Tribhuvan University

Institute of Science and Technology

Text Recognition in Image Using Deep Convolutional Neural Network

A Dissertation Report

Submitted to

Central Department of Computer Science and Information Technology

Tribhuvan University, Kirtipur

Kathmandu, Nepal

By

Roshan Tandukar

Under the supervision of

Mr. Dhiraj Kedar Pandey

Asst. Prof, CDCSIT, TU

July, 2017

Tribhuvan University
Institute of Science and Technology
Central Department of Computer Science and Information Technology

Student's Declaration

I hereby declare that I am the only author of this work and that no sources other than the listed here have been used in this work.

……

Roshan Tandukar
Date: 26 July, 2017

Supervisor's Recommendation

I hereby declare that the dissertation prepared under my supervision by Mr. Roshan Tandukar entitled "**Text Recognition in Image Using Deep Convolutional Neural Network**" in the partial fulfillment of the requirements for the degree of M.Sc. CSIT in Central Department of Computer Science and Information Technology be processed for the final evaluation.

……

Asst. Prof. Dhiraj Kedar Pandey
Central Department of Computer Science & Information Technology,
IOST,
TU, Kirtipur.
Date: 26 July, 2017

Tribhuvan University
Institute of Science and Technology
Central Department of Computer Science and Information Technology

Letter of Approval

We certify that we have read this dissertation and in our opinion it is satisfactory in the scope and quality as a dissertation in the partial fulfillment for the requirement of Masters Degree in Computer Science and Information Technology.

Evaluation Committee

……..
Asst. Prof. Nawaraj Paudel,
Central Department of Computer Science & Information Technology,
Tribhuvan University, Kirtipur, Nepal.
(Dept. Head)

……..
Asst. Prof Dhiraj Kedar Pandey
Central Department of Computer Science & Information Technology,
TU, Kirtipur, Nepal
(Supervisor)

……
Mr. Dipak Pudasaini
(External Examiner)

……
Mrs. Lalita Sthapit
(Internal Examiner)

Date: 26 July, 2017

Acknowledgements

I would like to thank all the people who contributed in some way to the work described in this dissertation. I would like to express my deepest gratitude to my advisor, **Mr. Dhiraj Kedar Pandey**, Lecturer of Central Department of Computer Science and Information Technology, Tribhuvan University for his excellent guidance, patience, insightful discussions, trust and correction of my thesis work so that my dissertation was possible.

Additionally, I would like to thank Asst. Prof. **Mr. Nawaraj Paudel**, Head of Central Department of Computer Science and Information Technology, Tribhuvan University for his encouragement and guidance.

I am extremely grateful to **University of Chinese Academy of Sciences** (UCAS) hosted by Institute of Tibetan Plateau Research Chinese Academy of Sciences (ITP-CAS) and coordinated by Kathmandu Centre for Research Education, CAS-TU for providing me thesis grant which helped a lot to complete my research work.

My special thanks to **Dr. Shashidhar Ram Joshi** and **Dr. Subarna Shakya** for inspiration, thoughtful guidance and critical comments.

I want to thank **Mr. Max Jaderberg**, a senior research scientist at Google DeepMind in machine learning and computer vision for his Matlab source code and model in the Deep Convolutional Neural Network, which I got from the Github.

I am also grateful to all the lecturers **Mr. Jagadish Bhatt**, **Mr. Arjun Singh Saud**, **Ms. Lalita Stapit** and **Mr. Bikash Balami** for the support till the date. I am also thankful to all the staff memebers of Department of Computer Science, TU for their cooperation and help.

Thanks to all my friends for their support and finally I want to thank my family for their love, continuous support and encouragement.

Abstract

Text recognition in an image is one of the challenging tasks in computer vision and pattern recognition which involves automatically reading the text from the images. The non-uniformity in text styles, font size and colors, the complex background and the orientation makes it different from Optical character recognition (OCR). In this research work, text recognition with deep convolutional neural network architecture is described. Convolutional neural network is based on deep learning. The multi-layer architecture of the deep convolutional neural network allows using the deep features with less image preprocessing tasks and sharing of weights making it a faster neural network model. The deep architecture of the convolutional neural network is investigated with its two models character sequence model and dictionary encoding model to recognize the scene text. The strength and weakness of the models are analyzed based on the experiments done with the two publicly available datasets which includes Synth90k dataset and ICDAR 2003 datasets and obtained the accuracy of 93.88% in Synth90k and 70.33% in ICDAR 2003 dataset with the dictionary encoding model.

Keywords: *Text recognition, computer vision, Pattern recognition, Optical character recognition (OCR), Deep convolutional neural network, Character sequence model, Dictionary encoding model*

Table of Contents

List of Figure

List of Abbreviation

ANN Artificial Neural Network

API Application Programming Interface

ASCII American Standard Code for Information Interchange

BP Backpropagation

CNN Convolutional Neural Network

CNTK Microsoft Cognitive Toolkit

CPU Central Processing Unit

CRF Conditional Random Field

DL Deep Learning

GPU Graphics Processing Unit

HOG Histogram of Oriented Gradients

ICDAR International Conference on Document Analysis and Recognition

IDL Interactive Data Language

MSE Mean Square Error

MSER Maximally Stable External Regions

NMS Non-maximal Suppression

OCR Optical Character Recognition

SGD Stochastic Gradient Descent

SWT Stationary Wavelet Transform

Chapter 1

Introduction

1.1 Introduction

Text recognition in images is a research area which attempts to develop a system able to automatically read the text from the images. It refers to the visual decoding of a localized, cropped instance of a text (a word image) into the string of characters depicted. Texts in an image carry high-level semantic information about a scene, which can be used to assist a wide variety of applications, such as image understanding, image search and indexing, navigation, and human computer interaction. With the increasingly popular use of information in images and video forms, detecting texts of random orientations from images taken by different devices under less controlled conditions has become an increasingly important and yet equally challenging task. Recognizing text in scene images is much more challenging due to the many possible variations in backgrounds, textures, fonts, and lighting conditions that are present in the images.

A Text recognition system receives an input in the form of image which contains some text information and the output of this system is in electronic format which includes the text information in image stored in computer readable form. Text detection along with the recognition in natural scene images plays an important role in content analysis of images.

1.1.1 Artificial Neural Network and Deep Learning

An artificial neuron network (ANN) is a non-linear, parallel, distributed and highly connected computational model based on the structure and functions of biological neural networks with a high capability of adaptivity, self-organization and fault tolerance [1]. Information that flows through the network affects the structure of the ANN as a neural network changes or learns based on inputs and corresponding outputs. ANNs are considered nonlinear statistical data modeling tools where the

complex relationships between inputs and outputs are modeled or patterns are found. These are composed of interconnecting artificial neurons where each neuron is a programming construct that mimics the properties of biological neurons. Artificial neural networks are responsible for many of the recent advances in artificial intelligence, including voice recognition, image recognition, and robotics.

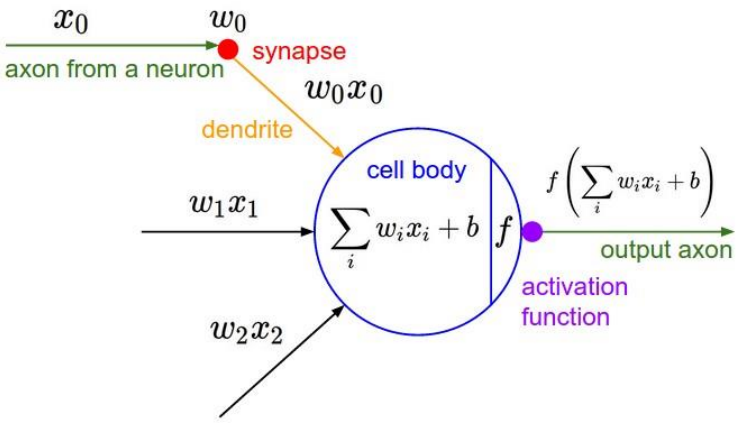

Fig 1.1: An Artificial Neuron Model

Deep learning [2] can be seen as a form of hierarchical learning, where algorithms make use of multiple layers of representations to gradually transform data into high level concepts. A deep architecture is one which is composed of more than one layer of transformations from input to output. Moving from the input, through each layer of transformations, higher level features are derived from the lower level features, leading to a hierarchical representation. Such a design choice systems built from multiple levels of representation is highly motivated from nature. It has been well observed that the mammalian visual cortex exhibits a layered structure, with layers comprising of groups of neuron cells in the brain.

It allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics.

Deep Learning uses layers of algorithms to process data, understand human speech, and visually recognize objects. Information is passed through each layer, with the output of the previous layer providing input for the next layer. The first layer in a

network is called the input layer, while the last is called an output layer. All the layers in between the two are referred to as hidden layers. Each layer is typically a simple, uniform algorithm containing certain kind of activation function.

Feature extraction is another aspect of Deep Learning. Feature extraction uses an algorithm to automatically construct meaningful "features" of the data for purposes of training, learning, and understanding. Currently, the processing of Big Data and the evolution of Artificial Intelligence are both dependent on Deep Learning.

There are multiple representations of deep neural architectures using different models such as Deep convolutional Neural Network, Recurrent Neural Network, Deep Belief Network(DBN), Restricted Boltzmann machine(RBM) and Autoencoders [3]. Deep convolutional nets are mostly used in processing images, video, speech and audio, whereas recurrent nets are used on sequential data such as text and speech.

1.1.2 Convolutional Neural Network

A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers, with a subsampling step, and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the 2-dimensional structure of an input image (or other 2D input such as a speech signal) [4]. This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units.

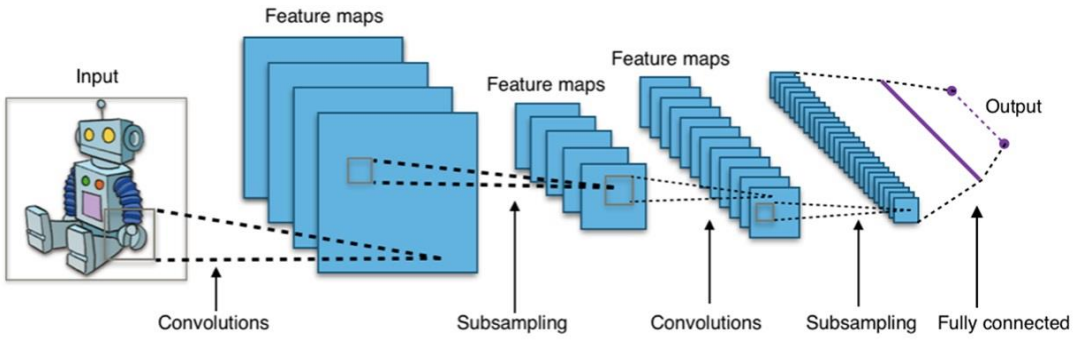

Fig 1.2: Convolutional Neural Network with different layers [3]

The most commonly associated idea with convolutional neural networks is the idea of a “moving filter” which passes through the image. This moving filter, or convolution, applies to a certain neighborhood of nodes. Sparse connections and constant filter parameters or weights make CNN more efficient. As the filter moves around the image the same weights are being applied. Each filter therefore performs a certain transformation across the whole image. This is in contrast to fully connected neural networks, which have a different weight value for every connection.

The detailed structure of CNN is described in Section 3.4.2. In the proposed model, the architecture of a CNN and the back propagation algorithm to compute the gradient with respect to the parameters of the model are used with gradient based optimization.

1.2 Motivation

At a high level, text recognition means being able to automatically read the text in images. No constraints should be imposed on the type of text, or the type of scene that should be able to be read. If a human can read it, a text recognition system should also aim to be able to read it.

Optical character recognition (OCR) systems are designed to convert printed text images to readable text codes such as ASCII, commonly used in data processing and give quite good performance in it, but perform poorly when text is embedded into complex background. In contrast to more classical OCR problems, where the characters are typically monotone on fixed backgrounds, character recognition in scene images is potentially far more complicated due to the many possible variations in background, lighting, texture and font. As a result, building complete systems for these scenarios requires inventing representations that account for all of these types of variations. Indeed, significant effort has gone into creating such systems, with top performers integrating dozens of cleverly combined features and processing stages.

1.3 Challenges

Extracting any sort of semantic information from the raw volume of pixel values of a digital image is not a trivial task which requires multiple levels of feature extraction and inference. Although scene text recognition has far more scopes and applications,

it is still a challenging task and has to cope with different challenges which are as follows:

Diversity of scene text: In contrast to characters in document images, which are usually with regular font, single color, consistent size and uniform arrangement, texts in natural scenes may bear entirely different fonts and non-standard font styles , colors, scales and orientations [5], even in the same scene.

Complexity of background: The backgrounds in natural scene images and videos can be very complex. Elements like signs, fences, bricks and grasses are virtually undistinguishable from true text, and thus these causes confusions and errors in the text recognition.

Interference factors: Various interference factors, for instance, noise, blurriness, distortion, low resolution, non-uniform illumination and partial occlusion, may give rise to failures in scene text detection and recognition.

Lack of language context: Scene texts often appear as a word or group of few words. So, applying larger contexts, such as, at the level of sentence or paragraph which means using any prediction based on language constructs is not possible.

Fig 1.3: Challenges for a Text recognition system: Sample images from ICDAR 2003 datasets

1.4 Problem Definition

The task of a scene text recognition system is to automatically recognize the text in natural images such that the obtained text can be manipulated by a word-processing program in the form of alphabets, words, and numerals. The recognition task is carried out by the deep convolutional neural network. Deep neural network uses a

simple way to do edge detection using convolution. Along with detecting the edges, it also removes a lot of redundant information which are residing in the image. There are various models proposed so far in different papers with different accuracy level. In this research work, two models namely character sequence encoding model and dictionary based constrained models, are taken for the analysis purpose that uses same trained model of CNN with different approaches for text recognition.

1.5 Objectives

The objective of this research work is to investigate the deep convolutional neural network for text recognition with the comparison of two models Comparative performance matrices are analyzed for them. The main objectives are given below:

- To investigate the use of deep features of the text through the use of deep convolutional neural network
- To analyze the character sequence encoding and dictionary encoding constrained models for text recognition

1.6 Outline of Thesis

The remaining part of this document is organized as follows:

Chapter 2 includes an overview of state-of-the-art methods exists for text recognition. It includes the methods and techniques used in the area of text recognition till now.

Chapter 3 includes the research methodologies used in this research work. All the image preprocessing, neural network model with different layers and recognition models are described in detail.

Chapter 4 describes the experimentation and results of the recognition system. Firstly, the implementation details of the system are included in Sections 4.1 and 4.2. It includes description of the platform in which system is implemented and its realization with the datasets in Section 4.3. Performance and efficiency evaluation of the proposed recognition system with two publicly available datasets is described in Section 4.4 with the discussion and analysis parts in Section 4.5.

Chapter 5 contains the conclusions of this research work and the directions for the future works.

Chapter 2

Literature Review

In recent years, text detection and recognition in natural images have become active research topics in the communities of computer vision, pattern recognition and even document analysis. Researchers from these communities have proposed a large amount of novel ideas and approaches for the extraction of textual information from natural images and videos. These methods can be broadly divided into three categories namely, text detection, text recognition and end-to-end text recognition [6]. The first category of methods concern how to discover and locate the regions possibly containing text from natural images, but do not need to perform recognition. The second category of methods supposes that texts have been detected and only focus on the process of converting the detected text regions into computer readable and editable symbols. The third category of methods aims at constructing end-to-end text recognition systems that accomplish both the detection and recognition tasks [7, 8]. This research works is to cover second category of method that takes localized cropped scene text images as inputs.

There are a variety of methods that attempt to construct a unified framework for both text detection and recognition. In the work of Wang et al.[9], an end-to-end text recognition system, inspired by general object detection algorithms in computer vision, treats words as a special kind of object, and characters as parts of the object. Experiments show that this method obtains excellent performance on multiple standard datasets. However, this algorithm can only handle words that are within the given word list, thus it is not applicable to images without a word list.

The first real end-to-end text recognition system for natural images is proposed by Neumann et al. [10], which does not require a word list. This system extracts the character candidates using MSER and eliminates non-text candidates through a trained classifier. The remaining candidates are fed into a character recognition module, which is trained using a large amount of synthetic characters. Neumann et al.

introduced new feature extraction methods and combination strategies, which significantly improves the accuracy and efficiency of this system. Later, Neumann et al. [11] further extended the methods to attain real-time text detection and recognition. He presented a new system for scene text localization and recognition, which combines the advantages of sliding-window based and component based methods. In this system, character parts (strokes) are modeled by oriented bar filters. These oriented bar filters are utilized to perform both character detection and recognition.

Based on the model proposed by Yao et al. [12], an end-to-end system can accomplish scene text detection and recognition concurrently. This is the first work that can localize and read texts of arbitrary orientations in natural images. Given a natural image, candidates (characters and lines) are firstly generated using SWT and clustering which will be processed for the classification.

Mishra et al. [13] employed bottom-up and top-down cues for scene text recognition, which works in an error correction manner. Due to the presence of complex backgrounds in natural scenes, it is very difficult to directly segment characters from local background. So this method uses sliding window to detect possible characters, and treat the detection results as the bottom-up information. The top-down information comes from the statistics of a large dictionary. The bottom-up and top down information is integrated in a unified model through Conditional Random Field (CRF). One of the advantages of this method is that is can tolerate errors in character detection.

Novikova et al. [14] proposed to characterize character appearance and the relationship between characters through a unified probabilistic model. The character candidates are extracted using MSER. This method adopts Weighted Finite-State Transducers [15] as the probabilistic model and searches the most likely word by an efficient reasoning algorithm. However, the procedure of this method is complicated and its word recognition performance has no obvious advantage over other error correction methods that also utilize statistic language models.

The great success of deep learning methods in various computer vision tasks has enlightened researchers in the area of scene text detection and recognition. Wang et al. [16] used CNN for text detection and character recognition. Wang et al. adapted a multi-layer neural network for end-to-end scene text localization and recognition. The

full end-to-end system combines a lexicon with detection/recognition modules using post-processing techniques including NMS and beam search. It works on the assumption on that a lexicon (a list of tens to hundreds of candidate words) are given for a particular image. The method also uses a graphical model, with unary terms from character classifiers acting on HOG features [17] and pairwise terms incorporating spatial layout and scale similarity between neighboring characters. This is done in the context of a lexicon, so each lexicon word is scored for each pictorial structure, with the position of the characters optimized efficiently with dynamic programming. The highest scoring lexicon word is then taken as the final recognition result.

2.1. Text Recognition Methods

Different methods based on character or on word for text recognition from a scene images are reviewed in this section. Since the input to the text recognition stage is generally a cropped word image, all the references to images in this section refer to word images. For the scene text recognition, methods can be split into two groups character based recognition and whole word based recognition.

2.1.1. Character Based Recognition

Character based recognition [9, 10, 18, 19, 20, 21] relies on an individual character classifier to perform per character recognition which is later integrated across the word image to generate the full word recognition. It treats characters as the atomic building blocks for text, modeling them explicitly, and constructing text lines and words by grouping character detections together. This therefore requires some level of character detection or segmentation within the word image.

2.1.2. Word Based Recognition

As an alternative approach to text recognition method mentioned above, other methods use whole word based recognition methods [7, 22, 23, 24, 25]. Rather than requiring character detection and classification, features from across the entire word images are integrated or pooled to perform word classification.

Such type of algorithms assume that characters occur in clusters (words, text lines, paragraphs) and include heuristics and grouping mechanisms to discover these clusters which are subsequently recognized. Indeed, a single, lone character occurring in an image can be a very ambiguous object, and it is often impossible to distinguish between characters like objects that are part of text from those that are just by-products of the rest of the world, when isolated from their context. For example the visual shapes of the characters “I”, “x” and “o” occur frequently in natural images without any connection to text. Only in the context of other spatially-near, character-like objects may create the shapes which may be distinguished as characters and not background structures.

Chapter 3

Research Methodology

This chapter describes the concept behind the methods used in this research. For the text recognition system, logically it can be viewed with four different sub-systems which include image acquisition, preprocessing, feature extraction and then recognition. In the architectural view, the feature extraction and the recognition task will be both implemented by CNN.

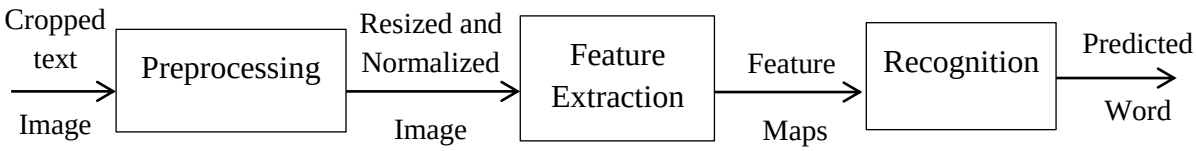

Fig 3.1: Sub-systems in text recognition system

3.1 Image Acquisition

For the recognition system, the cropped input text images are acquired from the publicly available datasets which contains scene texts. The details of the datasets are included in the Section 4.3. Each image is rescaled in the preprocessing stage for further processing.

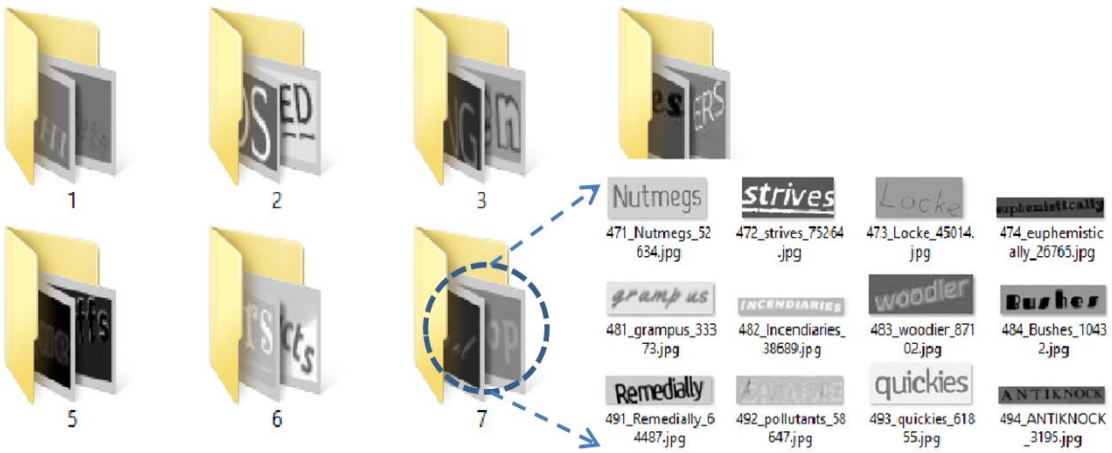

Fig. 3.2: Image Acquisition

3.2 Image Preprocessing

Image Preprocessing is the preliminary step of the recognition process which describes how the images are normalized before extracting features from it.

Algorithm:

1. Read the cropped input image
2. Convert RGB images to gray scale image
3. Normalize the image (feature scaling) using sample mean and standard deviation

The raw images which are input to the system are preprocessed by subjecting to the preliminary operations such that it can be fed to the CNNs for further processing. The input cropped images are converted to grayscale images and resized to 32 × 100 without aspect ratio preservation.

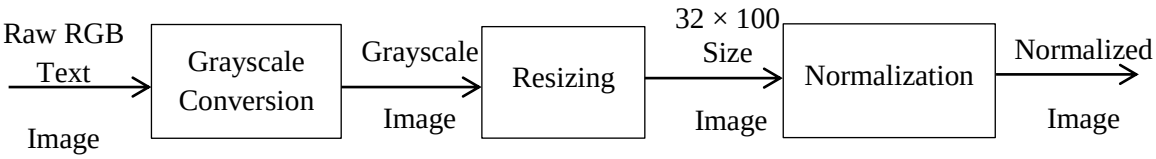

Fig 3.3: Image preprocessing stages

After resizing, to each image the sample mean subtraction and standard deviation normalization is performed.

3.2.1. RGB to grayscale conversion

The RGB image with 24-bit true color, is converted into 8-bit grayscale image using luminosity method. It is computed by taking weighted summation of R, G and B components of RGB image where the weights are selected as standard values (0.2989, 0.5870, 0.1140) for R, G and B components respectively. For the RGB image f(x, y) the corresponding grayscale image g(x, y) is obtained as,

$$g(x,y) = 0.2989 \times R\big(f(x,y)\big) + 0.5870 \times G\big(f(x,y)\big) + 0.1140 \times B\big(f(x,y)\big) \quad (3.1)$$

where R(f(x, y)), G(f(x, y)) and B(f(x, y)) are the red, green and blue components of the RGB image f(x, y).

3.2.2. Feature Scaling

Feature scaling is a method used to standardize the range of independent variables or features of data. It makes the values of each feature in the data to have zero-mean and unit-variance and is computed as,

$$g'(x,y) = \frac{g(x,y) - \text{mean}}{\text{s.d.}} \quad (3.2)$$

where mean is average value and s.d. is the standard deviation.

3.3 Feature Extraction and Classification

Feature extraction is an attribute reduction process. Unlike feature selection, which ranks the existing attributes according to their predictive significance, feature extraction actually transforms the attributes. The transformed attributes, or features, are linear combinations of the original attributes. The feature extraction process results in a much smaller and richer set of attributes. In this research work, the model of the deep convolutional neural network architecture is used for both feature extraction and classification of the text.

3.4 Convolutional Neural Networks

Although neural networks can be applied to computer vision tasks, to get good generalization performance, it is beneficial to incorporate prior knowledge into the network architecture. Convolutional neural networks aim to use spatial information between the pixels of an image and are based on discrete convolution. The basic components of convolutional neural networks are as follows [26, 27, 28]:

3.4.1 Convolution

Assume a grayscale image to be defined by a function

$$I: \{1, \dots, n1\} \times \{1, \dots, n2\} \to W \subseteq R, (i, j) \to I_{i,j}$$

such that the image I can be represented by an array of size n1 × n2. Given the filter $K \in R^{2h_1+1 \times 2h_{2+1}}$ the discrete convolution of the image I with filter K is given by

$$(I \times K)_{r,s} := \sum_{u=-h1}^{h1} \sum_{v=-h2}^{h2} K_{u,v} I_{r+u,s+v} \tag{3.3}$$

where the filter K is given by

$$k = \begin{pmatrix} k_{-h1,-h2} & \cdots & k_{-h1,h2} \\ \vdots & k_{0,0} & \vdots \\ k_{h1,-h2} & \cdots & k_{h1,h2} \end{pmatrix} \tag{3.4}$$

A commonly used filter for smoothing is the discrete Gaussian filter which is defined by

$$(K_{G(\sigma)})_{r,s} = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{r^2 + s^2}{2\sigma^2}\right) \tag{3.5}$$

where s is the standard deviation of the Gaussian distribution.

3.4.2 Layers of CNN

Different types of layers are used in convolutional neural networks. Based on these layers, the architecture is used for classification and can be built by stacking multiple layers [26].

3.4.2.1 Convolutional Layer

Let layer l be a convolutional layer. Then, the input of layer l comprises $m_1^{(l-1)}$ feature maps from the previous layer, each of size $m_2^{(l-1)} \times m_3^{(l-1)}$. In the case where l = 1, the input is a single image I consisting of one or more channels. This way, a convolutional neural network directly accepts raw images as input. The output of layer l consists of $m_1^{(l)}$ feature maps of size $m_2^{(l)} \times m_3^{(l)}$. The i^{th} feature map in layer l, denoted $Y_i^{(l)}$, is computed as

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K_{i,j}^{(l)} \times Y_j^{(l-1)} \tag{3.6}$$

where $B_i^{(l)}$ is a bias matrix and $K_{i,j}^{(l)}$ is the filter of size $2h_1 + 1 \times 2h_{2+1} + 1$ connecting the j^{th} feature map in layer (l -1) with the i^{th} feature map in layer l. As mentioned above, $m_2^{(l)}$ and $m_3^{(l)}$are influenced by border effects. When applying the discrete convolution only in the so called valid region of the input feature maps, that is only for pixels where the sum of equation is defined properly, the output feature maps have size

$$m_2^{(l)} = m_2^{(l-1)} - 2h_1^{(l)} \tag{3.7}$$

$$\text{and } m_3^{(l)} = m_3^{(l-1)} - 2h_2^{(l)} \tag{3.8}$$

Often the filters used for computing a fixed feature map $Y_i^{(l)}$ are the same, that is $K_{i,j}^{(l)} = K_{i,k}^{(l)}$ for $j \neq k$. In addition, the sum in equation (3.6) may also run over a subset of the input feature maps. To relate the convolutional layer and its operation as defined by equation 3.6 to the multilayer perceptron, the above equation can be also written as below.

Each feature map $Y_i^{(l)}$ in layer l consists of $m_2^{(l)} \cdot m_3^{(l)}$units arranged in a two-dimensional array. The unit at position (r, s) computes the output

$$\left(Y_i^{(l)}\right)_{r,s} = \left(B_i^{(l)}\right)_{r,s} + \sum_{j=1}^{m_1^{(l-1)}} \left(K_{i,j}^{(l)} \times Y_j^{(l-1)}\right)_{r,s} \tag{3.9}$$

$$\left(Y_i^{(l)}\right)_{r,s} = \left(B_i^{(l)}\right)_{r,s} + \sum_{j=1}^{m_1^{(l-1)}} \sum_{u=-h_1^{(l)}}^{h_1^{(l)}} \sum_{v=-h_2^{(l)}}^{h_2^{(l)}} \left(K_{i,j}^{(l)}\right)_{u,v} \left(Y_j^{(l-1)}\right)_{r+u,s+v} \quad (3.10)$$

The trainable weights of the network can be found in the filters $K_{i,j}^{(l)}$ and the bias matrices $B_i^{(l)}$.

It is known that subsampling is used to decrease the effect of noise and distortions. Subsampling can be done using so called skipping factors $s_1^{(l)}$and $s_2^{(l)}$. The basic idea is to skip a fixed number of pixels, both in horizontal and in vertical direction, before applying the filter again. With skipping factors as above, the size of the output feature maps is given by

$$m_2^{(l)} = \frac{m_2^{(l-1)} - 2h_1^{(l)}}{s_1^{(l)} + 1} \quad (3.11)$$

$$\text{and } m_3^{(l)} = \frac{m_3^{(l-1)} - 2h_2^{(l)}}{s_2^{(l)} + 1} \quad (3.12)$$

3.4.2.2 Non-Linearity Layer

If layer l is a non-linearity layer, its input is given by $m_1^{(l)}$feature maps and its output comprises again $m_1^{(l)} = m_1^{(l-1)}$ feature maps, each of size $m_2^{(l-1)} \times m_3^{(l-1)}$such that $m_2^{(l)} = m_3^{(l-1)}$ and $m_3^{(l)} = m_3^{(l-1)}$ given by,

$$Y_i^{(l)} = f\left(Y_i^{(l-1)}\right) \quad (3.13)$$

where f is the activation function used in layer l and operates point wise. An additional gain coefficient can be added as:

$$Y_i^{(l)} = g_i f\left(Y_i^{(l)}\right) \quad (3.14)$$

Rectification

Let layer l be a rectification layer. Then its input comprises $m_1^{(l-1)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$ and the absolute value for each component of the feature maps is computed:

$$Y_i^{(l)} = \left| Y_i^{(l)} \right| \tag{3.15}$$

where the absolute value is computed point wise such that the output consists of $m_1^{(l)} = m_1^{(l-1)}$ feature maps unchanged in size. It has been shown that rectification plays a central role in achieving good performance. Although rectification could be included in the non-linearity layer, it can also be implemented as an independent layer.

3.4.2.3 Feature Pooling and Subsampling Layer

The motivation of subsampling the feature maps obtained by previous layers is robustness to noise and distortions. Reducing the resolution can be accomplished in different ways [28].

Let l be a pooling layer. Its output comprises $m_1^{(l)} = m_1^{(l-1)}$ feature maps of reduced size. In general, pooling operates by placing windows at non-overlapping positions in each feature map and keeping one value per window such that the feature maps are subsampled. Two types of pooling are included here:

Average pooling: An average pooling layer performs down-sampling by dividing the input into rectangular pooling regions and computing the average of each region.

Max pooling: For max pooling, the maximum value of each window is taken. It is used to get faster convergence during training.

Both average and max pooling can also be applied using overlapping windows of size 2p× 2p which are placed q units apart. Then the windows overlap if q < p. This is found to reduce the chance of overfitting the training set.

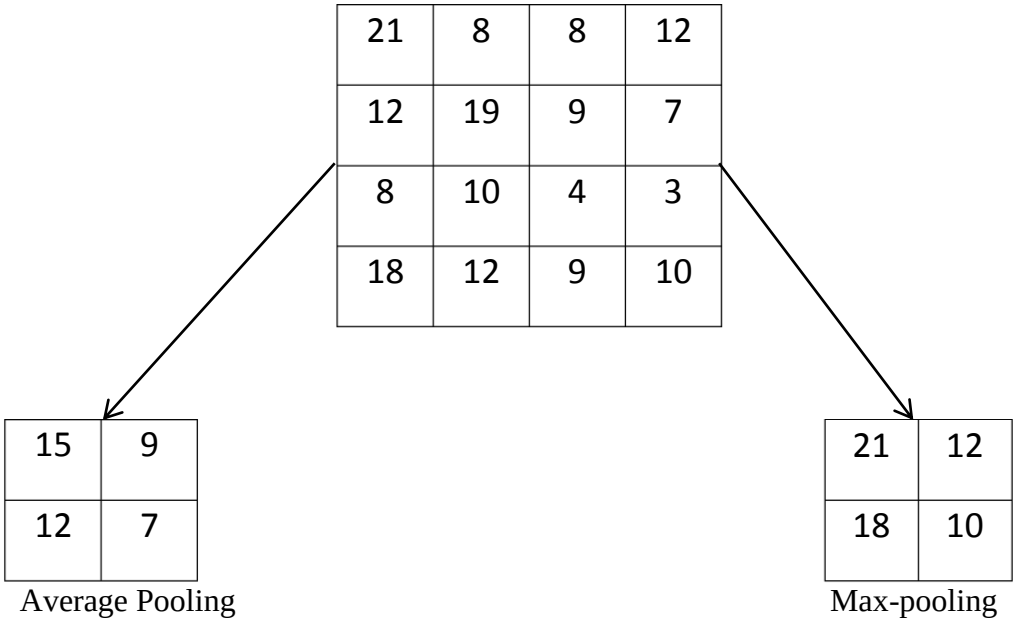

Fig. 3.4: Average pooling and Max-pooling

3.4.2.4 Fully Connected Layer

Let layer l be a fully connected layer. If layer (l - 1) is a fully connected layer, the following equation can be applied.

$$z^{(l)} = w^{(l)} y^{(l-1)} \tag{3.16}$$

where $z^{(l)}$, $w^{(l)}$ and $y^{(l-1)}$ denote the corresponding vector and matrix representations of the actual inputs $z_i^{(l)}$, the weights $w_{i,k}^{(l)}$ and the outputs $y_k^{(l-1)}$, respectively. Otherwise, layer l expects $m_1^{(l-1)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$ as input and the i_{th} unit in layer l computes:

$$y_i^{(l)} = f\left(z_i^{(l)}\right) \tag{3.17}$$

$$\text{with } z_i^{(l)} = \sum_{j=1}^{m_1^{(l-1)}} \sum_{r=1}^{m_2^{(l-1)}} \sum_{s=1}^{m_3^{(l-1)}} w_{i,j,r,s}^{(l)} \left(Y_j^{(l-1)}\right)_{r,\ s} \tag{3.18}$$

where $w_{i,j,r,s}^{(l)}$ denotes the weight connecting the unit at position (r; s) in the j^{th} feature map of layer (l - 1) and the i^{th} unit in layer l. In practice, convolutional layers are used to learn a feature hierarchy and one or more fully connected layers are used for

classification purposes based on the computed features. It should be noted that a fully-connected layer already includes the non-linearities while for a convolutional layer the non-linearities are separated in their own layer.

3.5 Base Architecture

A typical CNN architecture involves the above four layers with different size of feature maps as shown below. Each layer generates the feature map which is the input for the next layer [8].

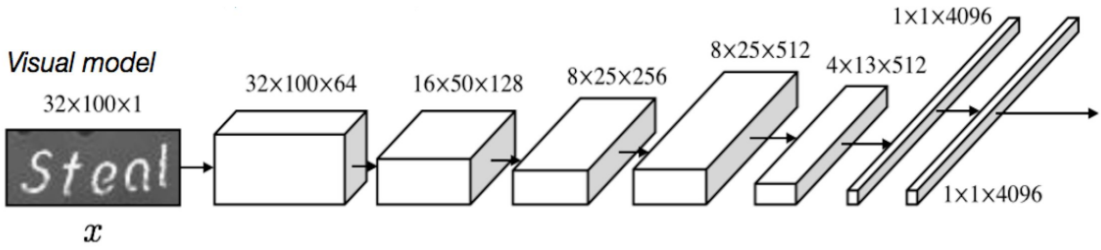

Fig. 3.5: Base Architecture of CNN [8]

It consists of eight weight layers which include five convolutional layers and three fully connected layers. The convolutional layers have the following configuration of the filers:

{filter_size, number of filters}: {5, 64}, {5,128}, {3,256}, {3,512}, {3,512}.

Each convolutional layer detects local conjunctions of features from the previous layer and mapping their appearance to a feature map. As a result of convolution, the image is split into perceptrons creating local receptive fields and finally compressing the perceptrons in feature maps. Thus, this feature map stores the information where the feature occurs in the image and how well it corresponds to the filter. Hence, each filter is trained spatial in regard to the position in the volume it is applied to. Rectified linear non-linarites follow every hidden layer and all but the fourth convolutional layers are followed by 2x2 max pooling layer. The pooling layers aim to preserve the detected features in a smaller representation by discarding less significant data at the cost of spatial resolution.

The first two fully connected layers each have 4096 units and the final layer has different number of units based on the model implemented.

- 851 units are used in the final layer for CHAR-model where the maximum length of the word is 23 and each element is of ['0, ..., 9, a, ..., z, ' '] so it will have 23*37=851 units

- 88172 units in the DICT-model as the number of words in the dictionary is 88172.

The final classification layer is followed by a softmax normalization layer.

3.6 Recognition models

There are various recognition models which are based on the convolutional neural network. The following are two models for the word recognition purpose discussed in this research work [7, 8, 29].

3.6.1 Character Sequence Model

This model encodes the character at each position in the word and so predicts the sequence of characters in an image region. Each position in the word is modeled by independent classifier acting on a shared set of features from a single CNN. By construction, this model makes no assumptions about the underlying language and allows completely unconstrained recognition.

A word w of length N is modeled as a sequence of characters such that $w = (c_1, c_2, \ldots, c_N)$ where each $c_i \in C = \{1,2,\ldots,36\}$ represents a character at position i in the word, from the set of 10 digits and 26 letters. Each c_i can be predicted with a single classifier, one for each character in the word. However, since words have variable length N which is unknown at test time, the number of characters is fixed to N_{max} (implemented here to be 23) which is the maximum length of a word in the training set, and introduce a null character class. Therefore a word is represented by a string $w \in (C \cup \{ \Phi \})^{Nmax}$. For a given input image x, we want to return the estimated word w^* which maximizes $P(w^*|x)$. Since we seek an unconstrained recognition system with this model, we assume independence between characters leading to

$$w^* = \arg\max_{w} P(w|x)$$

$$w^* = \arg \max_{c1,c2,\ldots cN_{max}} \prod_{i=1}^{N_{max}} P(c_i|\Phi(x)) \quad (3.19)$$

Where $P(c_i|\Phi(x))$ is given by the classifier for the i^{th} position acting on a single set of shared CNN features $\Phi(x)$. The word w^* can be computed by taking the most probable character at each position $c_i^* = \arg\max_{ci\epsilon C \cup \{\Phi\}} P(c_i|\Phi(x))$.

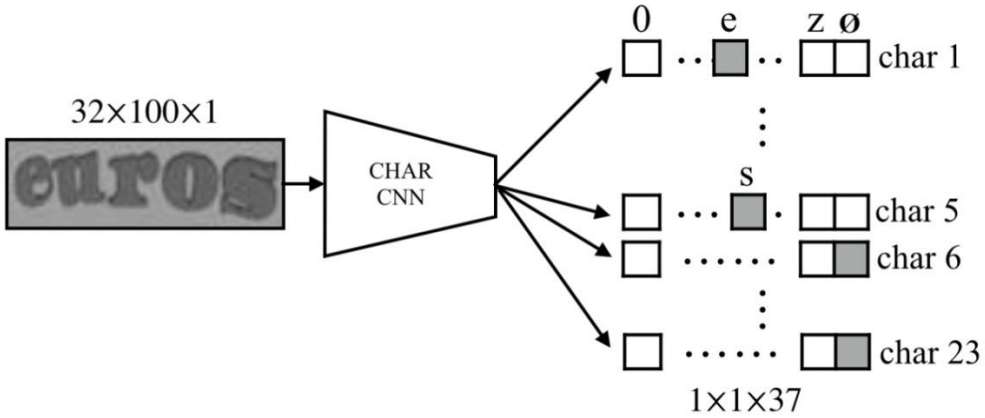

Fig 3.6: Character Sequence Model [8]

The CNN, as shown in the above figure, takes the whole word image x as input. Word images can be of different sizes, in particular due to the variable number of characters in the image. However, CNN of this model requires a fixed size input for all input images. This problem is overcome by simply resampling the original word image to a canonical height and width, without regard to preserving the aspect ratio, producing a fixed size input x. The base CNN has a number of convolutional layers followed by a series of fully connected layers, giving $\Phi(x)$. Rectified linear units are used throughout after each weight layer except for the last one. In forward order, the convolutional layers have 64, 128, 256, 512, and 512 square filters with an edge size of 5, 5, 3, 3, and 3. Convolutions are performed with stride 1 and there is input feature map padding to preserve spatial dimensionality. 2 × 2 max-pooling follows the first, second and third convolutional layers. The fully connected layers have 4096 units.

The output of the fully connected layer is then fed to 23 separate fully connected layers with 37 neurons each, one for each character class. These fully connected layers are independently softmax normalized and can be interpreted as the

probabilities $P(c_i|\Phi(x))$ of the width-resized input image x. $\Phi(x)$ is fed to N_{max} separate fully connected layers (one layer per character position) with 37 neurons each, one for each character class including the null character. These fully connected layers are independently softmax normalized and can be interpreted as the probabilities $P(c_i|\Phi(x))$ of the width-resized input image x.

The CNN is trained with multinomial logistic regression loss applied to each character position classifier, with the back-propagated gradients from each classifier accumulating on the penultimate fully connected layer, allowing the entire network to be trained through back-propagation and optimized with stochastic gradient descent (SGD) and dropout regularization.

3.6.2 Dictionary Encoding Model

To perform classification across a pre-defined dictionary of words dictionary encoding which explicitly models natural language. The cropped image of each of the proposed bounding boxes is taken as input to the CNN, and the CNN produces a probability distribution over all the words in the dictionary. The word with the maximum probability can be taken as the recognition result. The model can scale to a huge dictionary of 90k words, encompassing the majority of the commonly used English language. The synthetic data set has been used for the model and data is so realistic that the CNN can be trained purely on the synthetic data but still applied to real world data. The synthetic data engine produces a wide range of synthetic data samples, being drawn from a multitude of random distributions, mimicking real-world samples of scene text images. Here, the synthetic data is used in place of real-world data, and the labels are generated from a corpus or dictionary as desired.

Recognition process is formulated as a multi-class classification problem, with one class per word, where words w are constrained to be selected in a pre-defined dictionary W. While the dictionary W of a natural language may seem too large for this approach to be feasible, in practice an advanced English vocabulary, including different word forms, contains only around 90k words, which is large but manageable.

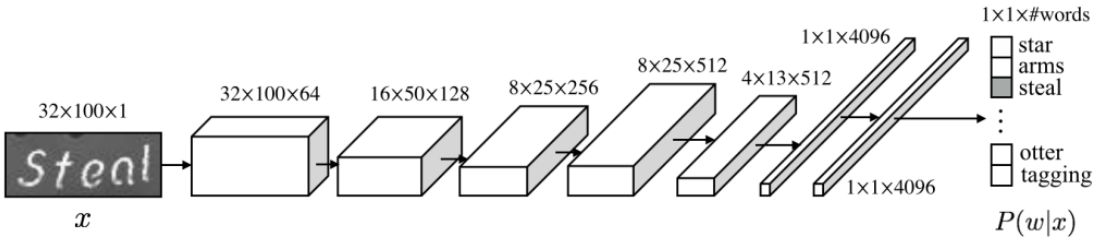

Fig. 3.7: Dictionary Encoding Model [8]

Here it is proposed to use a CNN classifier where each word w ϵ W in the lexicon corresponds to an output neuron. A CNN with five convolutional layers and three fully-connected layers is used where the final fully-connected layer performs classification across the dictionary of words, so it has the same number of units as the size of the dictionary we wish to recognize. The predicted word recognition result w^* out of the set of all dictionary words W in a language L for a given input image x is given by

$$w^* = \arg\max_{w} P(w|x, L) \tag{3.20}$$

With the assumptions that x is conditionally independent of L given w, i.e.$P(w|x, L)=(x|L)$, then $P(w|x, L)$can be written as

$$P(w|x, L) = \frac{P(w|x)P(w|L)P(x)}{P(x|L)P(w)} \tag{3.21}$$

and assuming that prior to any knowledge of our language all words are equally probable, our scoring function reduces to

$$w^* = \arg\max_{w\epsilon W} P(w|x)P(w|L) \tag{3.22}$$

The per-word output probability P (w|x) is modelled by the softmax output of the final fully-connected layer of the recognition CNN, and the language based word prior P (w|L) can be modeled by a lexicon or word frequency counts.

One limitation of this CNN model is that the input x must be a fixed, pre-defined size. This is problematic for word images, as although the height of the image is always one character tall, the width of a word image is highly dependent on the number of characters in the word, which can range between one and 23 characters. To overcome

this issue, the word image is simply reassembled to a fixed width and height. Although this does not preserve the aspect ratio, the horizontal frequency distortion of image features most likely provides the network with word-length cues. It has been also experimented with different padding regimes to preserve the aspect ratio, but found that the results are not quite as good as performing naive resampling.

3.7 Training and Testing

Feature extraction process is followed by the training and testing of the network. In the training of the recognition system the filters are learned from the input feature vectors. After the training of the filters testing is performed using the two datasets as mentioned in the following sections.

3.7.1 Stochastic Gradient Descent

The standard gradient descent algorithm updates the parameters θ of the objective $J(\theta)$ as,

$$\theta = \theta - \alpha \nabla_{\theta} E[J(\theta)] \tag{3.23}$$

where the expectation in the above equation is approximated by evaluating the cost and gradient over the full training set. The new update is given by,

$$\theta = \theta - \alpha \nabla_{\theta} J(\theta; x^{(i)}, y^{(i)}) \tag{3.24}$$

with a pair (x(i),y(i)) from the training set.

In SGD the learning rate α is typically much smaller than a corresponding learning rate in batch gradient descent because there is much more variance in the update. Choosing the proper learning rate and schedule (i.e. changing the value of the learning rate as learning progresses) can be fairly difficult. One standard method that works well in practice is to use a small enough constant learning rate that gives stable convergence in the initial epoch (full pass through the training set) or two of training and then halve the value of the learning rate as convergence slows down. An even better approach is to evaluate a held out set after each epoch and anneal the learning rate when the change in objective between epochs is below a small threshold. This tends to give good convergence to local optima.

Generally randomly shuffled data are presented to the algorithm prior to each epoch of training as data in some meaningful order as this can bias the gradient and lead to poor convergence.

3.7.2 Backpropagation

Backpropagation(BP) algorithm is used to adjust the filter values or weights of CNN. It is a training method that consists of mainly 4 distinct sections, the forward pass, the loss function, the backward pass, and the weight update. There are proposed various ways for speeding up the backpropagation learning process. One of the ways is to implement adaptive learning rate and momentum [30].

Typically for the conventional BP training algorithm, an error measure known as the mean square error (MSE) is used. The mean square error is defined as follows:

$$E_p = \frac{1}{2}\sum_{j=1}^{N}(t_{pj} - o_{pj})^2 \tag{3.25}$$

Where E_p is the error for the p^{th} presentation vector, t_{pj} is the desired value for the j^{th} output node and $o_{pj}$is the actual output of the j^{th} output node.

Algorithm:

1. Initialize all the value of w_{ji}, w_{kj} , q_j, q_k to small random values within the range [0, 1].
2. Apply the input vectors x_p to the input nodes.
3. Calculate the net-input values to the hidden layer nodes and the outputs from the hidden layer j.

$$o_{pj} = f(\sum w_{ji}x_i + q_j) \tag{3.26}$$

4. Move to the output layer k .Calculate the outputs as

$$o_{pk} = f(\sum w_{kj}o_{pj} + q_k) \tag{3.27}$$

5. Calculate the error terms for the output nodes.

$$d_{pk} = o_{pk}(1 - p_{pk})(t_{pk} - o_{pk}) \tag{3.28}$$

6. Calculate the error terms for the hidden nodes.

$$d_{pj} = o_{pj}(1 - o_{pj})(\sum_{j=1}^{n} w_{kj} d_{pk}) \quad (3.29)$$

7. Update weights and bias on the output layer

$$\Delta w_{kj}(n+1) = \sum_{p=1}^{P} h_k d_{pk}\, o_{pj} + a_k \Delta w_{kj}(n) \quad (3.30)$$

$$\Delta q_k(n+1) = \sum_{p=1}^{P} h_k d_{pk} + a_k \Delta q_k(n) \quad (3.31)$$

8. Update weights and bias on the hidden layer.

$$\Delta w_{ji}(n+1) = \sum_{p=1}^{P} h_j d_{pj}\, x_i + a_j \Delta w_{ji}(n) \quad (3.32)$$

$$\Delta q_j(n+1) = \sum_{p=1}^{P} h_j d_{pj} + a_j \Delta q_j(n) \quad (3.33)$$

9. Return to step 2 and repeat for each pattern *p* until the total error is acceptably small for each of the training vector pairs, training can be terminated.

3.7.3 Softmax Regression

Softmax regression is a generalized form of logistic regression which can be used in multi-class classification problems where the classes are mutually exclusive [26].

Given a training set $\{(x^{(1)},y^{(1)}),\ldots (x^{(m)}, y^{(m)})\}$ of m labeled examples, where the input features are $x^{(i)} \in \mathbb{R}^{n+1}$ and the labels $y^{(i)} \in \{1,2, \ldots k\}$. Given a test input x, the hypothesis to estimate the probability that p(y = j | x) for each value of j=1,2,…k. I.e., the estimated probability of the class label taking on each of the k different possible values will output a k-dimensional vector (whose elements sum to 1) giving k estimated probabilities. Concretely, hypothesis $h_\theta(x)$ takes the form:

$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1|x^{(i)};\theta) \\ p(y^{(i)} = 2|x^{(i)};\theta) \\ \vdots \\ p(y^{(i)} = k|x^{(i)};\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix} \quad (3.34)$$

where $\theta_1, \theta_2, \ldots, \theta_k \in \mathbb{R}^{n+1}$ are parameters of model and the model parameters θ were trained to minimize the cost function

$$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{k} 1\{ y^{(i)} = j \}\log\frac{\exp\left(\theta^{(k)T}x^{(i)}\right)}{\sum_{j=1}^{k}\exp(\theta^{(j)T}x^{(i)})}\right] \tag{3.35}$$

In softmax regression,

$$P\left(y^{(i)} = k\middle|x^{(i)};\theta\right) = \frac{\exp(\theta^{(k)T}x^{(i)})}{\sum_{j=1}^{k}\exp(\theta^{(j)T}x^{(i)})} \tag{3.36}$$

3.7.3 Dropout Regularization

The term "dropout" refers to dropping out units (hidden and visible) in a neural network. By dropping a unit out means temporarily removing it from the network along with all its incoming and outgoing connections. It prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently [31, 32]. Dropout also can be interpreted as a way of regularizing a neural network by adding noise to its hidden units.

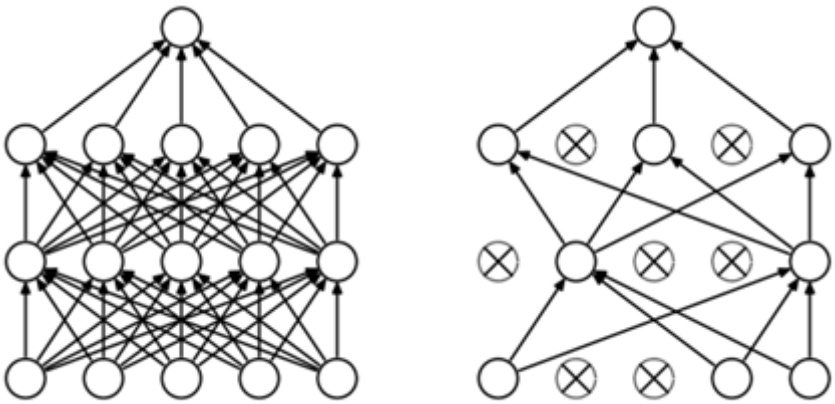

Fig.3.8: Neural net models: a. Standard Neural Network b. Neural Network After Applying Dropout [31]

Consider a neural network with L hidden layers. Let l ∈ {1,2,…, L} index the hidden layers of the network. Let $\mathbf{z}^{(l)}$ denote the vector of inputs into layer l, $y^{(l)}$ denote the vector of outputs from layer l ($\mathbf{y}^{(0)} = \mathbf{x}$ is the input). $W^{(l)}$ and $\mathbf{b}^{(l)}$ are the weights and

biases at layer l. The feed-forward operation of a standard neural network can be described as (for $l \in \{0, 1, \ldots, L-1\}$ and any hidden unit i)

$$z^{(l+1)} = w_i^{(l+!)} y^l + b_i^{(l+1)} \tag{3.37}$$

$$y_i^{(l+!)} = f\left(z_i^{(l+!)}\right) \tag{3.38}$$

where f is any activation function.

With dropout, the feed-forward operation becomes,

$$r_j^{(l)} \sim \text{Bernouli}(p)$$

$$\tilde{y}^{(l)} = r^{(l)} \times y^{(l)} \tag{3.39}$$

$$z^{(l+1)} = w_i^{(l+!)} \tilde{y}^{(l)} + b_i^{(l+1)} \tag{3.40}$$

$$y_i^{(l+1)} = f\left(z_i^{(l+1)}\right) \tag{3.41}$$

For any layer *l*, $\mathbf{r}^{(l)}$ is a vector of independent Bernoulli random variables each of which has probability *p* of being 1. This vector is sampled and multiplied element-wise with the outputs of that layer, $\mathbf{y}^{(l)}$, to create the thinned outputs $\tilde{\mathbf{y}}^{(\mathbf{l})}$**.** The thinned outputs are then used as input to the next layer. This process is applied at each layer. This amounts to sampling a sub-network from a larger network. For learning, the derivatives of the loss function are back-propagated through the sub-network. At test time, the weights are scaled as:

$$W_{\text{test}}^{(l)} = pW^{(l)} \tag{3.42}$$

Although dropout alone gives significant improvements, using dropout along with maxnorm regularization, large decaying learning rates and high momentum provides a significant boost over just using dropout [32].

Chapter 4

Experimentation and Results

The text recognition system with its models was implemented in Python 3.6.1 using Spyder 3.1.4 installed on a Intel(R) Core(TM) i5-3337U CPU @1.80GHz (4 CPUs) processors. The computer system is supported with 8GB RAM and Microsoft Windows 8 Pro 64-bit operating system.

Sections 4.1 and 4.2 describe the platform for implementations and in Section 4.3 the datasets used in this research work are described.

4.1 Python and Packages

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy which emphasizes code readability, and a syntax which allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale.

Python features a dynamic types system and automatic memory management and supports multiple programming paradigms, including object oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library for different purposes some of which are as follows.

- **Numpy**

  It is the fundamental package for scientific computing with Python. It contains a powerful N-dimensional array object, sophisticated (broadcasting) functions, tools for integrating C/C++ and Fortran code and useful linear algebra, Fourier transform, and random number capabilities. Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic

data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

- **Scipy**

It is a collection of mathematical algorithms and convenience functions built on the Numpy extension of Python. It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data. With SciPy an interactive Python session becomes a data-processing and system-prototyping environment rivaling systems such as MATLAB, IDL, Octave, R-Lab, and SciLab.

- **Matplotlib**

It is a library for making 2D plots of arrays in Python. Although it has its origins in emulating the MATLAB graphics commands, it is independent of MATLAB, and can be used in a Pythonic, object oriented way. Although Matplotlib is written primarily in pure Python, it makes heavy use of NumPy and other extension code to provide good performance even for large arrays.

- **Skimage**

The scikit-image SciKit (toolkit for SciPy) provides a versatile set of image processing routines. It is written in the Python language. It was developed by the SciPy community.

- **Theano**

It is a Python library that allows to define, optimize and evaluate mathematical expressions involving multi-dimensional arrays efficiently. Theano features are tight integration with NumPy, transparent use of a GPU, efficient symbolic differentiation, speed and stability optimizations, dynamic C code generation and extensive unit-testing and self-verification. It has been powering large-scale computationally intensive scientific investigations since 2007.

- **Keras**

  It is a high-level neural networks API, written in Python and capable of running on top of either TensorFlow, CNTK (Microsoft Cognitive toolkit) or Theano. It was developed with a focus on enabling fast experimentation.

4.2 Anaconda Distribution and Spyder

Anaconda is freemium open source distribution of Python and R programming languages (a language and environment for statistical computing and graphics) for large-scale data processing, predictive analytics, and scientific computing, that aims to simplify package management and deployment.

Anaconda Distribution is a free, easy-to-install package manager, environment manager and Python distribution with a collection of over 720 open source packages with free community support. It is platform-agnostic, so that it can be used on Windows, macOS or Linux. Package versions are managed by the package management system conda. It is a package manager application that quickly installs, runs, and updates packages and their dependencies.

Spyder is the **S**cientific **PY**thon **D**evelopment **E**nvi**R**onment and provides

- a powerful interactive development environment for the Python language with advanced editing, interactive testing, debugging and introspection features
- a numerical computing environment with the support of IPython (enhanced interactive Python interpreter) and popular Python libraries such as Numpy (linear algebra), SciPy (signal and image processing) or matplotlib (interactive 2D/3D plotting).

Spyder may also be used as a library providing powerful console-related widgets for PyQt-based applications and may be used to integrate a debugging console directly in the layout of the graphical user interface.

4.3 Training and Testing Datasets

In this section the datasets used throughout this research are described. The text recognition methods in this research are largely based on supervised machine learning

requiring labeled training datasets. There is a number of pre-existing publicly available datasets for text recognition. Two datasets Synth90k [8] and ICDAR 2003 [33] are taken. Synth90k is a dataset which contains the synthetic images which makes a neural network to learn faster [8]. The second datasets contains the challenging images from the real scene images. The training of the models is performed purely using the synthetic data Synth90k. The models are then evaluated on a real-world standard datasets ICDAR 2003 and Synth.

4.3.1 Synth90k

This synthetically generated dataset consists of 9 million word images with equal numbers of word samples from a 90k word dictionary. 900k of these images is used for a testing dataset, 900k for validation, and the remaining for training. This dataset has been generated through the synthetic data generation process which involved font rendering, border/shadow rendering, base coloring, projection distortion and natural data blending [8].

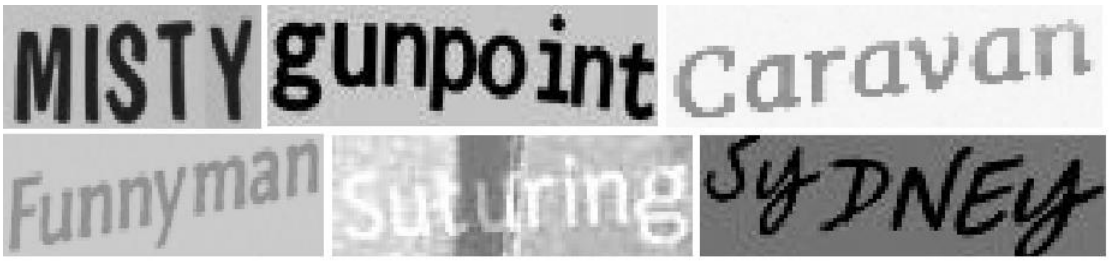

Fig 4.1: Sample Datasets from Synth90k [8]

The above figure shows some of the testing examples from the Synth90k dataset used in this research work.

4.3.2 ICDAR 2003

In this dataset, images were captured with a variety of digital cameras which were used with a range of resolution and other settings, with the particular settings chosen at the discretion of the photographer and is explained in the ICDAR-2003 paper [33].

In this dataset, the data is organized into: Sample, Trial and Competition datasets. Sample datasets were provided to give a quick impression of the data in the dataset, and also to allow functional testing of the software which means that the researchers

could check that their software could read and write the specified dataset formats, but not get any statistically meaningful results. Trial datasets had two intended uses. It could be used to get results for ICDAR 2003 papers. For this purpose, it is partitioned into two sets namely TrialTrain and Trial Test. TrialTrain contains 1126 images while TrailTest contains 1110 images. This dataset is used in this research work for testing. The images in Competition are intended for the Text recognition competition.

Fig 4.2: Sample datasets from ICDAR dataset [32]

The above figure shows some of the challenging sample images from ICDAR-2003 used for the classification.

4.4 Results

Firstly both models Character Sequence Model (CHAR-Model) and Dictionary Encoding Constrained Model(DICT-Model) were tested with the Synth90k dataset with 5000 images and then with ICDAR Trial datasets with 1116 images. Recognition Accuracy was used as the comparative measure for evaluating the efficiency of the models and is computed using the following equation.

$$\text{Recognition Accuracy} = \frac{\text{No. of correclty classified images}}{\text{Total number of labelled images}} \tag{4.1}$$

Each labeled input image is fed to the both models and the classified result is stored in a text file. Based on the comparison between each pair of actual label of the image and the obtained classified result for 5,000 test images from the dataset Synth90k, the recognition accuracy for each model is computed using equation 5.1. The following table shows the computed recognition accuracy and miss-classification rate for the models.

No. of Testing Images	Recognition Model	Recognition Accuracy (%)	Miss-classification Rate (%)
5000	CHAR-Model	79.92	20.08
	DICT-Model	93.88	6.12

Table 5.1: Recognition Accuracy result for the recognition models using Synth90k dataset

The following bar-graph shows the overall comparison between CHAR-model and DICT-model for the dataset Syth90k. DICT-model outperformed in the experiment by obtaining higher accuracy of 93.88%

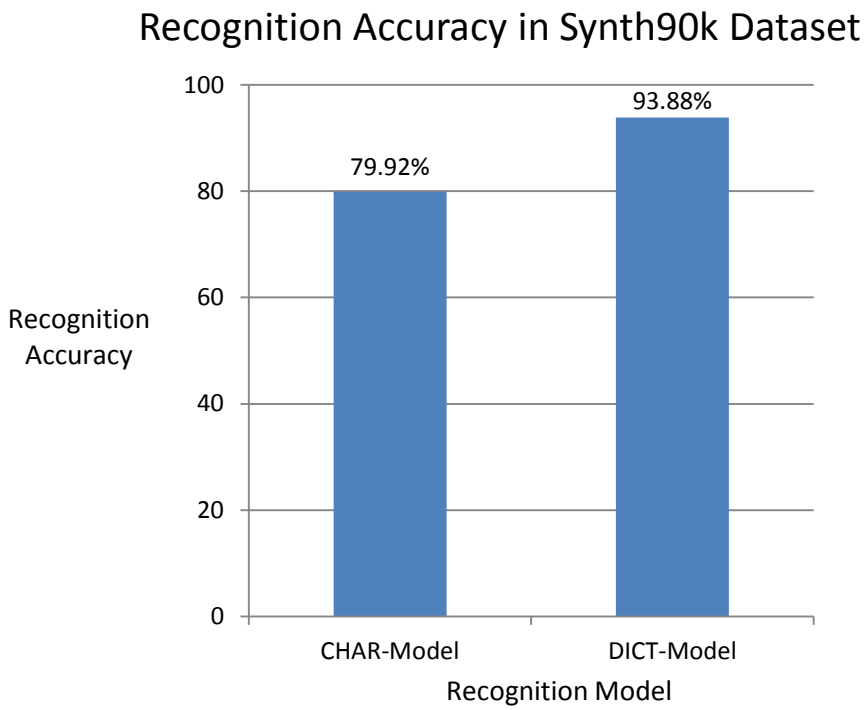

Figure 4.3 Comparison between two proposed models CHAR-model and DICT-model suing the Synth90k dataset

In the same way, 1110 images are taken from ICDAR-2003 dataset and each image is classified using both models and the recognition accuracy is computed using equation 4.1. The following table shows the computed recognition accuracy for both the models.

No. of Testing Images	Recognition Model	Recognition Accuracy (%)	Miss-classification Rate (%)
1110	CHAR-Model	53.72	46.28
	DICT-Model	70.33	29.67

Table 5.2: Recognition Accuracy result for the recognition models using ICDAR-2003 dataset

The following figure shows the overall comparison between CHAR model and DICT model for the dataset ICDAR-2003 based on the experimentation and again, the DICT model outperformed the CHAR-model.

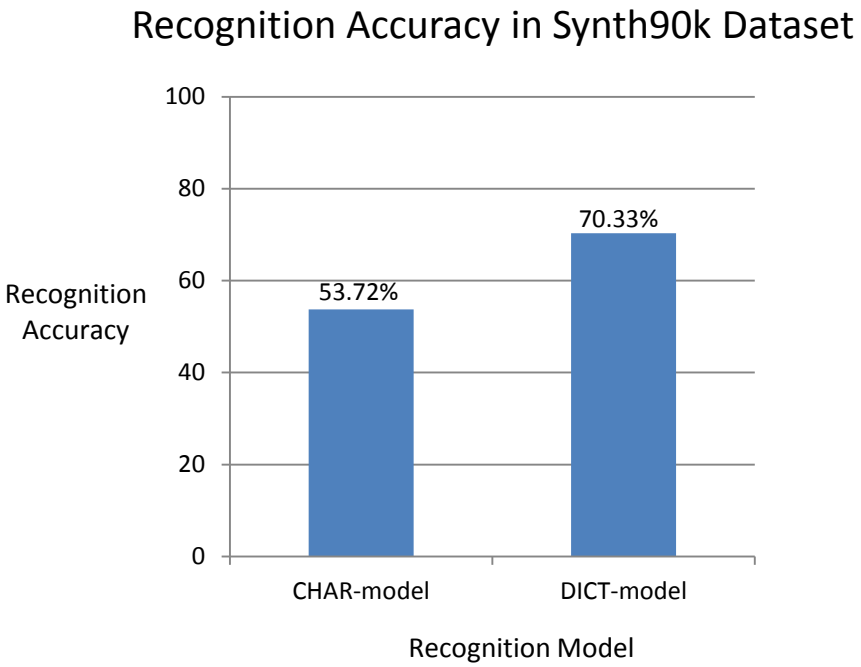

Figure 4.4: Comparison between two proposed models CHAR-model and DICT-model using ICDAR-2003 dataset

4.5 Discussion and Analysis

The convolutional neural network is based on the feature extraction through the convolution operation such that it requires less preprocessing operations. Only the preprocessing with grayscale conversion, resizing and the normalization provides the significant result in the recognition task. Only the thing is that it requires lots of training examples for the training for better result and accuracy.

The two models, as mentioned in the Sections 3.6.1 and 3.6.2, are based on the CNN architecture as described on Section 3.5 While evaluating both the models, DICT model has reached a recognition rate of 93.88% in Synth90k dataset and 70.33% in ICDAR 2003 which outperformed the CHAR model. Based on the observation of the classified list of predicted labels for the images, it is found that CHAR model failed to predict the correct recognition as it is based on predicting each character one-by-one and merging the characters to form a word. Even single character miss-classification on this model, failed the whole classification process. But in case of DICT model as it is based on extracting the features from the whole text, it gave better recognition result.

Another thing is that CHAR-model is unconstrained model and does not use any dictionary or any word related heuristics, it has got lower accuracy result. In contrast, DICT-model is a constrained language model and uses a dictionary with 88172 different words. It resulted in the better accuracy. But, it also leads to one limitation in this model. This limitation is that it could not recognize the words which are beyond the words from the dictionary. So, it makes the model as dictionary constrained model.

While analyzing the time for the classification for the models, DICT model is little bit slower in comparison to CHAR model as DICT model is based on the use of dictionary of size 88172 words which consumes little more resources and time while CHAR model is language unconstraint model resulting in faster recognition. So, although DICT model has better recognition accuracy for the classification, CHAR model is faster for the computation and classification. Timing of both the models can be improved by using GPUs or highly optimized distributed CPU architectures as suggested in [33].

There are certain limitations in both models which are as follows:

- Both models are trained to recognize the words up to the length of 23 characters.
- Similarly, both models are the character case insensitive models and are not trained to recognize different symbols such as &, -, ’ , and so on.

DICT model has one more limitation. In this model, there is no support for reading vertical or significantly rotated text. So, during recognition the height resizing to 32×100 px will cause rotated text to be unreadably distorted.

Chapter 5

Conclusion and Future Works

5.1 Conclusion

Text recognition from an image with deep convolutional neural network is presented and two models namely Character Sequence Model and Dictionary Encoding Model were analyzed and evaluated with the English language cropped image word datasets. The steps in the deep convolutional neural network architecture were investigated in full detail with the implementation for the scene word recognition.

For the word recognition in the two proposed models, CHAR-model is based on character sequence encoding model while DICT-model is based on the Dictionary encoding model. Both models were experimented for the recognition of scene text with the two publicly available text recognition datasets namely, Synth50k with the synthetic data and ICDAR 2003 dataset. 5000 text images were taken from Synth50k datasets for testing in both models and 1156 text images from the ICDAR 2003 dataset. On evaluation in both datasets DICT-model outperforms with the accuracy of 93.88% in Synth50k dataset and 70.33 in ICDAR 2003 dataset while the CHAR-model was successful for achieving accuracy of 79.92% in Synth50k dataset and 53.72% in ICDAR 2003 dataset. Although the DICT-model takes little more time for recognition, it recognized the words with more accuracy.

5.2 Future Works

Text recognition is one of the research topics from many decades and has been addressed by different methods for the recognition by a huge number of researchers. Although lots of research has been done, there is not any method for scene text recognition which can recognize all types of images with different symbols and text orientations. These methods included in this research work also can be improved for

recognizing different kinds of images and even in noisy images. The future scopes of this research are as mentioned below:

- Proposed system can be tested for the other well-known text recognition and detection datasets such as Street View Text Dataset, MSRA database, IIIT 5k-word Dataset, and other ICDAR datasets for further evaluation to identify the accuracy of the models such that the modification in the models can be done.
- It can be integrated with the text localization system such that a full end-to-end text detection and recognition system can be developed which directly takes an image to the system and return the texts from the image in computer readable form.
- It can be extended to multi-lingual text recognition system.

Appendix A:

Sample Source Code

A1. Create CHAR net model

```
from __future__ import print_function
import numpy as np
np.random.seed(1337)  # for reproducibility
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Flatten
from keras.layers.convolutional import Convolution2D, MaxPooling2D,
CustomZeroPadding2D
from keras.optimizers import SGD

output_str = '0123456789abcdefghijklmnopqrstuvwxyz '
output = [x for x in output_str]
L = len(output)

if __name__ == '__main__':
   all_weights = np.load('matlab_charnet_weights.npz')

   img_rows, img_cols = 32, 100
    nb_filters = [64,128,256,512,512]
   nb_pool = 2
        nb_conv = [5,5,3,3,3]

   layer1, layer2, layer3, layer35, layer4 = range(5)
   model = Sequential()

   # First layer - border_mode = 'same' preserves dimensionality
   weights = [all_weights['conv1W'],all_weights['conv1b']]
     model.add(Convolution2D(nb_filters[layer1],nb_conv[layer1],
     nb_conv[layer1],weights=weights,
       input_shape=(1, img_rows,img_cols),activation='relu',border_mode='same'))

   model.add(MaxPooling2D(pool_size=(nb_pool, nb_pool)))

   # Second layer - border_mode = 'same' preserves dimensionality
   weights = [all_weights['conv2W'],all_weights['conv2b']]
   model.add(Convolution2D(nb_filters[layer2], nb_conv[layer2],
nb_conv[layer2],weights=weights,
       border_mode='same',activation='relu',init='glorot_uniform'))
```

```
    model.add(MaxPooling2D(pool_size=(nb_pool, nb_pool)))

    # Third layer (no max pooling per model software) - border_mode = 'same'
preserves dimensionality
    weights = [all_weights['conv3W'],all_weights['conv3b']]
    model.add(Convolution2D(nb_filters[layer3], nb_conv[layer3],
nb_conv[layer3],weights=weights,
        border_mode='same',activation='relu',init='glorot_uniform'))

# 3.5 layer - border_mode = 'same' preserves dimensionality
    weights = [all_weights['conv35W'],all_weights['conv35b']]
    model.add(Convolution2D(nb_filters[layer35], nb_conv[layer35],
nb_conv[layer35],weights=weights,
        border_mode='same',activation='relu',init='glorot_uniform'))

    model.add(CustomZeroPadding2D(padding=(0,0,0,1)))
    model.add(MaxPooling2D(pool_size=(nb_pool, nb_pool)))

    # Fourth layer - border_mode = 'same' preserves dimensionality
    weights = [all_weights['conv4W'],all_weights['conv4b']]
    model.add(Convolution2D(nb_filters[layer4], nb_conv[layer4],
nb_conv[layer4],weights=weights,
        border_mode='same',activation='relu',init='glorot_uniform'))

    # First Dense layer
    model.add(Flatten())
    weights = [all_weights['dense1W'],all_weights['dense1b']]
    model.add(Dense(4096,activation='relu',weights=weights))
    model.add(Dropout(0.5))

    # Second Dense layer
    weights = [all_weights['dense2W'],all_weights['dense2b']]
    model.add(Dense(4096,activation='relu',weights=weights))
    model.add(Dropout(0.5))

    # Classification layer
    weights = [all_weights['classW'],all_weights['classb']]
    model.add(Dense(851,activation='softmax',weights=weights))

    sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
    model.compile(loss='categorical_crossentropy', optimizer=sgd)

    json_string = model.to_json()
    open('char2_architecture.json', 'w').write(json_string)
```

```
    model.save_weights('char2_weights.h5',overwrite=True)
```

A2. Feature Extraction and Classification

```
from __future__ import print_function
from numpy import argmax, array, dot, float32, mean, std, zeros
import os
os.environ['THEANO_FLAGS'] = "device=cpu"

class CharNet():
    def __init__(self, architecture_file=None, weight_file=None, optimizer=None):
        output_str = '0123456789abcdefghijklmnopqrstuvwxyz '
        self.output = [x for x in output_str]
        self.L = len(self.output)

        # Load model and saved weights
        from keras.models import model_from_json
        print('')
        if architecture_file is None:
            self.model =
model_from_json(open(../CHAR2/char2_architecture.json').read())
        else:
            self.model = model_from_json(open(architecture_file).read())

        if weight_file is None:
            self.model.load_weights('../CHAR2/char2_weights.h5')
        else:
            self.model.load_weights(weight_file)

        if optimizer is None:
            from keras.optimizers import SGD
            optimizer = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
        self.model.compile(loss='categorical_crossentropy', optimizer=optimizer)

    def _rgb2gray(self,rgb):
        return dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])

    def _preprocess(self,img):
        from skimage.transform import resize
        if len(img.shape) == 3 and img.shape[2] == 3:
            img = self._rgb2gray(img)
        img = resize(img, (32,100), order=1, preserve_range=True)
        img = array(img,dtype=float32)
```

```
        img = (img - mean(img)) / ( (std(img) + 0.0001) )
        print('preprocessed')
        return img

    def classify_image(self,img):
        img = self._preprocess(img)
        xtest = zeros((1,1,32,100))
        xtest[0,0,:,:] = img
        z = self.model.predict(xtest,verbose=0)[0,:]
        output_text = []
        for i in range(23):
            ind = argmax(z[i*self.L:(i+1)*self.L])
            output_text.extend(self.output[ind])
        return ''.join(output_text).rstrip(' ')

if __name__ == '__main__':
    import matplotlib.image as mpimg
    dir_prefix = '../IMAGES/'
    filename = dir_prefix + 'num.jpg'
    cnn_model = CharNet()

    img = mpimg.imread(filename)
    print (cnn_model.classify_image(img))
```

A3. Create DICT net model

```
from __future__ import print_function
import numpy as np
np.random.seed(1337)  # for reproducibility

from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Flatten
from keras.layers.convolutional import Convolution2D, MaxPooling2D,
CustomZeroPadding2D
from keras.optimizers import SGD
if __name__ == '__main__':
    all_weights = np.load('matlab_dictnet_weights.npz')
    img_rows, img_cols = 32, 100
    nb_filters = [64,128,256,512,512]
    nb_pool = 2
    nb_conv = [5,5,3,3,3]
    layer1, layer2, layer3, layer35, layer4 = range(5)
    model = Sequential()
```

```
    # First layer - border_mode = 'same' preserves dimensionality
    weights = [all_weights['conv1W'],all_weights['conv1b']]
    model.add(Convolution2D(nb_filters[layer1], nb_conv[layer1],
nb_conv[layer1],weights=weights,
        input_shape=(1, img_rows, img_cols),activation='relu',border_mode='same'))

    model.add(MaxPooling2D(pool_size=(nb_pool, nb_pool)))

    # Second layer - border_mode = 'same' preserves dimensionality
    weights = [all_weights['conv2W'],all_weights['conv2b']]
    model.add(Convolution2D(nb_filters[layer2], nb_conv[layer2],
nb_conv[layer2],weights=weights,
        border_mode='same',activation='relu',init='glorot_uniform'))

    model.add(MaxPooling2D(pool_size=(nb_pool, nb_pool)))

    # Third layer (no max pooling per model software) - border_mode = 'same'
preserves dimensionality
    weights = [all_weights['conv3W'],all_weights['conv3b']]
    model.add(Convolution2D(nb_filters[layer3], nb_conv[layer3],
nb_conv[layer3],weights=weights,
        border_mode='same',activation='relu',init='glorot_uniform'))

    # 3.5 layer - border_mode = 'same' preserves dimensionality
    weights = [all_weights['conv35W'],all_weights['conv35b']]
    model.add(Convolution2D(nb_filters[layer35], nb_conv[layer35],
nb_conv[layer35],weights=weights,
        border_mode='same',activation='relu',init='glorot_uniform'))

    model.add(CustomZeroPadding2D(padding=(0,0,0,1)))
    model.add(MaxPooling2D(pool_size=(nb_pool, nb_pool)))
    # Fourth layer - border_mode = 'same' preserves dimensionality
    weights = [all_weights['conv4W'],all_weights['conv4b']]
    model.add(Convolution2D(nb_filters[layer4], nb_conv[layer4],
nb_conv[layer4],weights=weights,
        border_mode='same',activation='relu',init='glorot_uniform'))

    # First Dense layer
    model.add(Flatten())
    weights = [all_weights['dense1W'],all_weights['dense1b']]
    model.add(Dense(4096,activation='relu',weights=weights))
    model.add(Dropout(0.5))

    # Second Dense layer
```

```
weights = [all_weights['dense2W'],all_weights['dense2b']]
model.add(Dense(4096,activation='relu',weights=weights))
model.add(Dropout(0.5))

# Classification layer
weights = [all_weights['classW'],all_weights['classb']]
model.add(Dense(88172,activation='softmax',weights=weights))

sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd)

json_string = model.to_json()
open('dict2_architecture.json', 'w').write(json_string)
model.save_weights('dict2_weights.h5')
```

A4. Feature Extraction and Classification using DICTnet Model

```
from __future__ import print_function
from numpy import array, dot, float32, mean, std, zeros
class DictNet():
    def __init__(self, architecture_file=None, weight_file=None, optimizer=None):
        import scipy.io as sio
        lex_file = '../DICT2/lex.mat'
        mat_contents = sio.loadmat(lex_file)
        self.output_word = mat_contents['lexicon'][0,:]

        from keras.models import model_from_json
        if architecture_file is None:
            self.model =
model_from_json(open('../DICT2/dict2_architecture.json').read())
        else:
            self.model = model_from_json(open(architecture_file).read())
        if weight_file is None:
            self.model.load_weights('../DICT2/dict2_weights.h5')
        else:
            self.model.load_weights(weight_file)

        if optimizer is None:
            from keras.optimizers import SGD
            optimizer = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
        self.model.compile(loss='categorical_crossentropy', optimizer=optimizer)

    def _rgb2gray(self,rgb):
```

```
        return dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])

    def _preprocess(self,img):
        from skimage.transform import resize
        if len(img.shape) == 3 and img.shape[2] == 3:
            img = self._rgb2gray(img)
        img = resize(img, (32,100), order=1, preserve_range=True)
        img = array(img,dtype=float32)
        img = (img - mean(img)) / ( (std(img) + 0.0001) )
        return img

    def classify_image(self,img):
        img = self._preprocess(img)
        xtest = zeros((1,1,32,100))
        xtest[0,0,:,:] = img
        z = self.model.predict_classes(xtest,verbose=0)[0]
        return self.output_word[z][0]


if __name__ == '__main__':
    import matplotlib.image as mpimg
    dir_prefix = '../IMAGES/'
    filename = dir_prefix + 'sample.jpg'
    cnn_model = DictNet()

    img = mpimg.imread(filename)
    print (cnn_model.classify_image(img))
```

References

[1] R. Pagariya, M. Bartere, "Review Paper on Artificial Neural Networks", *International Journal of Advanced Research in Computer Science,* 2013.

[2] Y. LeCun, Y. Bengio, G. Hinton," Deep learning", *Macmillan Publishers Limited,* 2015.

[3] I. Goodfellow, Y. Bengio, A. Courville, "Deep Learning", *MIT Press,* 2016.

[4] S.Pal, A. Culli, "Deep Learning with Keras", *Packt Publishing, 2017.*

[5] C. Yao, X. Bai, W. Liu, "A Unified Framework for Multi-Oriented Text Detection and Recognition", *IEEE Transactions on Image Processing,* Volume 23, Issue 11, 2014.

[6] Y. Zhu, C. Yao, X. Bai, "Scene Text Detection and Recognition: Recent Advances and Future Trends", *Frontiers of Computer Science: SPCU,* Volume 10 Issue 1, Pages 19-36, 2016.

[7] M. Jaderberg, K. Simonyan , A. Vedaldi, A. Zisserman ,"Reading Text in the Wild with Convolutional Neural Networks", *International Journal of Computer Vision,* Volume 116 Issue 1, Pages 1-20, 2016.

[8] M. Jaderberg, K. Simonyan, A. Vedaldi , A. Zisserman," Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition", *Visual Geometry Group,* 2014.

[9] K. Wang and S. Belongie, "Word spotting in the wild" *In Proc. of ECCV,* 2010.

[10] L. Neumann and J. Matas, "A method for text localization and recognition in real-world images", *ACCV: Computer Vision* pp 770-783, 2010.

[11] L. Neumann, J. Matas, "Real-Time Scene Text Localization and Recognition", *IEEE Conference on Computer Vision and Pattern Recognition,* 2012.

[12] C. Yao, X. Bai, W. Liu, Y. Ma, and Z. Tu. "Detecting texts of arbitrary orientations in natural images. *In Proc. of CVPR,* 2012.

[13] A. Mishra, k. Alahari, and C. V. Jawahar, "Top-down and bottom-up cues for scene text recognition", *In Proc. of CVPR,* 2012.

[14] T. Novikova, O. Barinova, P. Kohli, V. Lempitsky, "Large-lexicon attribute-consistent text recognition in natural images", *ECCV*, pp. 752–765. Springer, 2012.

[15] M. Mohri, "Weighted Finite-State Transducer Algorithms An Overview", *Studies in Fuzziness and Soft Computing,* volume 148, pp 551-563, 2004.

[16] T. Wang, D. J. Wu, A. Coates, A. Y. Ng, “End-to-End Text Recognition with Convolutional Neural Networks”, *Pattern Recognition (ICPR),* 2012.

[17] B. Su, S. Lu, S. Tian, J. H. Lim, C. L. Tan, “Character Recognition in Natural Scenes using Convolutional Co-occurrence HOG”, *Pattern Recognition (ICPR*), 2014.

[18] K. Wang, B. Babenko, and S. Belongie “End-to-end scene text recognition", *Proceedings of the International Conference on Computer Vision. IEEE*, pp. 1457, 2011.

[19] M. Jaderberg , K. Simonyan, A. Vedaldi, A. Zisserman, “Deep structured output learning for Unconstrained text recognition”, *CLR*, 2015.

[20] L. Neumann, J. Matas,”Scene Text Localization and Recognition with Oriented Stroke Detection”, *Proceedings of the International Conference on Computer Vision*, pp. 97-104, 2013.

[21] O. Alsharif, J. Pineau ,“End-to-End Text Recognition with Hybrid HMM Maxout Models”, *International Conference on Learning Representations*, 2014.

[22] J. Almazan, A. Gordo, A. Fornes, E. Valveny, “Word Spotting and Recognition with Embedded Attributes”, *ICCV*, 2013.

[23] A. Mishra, K. Alahari, C. Jawahar, et al. (2012), “Scene text recognition using higher order language priors”, *BMVC- 23rd British Machine Vision Conference,* 2012.

[24] V. Goel, A. Mishra, K. Alahari, and C. V. Jawahar,”Whole is greater than sum of parts: Recognizing scene text words”, *ICDAR*, pp. 398-402, 2013.

[25] A. Bissacco, M. Cummins, Y. Netzer, and H. Neven, “PhotoOCR: Reading Text in Uncontrolled Conditions”, *Proceedings of the International Conference on Computer Vision*, 2013.

[26] T. Wiatowski, H.Boleskei, “A Mathematical Theory of Deep Convolutional Neural Networks for Feature Extraction”, *Dept. IT & EE*, 2016.

[27] J. Gu, Z.Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, “Recent Advances in Convolutional Neural Networks”, *arXiv:1512.07108*, 2015.

[28] S. Sun, W. Chen, L. Wang, L. Xiaoguang, T. Y. Liu ,“On the Depth of Deep Neural Networks: A Theoretical View”, *AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

[29] M. Jaderberg, A. Vedaldi, A. Zisserman, “Deep Features for Text Spotting”, *Visual Geometry Group, Department of Engineering Science, University of Oxford* , 2014.

[30] N. Srivastava, G.Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, *Department of Computer Science University of Toront, Canada*, 2014.

[31] Z. Li1, B. Gong, T. Yang,” Improved Dropout for Shallow and Deep Learning”, *NIPS*, 2016.

[32] S. M. Lucas, A. Panaretos, L. Sosa, A. Tang, S. Wong, R. Young,” ICDAR 2003 Robust Reading Competitions”, *Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR’03)*, 2003.

[33] M. Jaderberg, A. Vedaldi, A. Zisserman,” Speeding up Convolutional Neural Networks with Low Rank Expansions”, *British Machine Vision Conference (BMVC)*, 2014.