Profiles
In this work, we proposed the possibility of learning an appropriate set of features for designing OCR for a specific language. We learned the language specific features from the data with no supervision. In

this work, we learned features using a stacked Restricted Boltzman Machines (RBM) and used it with the RNN based recognition solution. We validated our proposed features on five different languages. In addition, these novel features also resulted in better convergence rate of the RNNs. This feature can be interpreted as deep extension of projection profiles. These features can be used as a plug and play features to get improvements where profile feature (Rath and Manmatha) are used. In fact recent success in training deep learning models favours to take simple ideas and extend it to powerful non-linear model.

Future Directions

- **Attention Based Models**: At a finer level, one could attempt sequence to sequence learning with attention for optical character recognition. These models are capable of highlighting / attending regions in image which is responsible for current character output.
- **Efficient Semantic Representation of Sentences**: One could use a hierarchy of recurrent networks to project sentences to a fixed dimensional representations.
- **Multi-task recurrent networks for Optical Character Recognition**: These models could simultaneously give us language, script and text content.
- **Higher level document retrieval**: Since current recognition system at word and line level have reached a mature stage, it is now possible to build retrieval systems which could actually retrieve documents images which are similar at higher level semantics. Use of paragraph vectors [29] or skip-thought vectors [24] can give us some level representation and help facilitate useful retrieval system.

Related Publication

1. Devendra Kumar Sahu and C. V. Jawahar. ”Unsupervised Feature Learning for Optical Character Recognition.” 13th IAPR International Conference on Document Analysis and Recognition (ICDAR 2015), in Nancy, France. (Oral Presentation).

Bibliography

[1] ABBYY OCR ver. 9. http://www.abbyy.com/.

[2] Christopher Olah Blog. http://colah.github.io/.

[3] Tesseract Open-Source OCR. http://code.google.com/p/tesseract-ocr/.

[4] B. Al-Badr and S. A. Mahmoud. Survey and bibliography of arabic optical text recognition. *Signal processing*, 1995.

[5] Y. Bengio. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2009.

[6] Y. Bengio, A. C. Courville, and P. Vincent. Representation learning: A review and new perspectives. *CoRR*, 2012.

[7] Y. Bengio, I. J. Goodfellow, and A. Courville. Deep learning. Book in preparation for MIT Press, 2015.

[8] 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.

[9] M. A. Carreira-Perpinan and G. E. Hinton. On contrastive divergence learning.

[10] S. Chandra, S. Kumar, and C. V. Jawahar. Learning multiple non-linear sub-spaces using k-rbms. In *CVPR*, 2013.

[11] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, 2014.

[12] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, 2014.

[13] D. Doermann. The indexing and retrieval of document images: A survey. *Computer Vision and Image Understanding*, 1998.

[14] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. Technical report, 2009.

[15] V. Govindan and A. Shivaprasad. Character recognitiona review. *Pattern recognition*, 1990.

[16] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *ICML*, 2006.

[17] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber. A Novel Connectionist System for Unconstrained Handwriting Recognition. *PAMI*, 2009.

[18] Y. N. Hammerla, T. Plotz, S. Vajda, and G. A. Fink. Towards feature learning for hmm-based offline handwriting recognition. In *International Workshop on Frontiers in Arabic Handwriting Recognition*, 2010.

[19] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 2002.

[20] G. E. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Process. Mag.*, 2012.

[21] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 1997.

[22] G. B. Huang, H. Lee, and E. Learned-Miller. Learning hierarchical representations for face verification with convolutional deep belief networks. In *CVPR*, 2012.

[23] C. V. Jawahar and A. Kumar. Content-level Annotation of Large Collection of Printed Document Images. In *ICDAR*, 2007.

[24] R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler. Skip-thought vectors. *CoRR*, abs/1506.06726, 2015.

[25] P. Krishnan, N. Sankaran, A. K. Singh, and C. V. Jawahar. Towards a robust ocr system for indic scripts. In *DAS, 2014*, 2014.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*. 2012.

[28] A. Kumar, C. V. Jawahar, and R. Manmatha. Efficient search in document image collections. In *ACCV*, 2007.

[29] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[30] J. Liang, D. Doermann, and H. Li. Camera-based analysis of text and documents: a survey. *International Journal of Document Analysis and Recognition (IJDAR)*, 2005.

[31] L. M. Lorigo and V. Govindaraju. Offline arabic handwriting recognition: a survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2006.

[32] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.

[33] S. Mao, A. Rosenfeld, and T. Kanungo. Document structure analysis algorithms: a literature survey. In *Electronic Imaging 2003*, 2003.

[34] M. Mathew, A. K. Singh, and C. Jawahar. Multilingual ocr for indic scripts. In *DAS*, 2016.

[35] G. Nagy. Twenty years of document image analysis in pami. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 2000.

[36] L. O'Gorman. The document spectrum for page layout analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 1993.

[37] N. Otsu. A threshold selection method from gray-level histograms. *Automatica*, 1979.

[38] U. Pal and B. Chaudhuri. Indian script character recognition: a survey. *Pattern Recognition*, 2004.

[39] R. Plamondon and S. N. Srihari. Online and off-line handwriting recognition: a comprehensive survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2000.

[40] T. M. Rath and R. Manmatha. Word image matching using dynamic time warping. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, 2003.

[41] N. Sankaran, A. Neelappa, and C. Jawahar. Devanagari text recognition: A transcription based formulation. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, 2013.

[42] J. Sauvola and M. Pietikäinen. Adaptive document image binarization. *Pattern recognition*, 2000.

[43] R. Smith. An overview of the tesseract ocr engine. In *ICDAR*, 2007.

[44] B. Su and S. Lu. Accurate scene text recognition based on recurrent neural networks. In *ACCV*, 2014.

[45] T.-H. Su, T.-W. Zhang, H.-J. Huang, and Y. Zhou. Hmm-based recognizer with segmentation-free strategy for unconstrained chinese handwritten text. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, 2007.

[46] I. Sutskever, J. Martens, and G. Hinton. Generating text with recurrent neural networks. In *ICML*, 2011.

[47] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *CoRR*, 2014.

[48] G. W. Taylor, G. E. Hinton, and S. Roweis. Modeling human motion using binary latent variables. In *NIPS*, 2006.

[49] T. Tieleman. Training Restricted Boltzmann Machines using Approximations to the Likelihood Gradient. In *ICML*, 2008.

[50] T. Tieleman and G. Hinton. Lecture 6.5 - rmsprop, coursera: Neural networks for machine learning. *Technical report*, 2012.

[51] L. van der Maaten and G. Hinton. Visualizing high-dimensional data using t-sne. *JMLR*, 2008.

[52] O. Vinyals and Q. V. Le. A neural conversational model. *CoRR*, 2015.

[53] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. *CoRR*, 2014.

[54] W. Zaremba and I. Sutskever. Learning to execute. *CoRR*, 2014.