Medical Prescription Identification Solution

W.R.A.D Wijewardena
Index No: 14001578

Supervisor: Prof. N.D. Kodikara

January 2019

Submitted in partial fulfillment of the requirements of the B.Sc in Computer Science Final Year Project (SCS4124)

DECLARATION

I certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate Name: W. R. A. D. Wijewardena

....

Signature of Candidate   Date:

This is to certify that this dissertation is based on the work of
Ms. W. R. A. D. Wijewardena
under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Supervisor's Name: Prof. N. D. Kodikara

....

Signature of Supervisor   Date:

ABSTRACT

A medical prescription is a very familier document to any person and is a one that is usually believed impossible to read. There are several reasons for that well known conclustion. First, the sloppy handwriting of the doctors and second, the lack of the domain knowledge. Because of these difficulties in reading a prescription have a high probability of ending up in misreading the content. These misreading often lead to many health issues with regard to the patient and even a threat to their lives. But unfortunatly both the mentioned reasons for such situations cannot be changed.

In the present, one of the leading research area is Optical Character Recognition. Among that, handwritten character recognition takes a significant interest in researchers. Taking these advantaged into account and with the help of the domain knowledge, this research is to find a way to accuratly read the content of a medical prescription. This research uses a neural network approach for the charater recognition process and a knowledge base matching to accuralty output the result. The outcome of this research has been a successfull enhancement in the prescription identification domain and has established for further improvements.

PREFACE

The work presented in this study has utilize image processing and neural network. The system has been built using python, OpenCV and TensorFlow. The design and implementation of the neural network is based on the work found in githubharald/SimpleHTR repository on GitHub. Knowledge base has been constructed by the author and been overlooked by the advisor. Integration of the knowledge base to the recogntion system is also been done by the author. The evaluation of the study is done in collaboration with the domain advisor.

ACKNOWLEDGEMENT

I would like to thank my Supervisor Prof. N.D. Kodikara for letting me select this project and guiding me through out the entire project. Special thank for encouraging me at every time to finish the tasks properly and ahead in time and for taking me all the way till the end in the right path.

Dr. Pamoda Kamaladasa of General Hospital, Colombo and M. L. T Perera of New Nawala Chemists for their help and advices given in terms of medical domain.

Ms. Soma Kodithuwakku of Kothalawala Defense Universiry for the help and adivces given in terms of medical domain as well as for the help in the preparing the dataset and the evaluation process.

My parents and colleagues for all the support and strength given to me through out the project.

CONTENTS

Declaration . . . i

Abstract . . . i

Preface . . . ii

Acknowledgement . . . iii

Contents . . . v

List of Figures . . . vii

List of Tables . . . viii

1. Chapter 1 - Introdcution . . . 1
- 1.1 Background to the Research . . . 1
- 1.2 Research Problem and Research Questions . . . 4
  - 1.2.1 Research Question . . . 4
  - 1.2.2 Research Aim . . . 4
  - 1.2.3 Research Objectives . . . 4
- 1.3 Justification for the research . . . 4
- 1.4 Methodolody . . . 5
- 1.5 Outline of the Disseration . . . 5
- 1.6 Definitions . . . 5
- 1.7 Delimitation of the Scope . . . 6
- 1.8 Conclusion . . . 6

2. Chapter 2 - Literature Review . . . 7
- 2.1 Image Binarization . . . 7
- 2.2 Domain Specific Handwritten Character Recognition . . . 9

3. Chapter 3 - Research Design . . . 14
- 3.1 A Conceptual Overview of the Project . . . 14
  - 3.1.1 Preprocessing Subsystem . . . 15
  - 3.1.2 Recognition Subsystem . . . 17
  - 3.1.3 Knowledge Base Matching Subsystem . . . 18
- 3.2 Knowledge Base . . . 19
- 3.3 Evaluation Design . . . 19
- 3.4 Chapter Summary . . . 19

4. Implementation 20
4.1 Technologies Used 20
4.1.1 Adobe Photoshop CC 2015 20
4.1.2 Python 3.5 20
4.2 Training Dataset - IAM Handwriting Database 21
4.3 Implementation 22
4.3.1 Data 22
4.3.2 Using TensorFlow 24
4.3.3 Matching with Knowledge Base 25
4.4 Chapter Summary 26

5. Results and Evaluation 27
5.1 Evaluation Dataset 27
5.2 Final Evaluation 27
5.2.1 Segmented Single Line images 28

6. Conclutions 30
6.1 Introcution 30
6.2 Conclusion about the Research Question 30
6.3 Conclusion about the Research Problem 30
6.4 Limitations 30
6.5 Implications for further research 31

Bibliography 31

7. Bibliography 32

Appendices 33

Code Listing 34
7.1 dataloader.py 35
7.2 samplepreprocessor.py 38
7.3 Model.py 39
7.4 main.py 45

LIST OF FIGURES

1.1 Sample Prescriptions . . . 1
1.2 Case Summary . . . 2
1.3 Abbreviations that are possible for being misread . . . 2
1.4 Percentage of errors per prescription . . . 3

3.1 High Level Architecture . . . 14
3.2 High Level Architecture of the Preprocessing Subsystem . . . 15
3.3 Example Raw Input Images . . . 15
3.4 Problametic Raw Input Images . . . 16
3.5 Segmented Images . . . 16
3.6 Resizing the Image . . . 16
3.7 Output of Preprocessing Subsystem . . . 17
3.8 High Level Architecture of the Recognition Subsystem . . . 17
3.9 The NN written as a mathematical function which maps an image M to a character sequence (c1, c2, . . .) . . . 17
3.10 High level Architecture of the Knowledge Base Matching Subsystem . 18

4.1 Features in the input image . . . 22
4.2 Top: output matrix of the RNN layers. Middle: input image. Bottom: Probabilities for the characters “l”, “i”, “t”, “e” and the CTC blank label. . . . 23
4.3 CNN Kernel . . . 24
4.4 CNN Relu and Pool . . . 24
4.5 Creating RNN . . . 24
4.6 Bidirectional . . . 24
4.7 Setting up . . . 25
4.8 CTC Loss and Decoder . . . 25
4.9 Training NN . . . 25
4.10 Loading the Knowledge Base . . . 25
4.11 Calculate the matching ratio . . . 26
4.12 Obtaining the maximum value and the resulting drug name . . . 26

5.1 Segmented Single Line images . . . 28

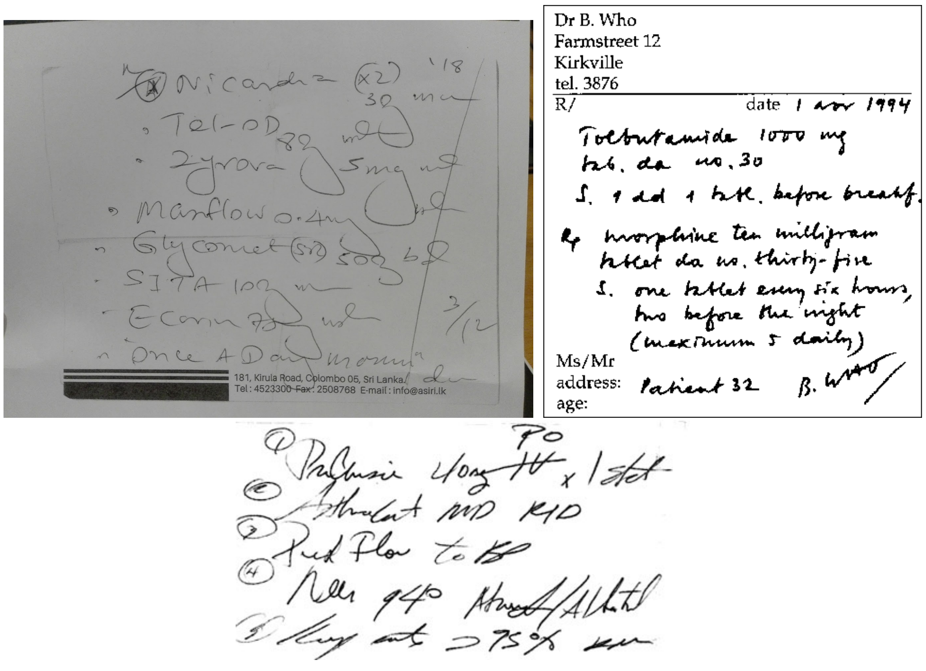
LIST OF TABLES

5.1 Results. 28

1. CHAPTER 1 - INTRODCUTION

1.1 Background to the Research

Prescriptions are mostly used as note on prescribed or recommended drugs for a certain patient. A prescription contains the name of the drug (sometimes even the brand name or the company name of the manufacturer), dosage of the drug and the duration of use. In practice, most of the doctors write their prescriptions at their convenience rather than in a manner that can be identified by everyone and they have a specific pattern on how to write these prescriptions. So most of the times, patient or any other normal person cannot understand what the prescription says about the drugs that they are supposed to use. Other than that, for many doctors, they have their own method of writing too. Fig 1. Shows few such sample prescriptions.



'18
Nicardia (x2) 30 mg
Tel-OD 80 mg
Zyrova 5mg
Manflow 0.4mg
Glycomet (SR) 500 bd
SITA 100 mg
Ecom 75 mg
Once A Day morning
3/12

181, Kirula Road, Colombo 05, Sri Lanka.
Tel : 4523300 Fax : 2508768 E-mail : info@asiri.lk

Dr B. Who
Farmstreet 12
Kirkville
tel. 3876

R/ date 1 nov 1994

Tolbutamide 1000 mg
tab. da no. 30
S. 1 dd 1 tabl. before breakf.

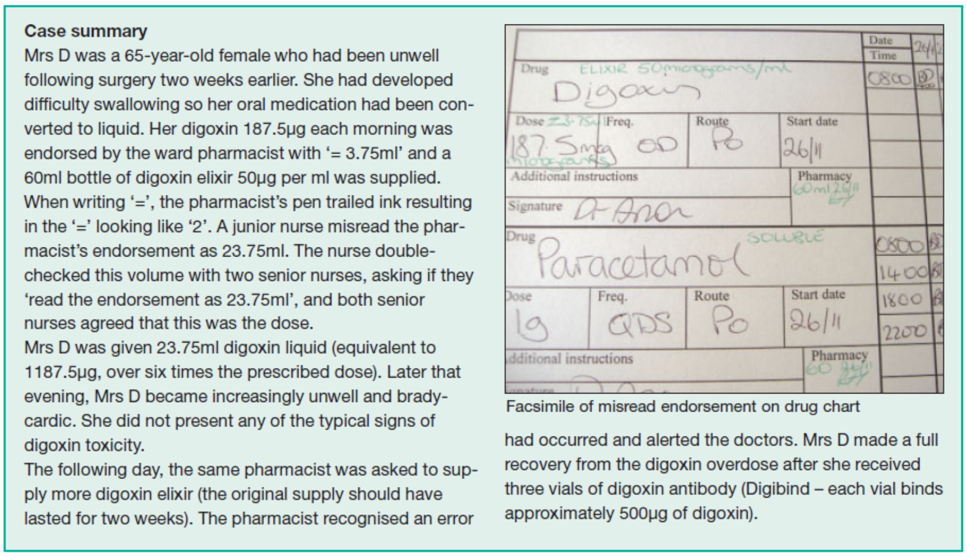
Rp morphine ten milligram
tablet da no. thirty-five
S. one tablet every six hours,
two before the night
(maximum 5 daily)

Ms/Mr
address: Patient 32 B. Who
age:

1. [illegible] 40mg PO x 1 stat
2. [illegible] MD KID
3. [illegible] Flow to [illegible]
4. [illegible] 94% [illegible]
5. [illegible] > 75% [illegible]

Fig. 1.1: Sample Prescriptions

Because of these differences, prescription identification are prone to errors and mis-identifications. Identifying either the drug, dosage or the duration wrong may lead to some serious problems regarding the patient's health and may even lead to deaths. One of such incident is described in the case summary below.

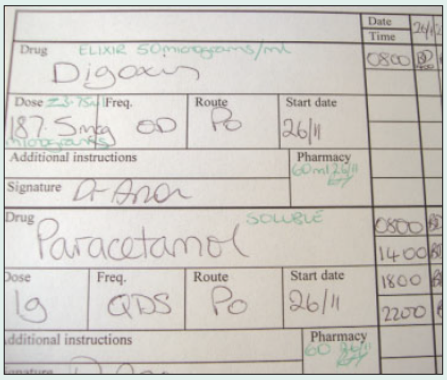


Case summary

Mrs D was a 65-year-old female who had been unwell following surgery two weeks earlier. She had developed difficulty swallowing so her oral medication had been converted to liquid. Her digoxin 187.5µg each morning was endorsed by the ward pharmacist with '= 3.75ml' and a 60ml bottle of digoxin elixir 50µg per ml was supplied. When writing '=', the pharmacist's pen trailed ink resulting in the '=' looking like '2'. A junior nurse misread the pharmacist's endorsement as 23.75ml. The nurse double-checked this volume with two senior nurses, asking if they 'read the endorsement as 23.75ml', and both senior nurses agreed that this was the dose.

Mrs D was given 23.75ml digoxin liquid (equivalent to 1187.5µg, over six times the prescribed dose). Later that evening, Mrs D became increasingly unwell and bradycardic. She did not present any of the typical signs of digoxin toxicity.

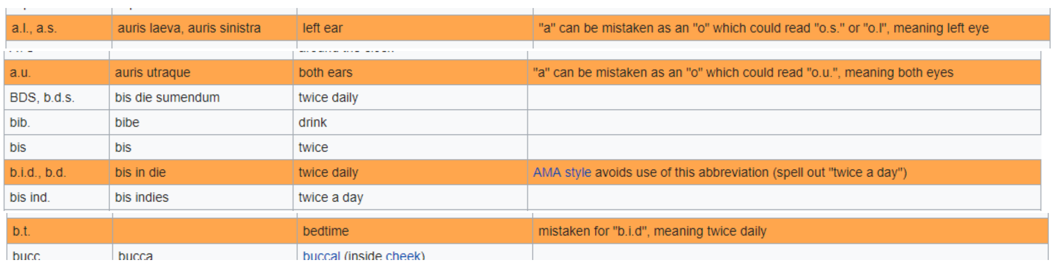
The following day, the same pharmacist was asked to supply more digoxin elixir (the original supply should have lasted for two weeks). The pharmacist recognised an error had occurred and alerted the doctors. Mrs D made a full recovery from the digoxin overdose after she received three vials of digoxin antibody (Digibind – each vial binds approximately 500µg of digoxin).



Facsimile of misread endorsement on drug chart

Fig. 1.2: Case Summary

There are so many other instances where patients are given the wrong drug, wrong dosage or in a wrong duration. When writing prescriptions, there is a certain set of abbreviations used by the doctors. The wikipedia article "List of abbreviations used in medical prescriptions" list this set of abbreviations. In this list also they have mentioned the possible misreadings. Fig 3 shows few of them.



a.l., a.s.	auris laeva, auris sinistra	left ear	"a" can be mistaken as an "o" which could read "o.s." or "o.l", meaning left eye
a.u.	auris utraque	both ears	"a" can be mistaken as an "o" which could read "o.u.", meaning both eyes
BDS, b.d.s.	bis die sumendum	twice daily	
bib.	bibe	drink	
bis	bis	twice	
b.i.d., b.d.	bis in die	twice daily	AMA style avoids use of this abbreviation (spell out "twice a day")
bis ind.	bis indies	twice a day	
b.t.		bedtime	mistaken for "b.i.d", meaning twice daily
bucc.	bucca	buccal (inside cheek)	

Fig. 1.3: Abbreviations that are possible for being misread

Together with the illegible handwriting, reading prescriptions is also subjected to human errors. A study was conducted in National District Hospital, Bloemfontein to identify who can read the prescriptions best among doctors, nurses and pharmacists. The expected outcome of this research was that the pharmacists should be able to read the prescription best. But the actual outcome was unfavorable and it showed that the pharmacists are the party who make most of the mistakes in reading prescription.

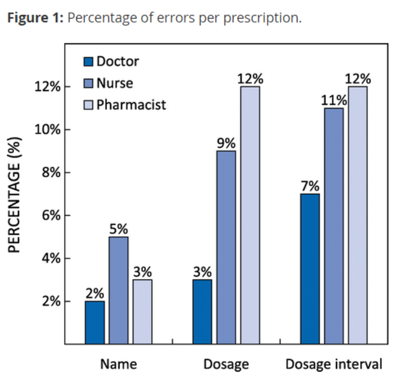


Fig. 1.4: Percentage of errors per prescription

The above study also states that the Doctors' sloppy handwriting kills more than 7000 people annually. And also preventable medication errors affect more than 1.5 million Americans annually because of the unclear abbreviations and doses and illegible handwriting.

There is an algorithm that can be used to read prescriptions which is currently performed by the doctors, nurses and pharmacists. But up until now there had been no studies performed to automate this algorithm so that the public can use it to read the prescriptions. That is the main motivation behind this research.

1.2 Research Problem and Research Questions

It is hard to read doctor's prescription for normal people who does not have any knowledge on the drugs. Even or people who have domain knowledge it is not possible to read illegible prescriptions. So because of similar looking drug names, similar usage directions and dosages, misreading of prescriptions can cause critical health problems, even death. The main concern in this research is those problems that are caused by misreading medical prescriptions and finding ways to minimize those problems.

1.2.1 Research Question

- How to identify the content of a handwritten medical prescription using the image of the prescription

1.2.2 Research Aim

The goal of this project is to identify a methodology for the computers to properly identify the content of a handwritten medical prescription. The aim is to identify and automate the current identification methods that are used in practice by doctors and pharmacists

1.2.3 Research Objectives

- Review a literature to identify different approaches, tools and technologies used in image binarization, intelligent classification systems, character recognition systems and domain specific character recognition.
- Apply a suitable technology to preprocess the images of prescriptions
- Use a suitable intelligent classification system to identify and classify the content of the prescription.
- Train and evaluate the classification.
- Publish a research paper on "Medical Prescription Identification Solution"

1.3 Justification for the research

Properly identify the content of a medical prescription is crucial to the preservation of life of the recipient of the treatment as well as to a proper recovery.

When it comes to optical character recognition (OCR), there are many existing methods classified into online, offline as well as handwritten and typed. But when in comes to the medical prescriptions, the recognition method should be a offline handwritten character recognition method. Even though it would be better to identify the content real-time using online methods, as the foundation, developing an offline handwritten recognition scheme must be explored.

1.4 Methodolody

This research is to find a method to accuralty identify the content of a medical prescription. Input to the system will be a colored image of the prescription. The preprocessing step will take place in order to make the image ready for the recognition process. In the preprocssing step, first the image will be segmented into single lines. These segmented imaged will then be resize to cater to the requirement of the recongition system which in the dimension of 128x32px. Then the resized image will be converted into gray scale.

The preprocessed image will then be input to the recognition system. First a CNN is used to extract the relevent feature of the input image. The extracted features will then be input to the RNN to identify the content of the image. Output of this phase will be a word which is recognised by the RNN.

The recongnized word will then be compared against the knowledge base inorder to find the mose suitable drug name. The output of the entire system would be the most matching drug name of the input image.

1.5 Outline of the Disseration

The remainder of the document is structured as explained in this section. This section also contains details of each of the chapters that make the remainder of the document.

Chapter 2 is the Literature Review which contains the review of the related work. This review was conducted with the purpose of finding any similar researches conducted in the area of interest, identify the strength and weaknesses of the existing methods, gaps in the available body of work and to identify ways to improve the existing methods. It also helped to finalize the proposing research design.

Chapter 3 contain the Design of the research. High level architecture of the research is dicussed in detail in this chapter which is followed by details of the subsystems as well.

Chapter 4 contains the details of the Implementation of the proposed solution. This chapter includes details on the technologies that are used for the proposing method as well as details on the implementation.

Chapter 5 discuss the results and the evaluation.

Chapter 6 contains the conclution of the research carried out.

1.6 Definitions

Through out this document the terms *prescription* will be used to refer to the medical prescription written by the doctor, *network* will be used to refer to the classification neural network. *Image* will refer to the preprocessed image of a single line of the prescription which will be in a predefined size. The term *knowlege base* will refer to the database of drug names, list of abbreviations and list of other prescription details.

Any further definitions related to the project wil be discussed under the regarding topic.

1.7 Delimitation of the Scope

Since the range of drugs that are in use are wide and their combination for illnesses can be even wider than the number of drugs, the scope of this research is limited to few common illnesses like

- Diabetes
- Cholesterol
- High blood pressure

Even in this much limited scope, this research will have a bit of a complexity when in comes to the classification system.

Usually prescriptions have details of several prescribed drugs. But the proposed method only identifies individual lines of the prescription, which will contain only one drug and it's prescription information.

In the identification phase, gray-scaled image will be input to the network and the output will be the recognized word in the image. It is then compared with the drug names to get the best match and identify the drug. Writing pieces characters which cannot be used to extract handwriting features will be considered as misclassified. Segments that have no best matching will be considered misclassified as well as the drug names that are not in the database.

The knowledge base will only have drug details for diabetes, cholesterol and high blood pressure. But the prescriptions may contain various other drugs that are intended for other diseases. Those will be misclassified in the system.

1.8 Conclusion

This concludes the introductory chapter of this dissertation. The intention of this chapter is to give the readers, a detailed idea of the background of the research and the area of interest. This chapter introduce the research problem, research question along with the aim and the objective of the research. Then the justification for the research is provided followed by a brief introduction of the methodolody. Given this, the dissertation will proceed with a detailed insight of the research.

2. CHAPTER 2 - LITERATURE REVIEW

2.1 Image Binarization

Throughout the years there were lots of research on proper ways to binarize images of documents without damaging the details.

In 1970 Nobuyuki Otsu proposed a method to select the threshold of the gray-level histograms in his paper “A Threshold Selection Method from Gray-Level Histograms”. The suggested method was described as a nonparametric and unsupervised method of automatic threshold selection for picture segmentation. The paper presented a new method is proposed from the viewpoint of discriminant analysis; it directly approaches the feasibility of evaluating the ”goodness” of threshold and automatically selecting an optimal threshold. The proposed method was to calculate the intra-class variance and the inter-class variance and then select the threshold such that intra-class variance is minimal and the inter-class variance is maximal.

Then in 1997 Sauvola et al. proposed another adaptive document binarization method. Document image understanding methods require logical and semantic content preservation during thresholding. For example, a letter connectivity must be maintained for optical character recognition and textual compression. This requirement narrows down the use of a global threshold in many cases. There are many important points that we need to keep in mind when it comes to document image binarization. First, a document image usually contains many regions with differing structure and semantic content, for example picture, text, background and line drawing. Therefore, specialized methods are needed to analyze the various types of regions. Second, the state and degree of a degradation can vary significantly within a document image due to various sources of error, such as scanning, copying and poor source material (paper and print quality etc.). Their proposed method was as follows.The grey-level document image is first analyzed to determine the surface properties. Then, according to analysis information the recognized surface properties are treated with two different binarization methods. For background and ‘scene’ type areas an algorithm utilizing the soft decision method in a new contextis performed. For textual and badly illuminated regions a histogram method is applied. Our algorithm was tested with several severe cases of degradations, natural and synthetic. The test results show that the algorithm adapts well to even severe degradations, enhancing for example the OCR rate in badly degraded images from non-recognizable to correct or near-correct results. Furthermore the algorithm’s custom parametrization is minimized with the use of soft decision methods and special analysis procedures. The main features of their approach include locally adaptive threshold selection, information content preserving analysis, and seamless applicability to various types of documents.

In 2000 again Sauvola et al. proposed another method for adaptive document image binarization. Here for adaptive document image binarization, where the page is considered as a collection of subcomponents such as text, background and picture. The problems caused by noise, illumination and many source type-related degradations are addressed. Two new algorithms are applied to determine a local threshold for each pixel. Their method uses two algorithms to binarized textual and non-textual regions. Comparison was carried out between the proposing method, Eikvil, Niblack, Bernsen and Parker's methods. Researchers were able to prove that the proposed method performs better than the other 4 binarization methods. Their method includes region analysis and switching, binarization of non-textual components using weighted bound calculation, transient difference calculation, membership function generation and soft decision rules and defuzzification, binarization of textual components and finally interpolative threshold selection. Key features of this research are using hybrid approach and taking document region class properties into consideration and aimed at generic document types coping also with severe cases of different types of degradation.

After that in 2005 Gatos et al. proposed a new method in their paper "Adaptive degraded document image binarization". It was new adaptive approach for the binarization and enhancement of degraded documents The proposed method does not require any parameter tuning by the user and can deal with degradations which occur due to shadows, non-uniform illumination, low contrast, large signal-dependent noise, smear and strain. Compares the proposed system with Otsu's method, Niblack's method, Sauvola et al. method, Kim et al. method. They were able to prove that the proposed method performs better than the other 4 comparing methods. Their methodology included pre-processing procedure using a low-pass Wiener filter, a rough estimation of foreground regions, a background surface calculation by interpolating neighboring background intensities, a thresholding by combining the calculated background surface with the original image while incorporating image up-sampling and finally a post-processing step in order to improve the quality of text regions and preserve stroke connectivity. As future work they have mentioned that further research will focus on the challenges that emerge from the binarization of low resolution images and videos found on the Web.

A double-threshold image binarization method based on edge detector was proposed in 2007 by Chena et al. the proposed method was stated to be effective on the binarization of images with low contrast, noise and non-uniform illumination. The proposed methodology is as follows. Generate the edge image by using the Canny edge detector, Determine seeds, low and high thresholds, edge connection, binarize the image with the high threshold, seed filling, combine the low-threshold binary image, remove noising and generate result image. Compares the results with Otsu's method, Bernsen's method, Niblack's method, Yanowitz–Bruckstein's method, Sauvola's method, Gatos's method. The test results show that the method has several good properties, such as less loss of object information, and high robustness, and has better overall performance than several classic binarization methods.

In 2012 another separate approach was proposed by Shaikh et al., A new image binarization method using iterative partitioning. It is said to be a new method for image binarization that uses an iterative partitioning approach, especially for degraded documents and graphic images and suitable for a multi-core processing environment as it can be split into multiple parallel units of executions after the initial partitioning. They have compared the proposing technique with other 4 major binarization techniques and have proved that the proposing technique perform well than the other methods by presenting the evaluation criteria and error percentages. When is comes to the methodology, they uses otsu method for selecting the threshold. first compute the sharp peaks. if only 2 peaks, binarization using otsu, otherwise partition the image into 4 sub images, compute sharp peaks in each. If 2 peaks binarization using otsu. Otherwise partition image again and repeat for every sub image until the minimum subimage size is reached. This method perform very well compared to otsu, niblack, sauvola and bernsen.

2.2 Domain Specific Handwritten Character Recognition

Throughout the years, there have been many attempts on finding a proper way to make the machines read human handwriting. Among those researches there are general character recognition researches as well as domain specific character recognition researches.

Rajavelu et al. has proposed the neural network approach for character recognition in 1989 in [7]. Their purpose was to develop an algorithm which effectively reduces image processing time while maintaining efficiency and versatility. Their method has optimal selection of features as well as parallel computational capability of neural network that ensures a high speed of recognition which is crucial for commercial application. They have used Walsh function in order to reduce the computational time. Their neural network has an input layer of 20 nodes, one hidden layer of 20 nodes, output layer of 7 nodes and the learning was done using backpropagation. They were able to gain a successful recognition rate of over 98%.

Pradeep et al. has proposed a diagonal based feature extraction for handwritten character recognition system using neural networks in [1]. Purpose of this research is to design a system for off-line handwritten alphabetical character recognition using multilayer feedforward neural network. They have used a neural network with an input layer of 54/69 nodes, two hidden layers of 100 nodes each, an output layer of 26 nodes and log sigmoid function as the activation function. The extracted features from the preprocessing was input to the neural network which did the classification and the recognition. They were able to achieve a good recognition accuracy of 96.52% with 54 features and 97.84% with 69 features.

When it comes to the domain specific character recognition most of the character recognition researches were done on mail delivery systems where the focus of the research is to successfully identify the handwritten address/ZIP code on the envelope.

Jonathan et al. has proposed a blackboard based approach to recognise handwritten ZIP codes in his paper [4] in 1988. For character recognition the blackboard model provides flexibility, a coordinated method of integrating different knowledge sources, and a means of hierarchical data organization. Given a high-resolution image of a hand-written address block, the solution invokes routines capable of hypothesizing the location of the ZIP Code, segmenting and recognizing ZIP Code digits, locating and recognizing City and State names, and looking-up the results in a dictionary. The main steps of the methodology was thresholding, testing for machine or handwritten text, horizontal line removal, text line segmentation, text word segmentation, ZIP Code location, ZIP Code segmentation, parallel digit recognition algorithm and ZIP Code dictionary lookup. As the results, 31% of the ZIP Codes were read correctly, 9.4% of the images did not contain a ZIP Code and were rejected. Thus, 40.4% of the pieces were correctly processed. An erroneous ZIP Code was assigned to 2.3% of the pieces and 57.3% contained a valid ZIP Code and were rejected.

Cun et al. has proposed a method to recognize handwritten zip codes with multilayer network [2]. Purpose of their research was to apply backpropagation network with minimal preprocessing of data. But the network is highly constrained because it is specifically designed for this task. They have use both handwritten and printed samples on the dataset where printed dataset contain more than 35 fonts and handwritten dataset is collected from the USPS. The feature extraction have used feature maps and these feature maps were input to the neural network. The network, as they describe is a combination of statistical and the structural model. The connection patterns were guided by the knowledge of the researchers, thus making the neural network highly constrained and specific for this research. The network had 4 hidden layers, two for shared-weight feature extraction and another two for averaging/subsampling. The learning was based on neocognitron architecture but instead of unsupervised learning they have used backpropagation learning.

Another approach for handwritten ZIP code recognition[3] was proposed by Dzuba et al. in 1997. It is a real-time system intended to recognize the 5-digit ZIP code part of DPC. The results of ZIP code recognition are cross-validated with those of city and state names recognition. This approach recognize words as a whole without preliminary letter identification which uses handwritten character features. Input to the system is a digitized image of an address block. The process includes line segmentation, CSZ block extraction, CSZ block segmentation, recognition and cross-validation of segmentation hypotheses. The output of the system is the ZIP code string with the confidence value. The system achieves a throughput of 40.000 address blocks per hour. With the performance of 73% recognition rate with 1.0% error rate.

In 1993, Kimura et al. presented a method suggesting improvements of a lexicon directed algorithm for recognition of unconstrained handwritten words [5] like cursive, discrete or mixed. Their method include image binarization using Otsu's algorithm, slant correction, calculate cumulative chain code histogram, feature extraction, calculate character likelihood, segmentation recognition and post processing. To improve the accuracy in the method, they have done an error analysis, pre segmentation which includes parameter optimization in valley point detection and single-run stretch splitter, word length estimation, improvement of character classifier with design sample size and refinement of feature extraction process and finally splitting cost function. Since the segmentation recognition are repeated for all lexicon words having the word length within the estimated wordlength interval, the processing time increases proportionally to the lexicon size .Daemons for feature extraction and character likelihood calculation were introduced to avoid unnecessary calculations. In total the results for 10, 100. and 1000 lexicon sizes were improved by 4.47%. 7.77% and 10.57% and amounted to 98.01% 95.46%. and 91.49% respectively.

There was another research for interpreting handwritten addresses in US mailstream by Sargur et al. [6]. Their methodology includes binarization, noise (postal marks underlines, etc.) removal, skew correction, address lines separation, identification of word categories (e.g. ZIP field, street number field), recognition of words and digit strings, and determining the destination point code (DPC) of the address. Word recognition was done using a new method that was built using both Hypothesis Generate and Reduce (HGR) paradigm and Hidden Markov Model (HMM). They were able to achieve 44% encode rate with ¡ 6% error rate.

In 2011 Cries et al. proposed a method, “Convolutional Neural Network Committees For Handwritten Character Classification”. At some stage in the classifier design process one usually has collected a set of reasonable classifiers. Typically one of them yields best performance. Intriguingly, however, the sets of patterns misclassified by different classifiers do not necessarily greatly overlap. Here they have focused on improving recognition rates using committees of neural networks. Their goal was to produce a group of classifiers whose errors on various parts of the training set differ as much as possible. Other approaches aiming at optimally combining neural networks do not do this, thus facing the problem of strongly correlated individual predictors. Furthermore, they simply average individual committee member outputs, instead of optimizing their combinations , which would cost additional valuable training data. Simple training data pre-processing gave them experts with errors less correlated than those of different nets trained on the same or bootstrapped data. Hence committees that simply average the expert outputs considerably improve recognition rates. Their committee-based classifiers of isolated handwritten characters were the first on par with human performance, and can be used as basic building blocks of any OCR system.

In 1988 Kunihiko Fukushima proposed ”Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition”. As has been shown here, the neocognitron has many remarkable properties which most modern computers and pattern-recognizers do not possess. Since the neocognitron can learn, it can be trained to recognize not only Arabic numerals, but also other sets of patterns, like letters of the alphabet, geometrical shapes, or others. Hence, it is possible to design a neocognitron as a universal pattern-recognizer, which can be used, after training, for an individual purpose. If the number of categories of the patterns to be recognized is increased, the number of cell-planes in each layer of the network also has to be increased. The number of cell-planes, however, need not be increased in proportion to the number of categories of the patterns. It is enough to increase it m less than linear proportion, because local features to be extracted at lower stages are usually contained in common in patterns of different categories. If we want to construct a system which can recognize more complex patterns like Chinese characters, it is recommended to increase the number of stages (or layers) in the network depending on the complexity of the patterns to be recognized. The principles of the neocognitron are not restricted to the processing of visual information only, but can also be applied to other sensory information. For example, it would be possible to construct a speech-recognition system with a little modification. Although the neocognitron has forward (i.e., afferent or bottom-up) connections only, the information-processing ability of the network can be greatly increased if backward (i.e., afferent or top-down) connections are added. The model of selective attention recently proposed by the author is an example of such an advanced system. We are continuing the research, and we hope to develop an artificial brain closer to the human brain.

In 1996 a survey was done in pattern recognition by Oivind et al. Their survey states its results as follows. “Optical character recognition (OCR) is one of the most successful applications of automatic pattern recognition. Since the mid 1950s, OCR has been a very active field for research and development. Today, reasonably good OCR packages can be bought for as little as $100. However, these are only able to recognize high quality printed text documents or neatly written hand-printed text. The current research in OCR is now addressing documents that are not welt handled by the available systems, including severely degraded, omnifont machine-printed text and (unconstrained) handwritten text. Also, efforts are being made to achieve lower substitution error rates and reject rates even on good quality machine-printed text, since an experienced human typist still has a much lower error rate, albeit at a slower speed. Selection of feature extraction method is probably the single most important factor in achieving high recognition performance. Given the large number of feature extraction methods reported in the literature, a newcomer to the field is faced with the following question: Which feature extraction method is the best for a given application? This question led us to characterize the available feature extraction methods, so that the most promising methods could be sorted out. An experimental evaluation of these few promising methods must still be performed to select the best method for a specific application. Devijver and Kittler define feature extraction as the problem of ”extracting from the raw data the information which is most relevant for classification purposes, in the sense of minimizing the within-class pattern variability while enhancing the between-class pattern variability”. In this paper, we reviewed feature extraction methods including,

- template matching
- deformable templates
- unitary image transforms
- graph description
- projection histograms
- contour profiles
- zoning
- geometric moment invariants
- Zernike moments
- spline curve approximation
- Fourier descriptors

Each of these methods may be applied to one or more of the following representation forms:

- gray-level character image
- binary character image
- character contour

- character skeleton or character graph.

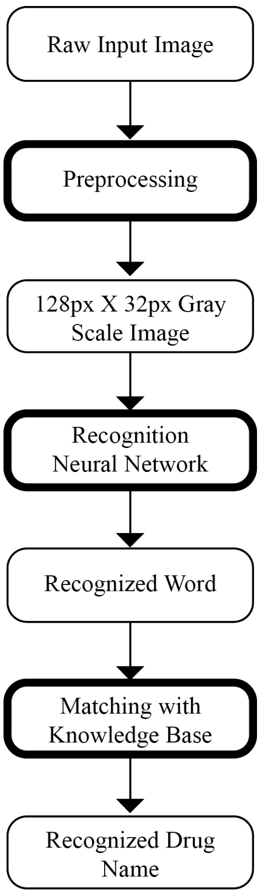
For each feature extraction method and each character representation form, we discussed the properties of the extracted features. Before selecting a specific feature extraction method, one needs to consider the total character recognition system in which it will operate. The process of identifying the best feature extraction method was illustrated by considering the digits in the hydrographic map as an example. It appears that Zernike moments would be good features in this application. However, one really needs to perform an experimental evaluation of a few of the most promising methods to decide which feature extraction method is the best in practice for each application. The evaluation should be performed on large data sets that are representative for the particular application.

3. CHAPTER 3 - RESEARCH DESIGN

This research is design to address the following.

- Identify the separate drugs in the prescription image
- Preprocess the image to make it suitable for the recognition system
- Recognize the characters in the input image
- Match the output of the recognition system with the knowledge base to get the exact word

3.1 A Conceptual Overview of the Project



Raw Input Image
Preprocessing
128px X 32px Gray Scale Image
Recognition Neural Network
Recognized Word
Matching with Knowledge Base
Recognized Drug Name

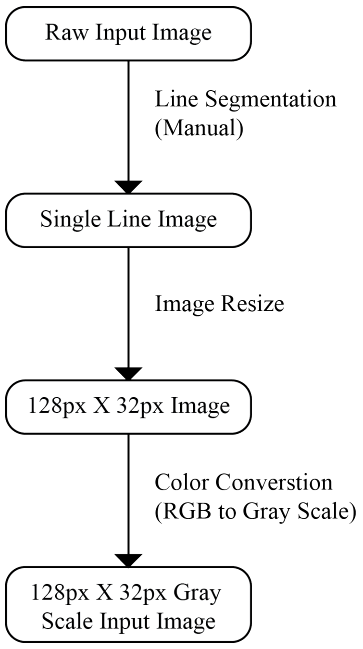
Fig. 3.1: High Level Architecture

When talking about the tasks that are mentioned above, a high level architecture of the system is shown in Figure 3.1. As shown in there, the complete system has three main sub systems.

- Preprocessing Subsystem
- Recognition Subsystem
- Knowledge Base Matching Subsystem

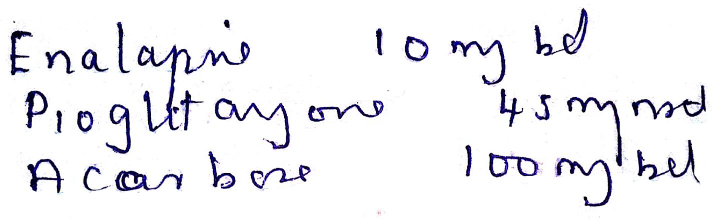
The raw input image is input to the preprocessing subsystem. In here, the image is processed to made suitable for the recognition neural network. The output of the preprocessing subsystem is 128 x 32px gray scale image which is then input into the recognition neural network. The recognition NN will identify the characters in the input image and the output will be the word recognized by the NN. This word is then matched with the knowledge base inorder to identify the relavent word with regard to the domain. Each subsystem will be discussed in details in the next section.

3.1.1 Preprocessing Subsystem



Raw Input Image
Line Segmentation (Manual)
Single Line Image
Image Resize
128px X 32px Image
Color Converstion (RGB to Gray Scale)
128px X 32px Gray Scale Input Image

Fig. 3.2: High Level Architecture of the Preprocessing Subsystem



Enalapril 10 mg bd
Pioglitazone 45 mg mod
Acarbose 100 mg bd

Fig. 3.3: Example Raw Input Images

High level architecture of the preprocessing subsystem is as of figure 3.2. Input for this subsystem is the raw image of the prescriptions. Sample images are shown in the Figure 3.3. Since there are several lines in a single prescription, each prescription needs to be segmented into individual lines before inputing to the recognition NN. This segmentation can be done using projection histograms, but the problem with prescriptions is that, the lines are not always perfectly horizontal. Two of such images are shown in Fig 3.4. So the segmentation is done manualy. Output of this segmentation is as in Fig 3.5.

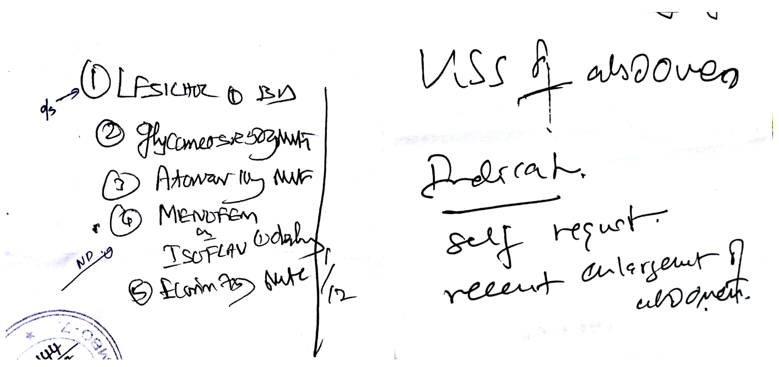


Fig. 3.4: Problametic Raw Input Images

Fig. 3.5: Segmented Images

Then these segmented images are resize to make the dimension 128 X 32px. When resizing, the image is scaled untile either the height becomes 32px or the width becomes 128px. When the either of these is achieved, the rest is filled with white pixels to get the final image. An example is shown in Fig 3.6.

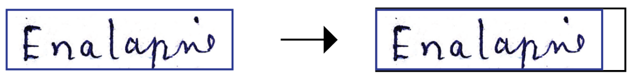


Fig. 3.6: Resizing the Image

Upto this point the images used are color images. But to feed into the recognition NN, the image needs to be in gray-scale. So after resizing the image, the color scheme of the image is converted to gray scale from RGB. Finally, the output of this preprocessing subsystem will be a gray-scaled, single line prescription image with the dimension of 128 X 32px. (Fig 3.7)

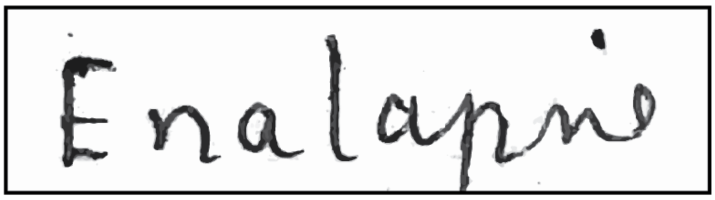


Fig. 3.7: Output of Preprocessing Subsystem

3.1.2 Recognition Subsystem

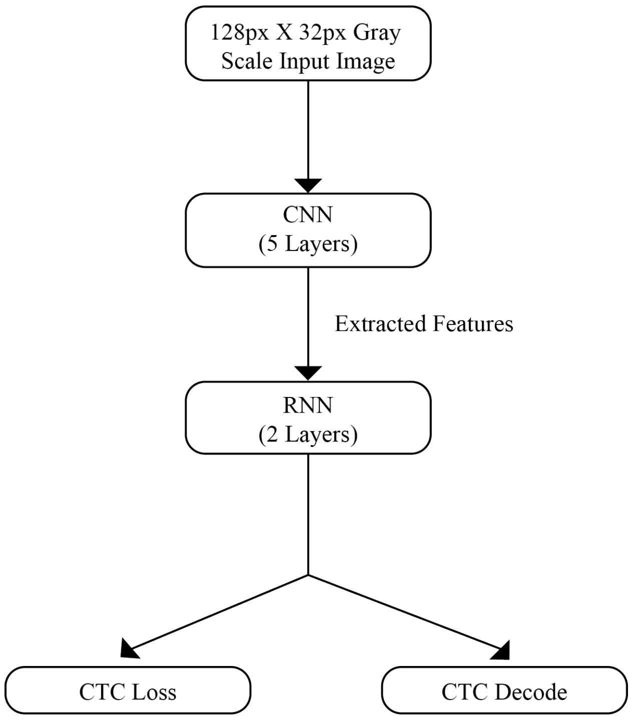


Fig. 3.8: High Level Architecture of the Recognition Subsystem

As shown in Fig 3.8, the recognition subsystem uses a NN for the task. This system consists of Convolutional NN (CNN) layers, Recurrent NN (RNN) layers and a final Connectionist Temporal Classification (CTC) layer. In a more formal way, the NN can also be represented as a function (Fig 3.9) which maps an image (or matrix) M of size WxH to a character sequence (c1, c2, ...) with a length between 0 and L. As you can see, the text is recognized on character-level, therefore words or texts not contained in the training data can be recognized too (as long as the individual characters get correctly classified).

$$\mathrm{NN}: \underset{W \times H}{M} \rightarrow \underset{0 \leq n \leq L}{(c_1, c_2, \ldots, c_n)}$$

Fig. 3.9: The NN written as a mathematical function which maps an image M to a character sequence (c1, c2, ...)

Convolutional Neural Network

The input image is fed into the CNN layers. These layers are trained to extract relevant features from the image. Each layer consists of three operation. First, the convolution operation, which applies a filter kernel of size 5x5 in the first two layers and 3x3 in the last three layers to the input. Then, the non-linear RELU function is applied. Finally, a pooling layer summarizes image regions and outputs a downsized version of the input. While the image height is downsized by 2 in each layer, feature maps (channels) are added, so that the output feature map (or sequence) has a size of 32x256

Recurrent Neural Network

The feature sequence contains 256 features per time-step, the RNN propagates relevant information through this sequence. The popular Long Short-Term Memory (LSTM) implementation of RNNs is used, as it is able to propagate information through longer distances and provides more robust training-characteristics than vanilla RNN. The RNN output sequence is mapped to a matrix of size 32x80. The IAM dataset consists of 79 different characters, further one additional character is needed for the CTC operation (CTC blank label), therefore there are 80 entries for each of the 32 time-steps.

Connectionist Temporal Classification

While training the NN, the CTC is given the RNN output matrix and the ground truth text and it computes the loss value. While inferring, the CTC is only given the matrix and it decodes it into the final text. Both the ground truth text and the recognized text can be at most 32 characters long.

3.1.3 Knowledge Base Matching Subsystem

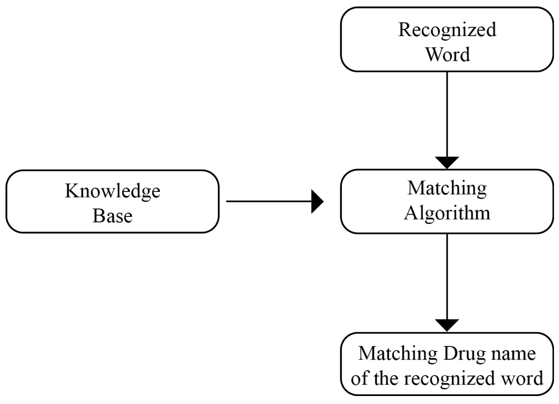


Fig. 3.10: High level Architecture of the Knowledge Base Matching Subsystem

As shown in Fig 3.10, the input to this subsystem is the recognized word which is the output of the recognition NN. The matching algorithm will then compare the recognized word against the Knowledge base to get the most suitable drug name. Output of this subsystem as well as the output of the entire system will be the most suitable drug name.

3.2 Knowledge Base

The knowledge base of this system contains drug names, both the generic names and the brand names of the drugs that are used as treatments to Diabetes, High Blood Pressure and Cholesterol. The size of the knowledge base is as follows.

- Diabetes : 166 entries
- High Blood Pressure : 200 entries
- Cholesterol : 48 entries

3.3 Evaluation Design

When it comes to the evaluation phase of the research, expert help was needed since it's the medical domain. The segmented single line prescription images were labeled by the domain advisors as the first step. After the entire process of recognition is completed, as the evaluation phase, what was left to do was to compare the output of the system with the label of the corresponding segmented single line image.

3.4 Chapter Summary

With this chapter the conceptual overview of the project has been discussed followed by a brief discussion about the architecture of the subsystems, knowledge base and the evaluation design. Implementation details of the mentioned process will be then discussed in the next chapter, Chapter 4.

4. IMPLEMENTATION

4.1 Technologies Used

4.1.1 Adobe Photoshop CC 2015

Since the segmentation of the prescription image could not be done using programtic methods, to segment the lines of the prescription image, Adobe Photoshop CC 2015 was used. While segmenting the image, the drastic noice areas were also removed inorder to make the best out of the recognition NN. Areas that are considered to be noisy were the areas with stains or any other degraded areas.

When talking about tool, Photoshop CC 2015 was released on June 15, 2015. Adobe added various creative features including Adobe Stock, which is a library of custom stock images. It also includes and have the ability to have more than one layer style. For example, in the older versions of Photoshop, only one shadow could be used for a layer but in CC 2015, up to ten are available. Other minor features like Export As, which is a form of the Save For Web in CC 2014 were also added. The updated UI as of November 30, 2015 delivers a cleaner and more consistent look throughout Photoshop, and the user can quickly perform common tasks using a new set of gestures on touch-enabled devices like Microsoft Surface Pro.CC 2015 also marks the 25th anniversary of Photoshop.

4.1.2 Python 3.5

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming. When it comes to NN and image processing, python have many supporting libraries to integrate required technologies with the project.

OpenCV2

OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license. OpenCV supports the deep learning frameworks TensorFlow, Torch/PyTorch and Caffe. In this project, OpenCV is used in combination with python to resize the input image and for the color conversion.

TensorFlow

TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google. TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015. Tensorflow integrated with python is used to develop the recognition NN, train and validate the NN and to test with the whole system in this project.

4.2 Training Dataset - IAM Handwriting Database

The IAM Handwriting Database contains forms of handwritten English text which can be used to train and test handwritten text recognizers and to perform writer identification and verification experiments. The database was first published in at the ICDAR 1999. Using this database an HMM based recognition system for handwritten sentences was developed and published in at the ICPR 2000. The segmentation scheme used in the second version of the database is documented in and has been published in the ICPR 2002. The IAM-database as of October 2002 is described in . We use the database extensively in our own research, see publications for further details. The database contains forms of unconstrained handwritten text, which were scanned at a resolution of 300dpi and saved as PNG images with 256 gray levels. The figure below provides samples of a complete form, a text line and some extracted words. All forms and also all extracted text lines, words and sentences are available for download as PNG files, with corresponding XML meta-information included into the image files. All texts in the IAM database are built using sentences provided by the LOB Corpus .

Characteristics The IAM Handwriting Database 3.0 is structured as follows:

- 657 writers contributed samples of their handwriting
- 1 539 pages of scanned text
- 5 685 isolated and labeled sentences
- 13 353 isolated and labeled text lines
- 115 320 isolated and labeled words

The words have been extracted from pages of scanned text using an automatic segmentation scheme and were verified manually. The segmentation scheme has been developed at our institute. All form, line and word images are provided as PNG files and the corresponding form label files, including segmentation information and variety of estimated parameters, are included in the image files as meta-information in XML format which is described in XML file and XML file format (DTD).

4.3 Implementation

4.3.1 Data

Input

It is a gray-value image of size 128x32. Usually, the images from the dataset do not have exactly this size, therefore image is resized (without distortion) until it either has a width of 128 or a height of 32. Then, image is copied into a (white) target image of size 128x32. This process is shown in the previous chapter. Finally, the gray-values of the image are normalize which simplifies the task for the NN. Data augmentation can easily be integrated by copying the image to random positions instead of aligning it to the left or by randomly resizing the image.

CNN Output

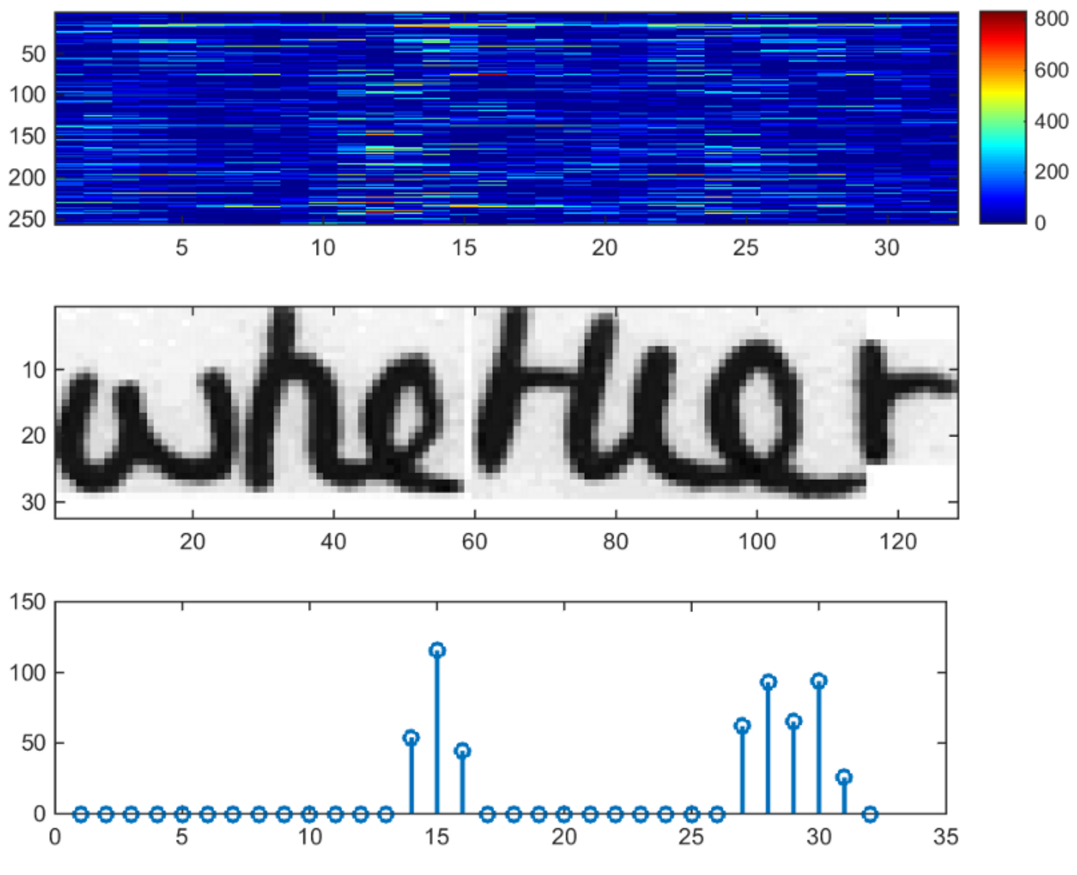


Fig. 4.1: Features in the input image

Fig. 4.1 shows the output of the CNN layers which is a sequence of length 32. Top image shows 256 feature per time-step are computed by the CNN layers while the middle image shows the input image and the bottom image plot of the 32nd feature, which has a high correlation with the occurrence of the character "e" in the image. Each entry contains 256 features. Of course, these features are further processed by the RNN layers, however, some features already show a high correlation with certain high-level properties of the input image: there are features which have a high correlation with characters (e.g. “e”), or with duplicate characters (e.g. “tt”), or with character-properties such as loops (as contained in handwritten “l”s or “e”s).

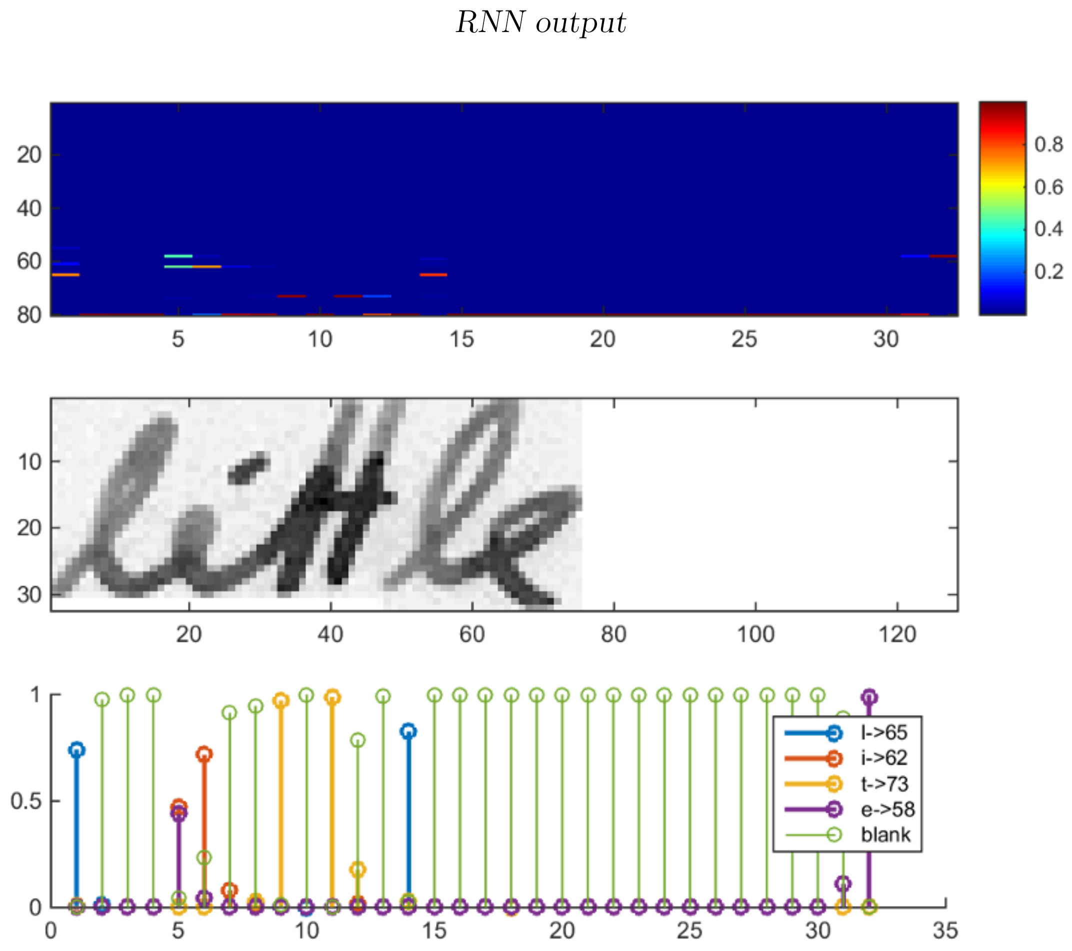


Fig. 4.2: Top: output matrix of the RNN layers. Middle: input image. Bottom: Probabilities for the characters “l”, “i”, “t”, “e” and the CTC blank label.

Fig. 4.2 shows a visualization of the RNN output matrix for an image containing the text “little”. The matrix shown in the top-most graph contains the scores for the characters including the CTC blank label as its last (80th) entry. The other matrix-entries, from top to bottom, correspond to the following characters: “ !”#&’()*+,-./0123456789:;?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz”. It can be seen that most of the time, the characters are predicted exactly at the position they appear in the image (e.g. compare the position of the “i” in the image and in the graph). Only the last character “e” is not aligned. But this is OK, as the CTC operation is segmentation-free and does not care about absolute positions. From the bottom-most graph showing the scores for the characters “l”, “i”, “t”, “e” and the CTC blank label, the text can easily be decoded: just take the most probable character from each time-step, this forms the so called best path, then throw away repeated characters and finally all blanks: “l—ii–t-t–l-. . . -e” -¿ “l—i–t-t–l-. . . -e” -¿ “little”.

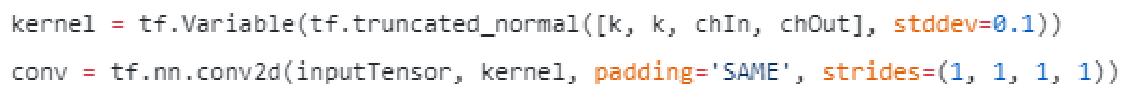
4.3.2 Using TensorFlow

The implementation consists of 4 modules:

- SamplePreprocessor.py: prepares the images from the IAM dataset for the NN
- DataLoader.py: reads samples, puts them into batches and provides an iterator-interface to go through the data
- Model.py: creates the model as described above, loads and saves models, manages the TF sessions and provides an interface for training and inference
- main.py: puts all previously mentioned modules together and compare the output with the knowledge base.

CNN

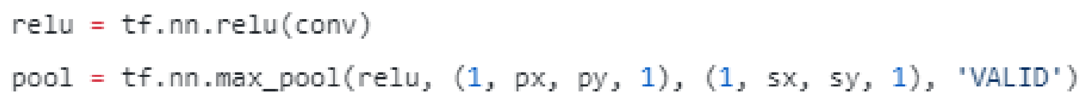
For each CNN layer, create a kernel of size kxk to be used in the convolution operation.



```
kernel = tf.Variable(tf.truncated_normal([k, k, chIn, chOut], stddev=0.1))
conv = tf.nn.conv2d(inputTensor, kernel, padding='SAME', strides=(1, 1, 1, 1))
```

Fig. 4.3: CNN Kernel

Then, feed the result of the convolution into the RELU operation and then again to the pooling layer with size pxxpy and step-size sxxsy.



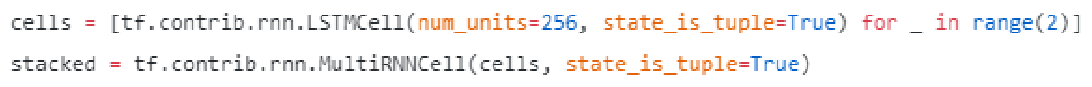
```
relu = tf.nn.relu(conv)
pool = tf.nn.max_pool(relu, (1, px, py, 1), (1, sx, sy, 1), 'VALID')
```

Fig. 4.4: CNN Relu and Pool

These steps are repeated for all layers in a for-loop.

RNN

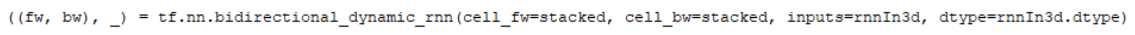
Create and stack two RNN layers with 256 units each.



```
cells = [tf.contrib.rnn.LSTMCell(num_units=256, state_is_tuple=True) for _ in range(2)]
stacked = tf.contrib.rnn.MultiRNNCell(cells, state_is_tuple=True)
```

Fig. 4.5: Creating RNN

Then, create a bidirectional RNN from it, such that the input sequence is traversed from front to back and the other way round. As a result, we get two output sequences fw and bw of size 32x256, which we later concatenate along the feature-axis to form a sequence of size 32x512. Finally, it is mapped to the output sequence (or matrix) of size 32x80 which is fed into the CTC layer.



```
((fw, bw), _) = tf.nn.bidirectional_dynamic_rnn(cell_fw=stacked, cell_bw=stacked, inputs=rnnIn3d, dtype=rnnIn3d.dtype)
```

Fig. 4.6: Bidirectional

CTC

For loss calculation, we feed both the ground truth text and the matrix to the operation. The ground truth text is encoded as a sparse tensor. The length of the input sequences must be passed to both CTC operations.

```
self.gtTexts = tf.SparseTensor(tf.placeholder(tf.int64, shape=[None, 2]) ,
                               tf.placeholder(tf.int32, [None]), tf.placeholder(tf.int64, [2]))
self.seqLen = tf.placeholder(tf.int32, [None])
```

Fig. 4.7: Setting up

We now have all the input data to create the loss operation and the decoding operation.

```
loss = tf.nn.ctc_loss(labels=gtTexts, inputs=inputTensor, sequence_length=seqLen, ctc_merge_repeated=True)
decoder = tf.nn.ctc_greedy_decoder(inputs=inputTensor, sequence_length=seqLen)
```

Fig. 4.8: CTC Loss and Decoder

Training

The mean of the loss values of the batch elements is used to train the NN: it is fed into an optimizer such as RMSProp.

```
optimizer = tf.train.RMSPropOptimizer(0.001).minimize(loss)
```

Fig. 4.9: Training NN

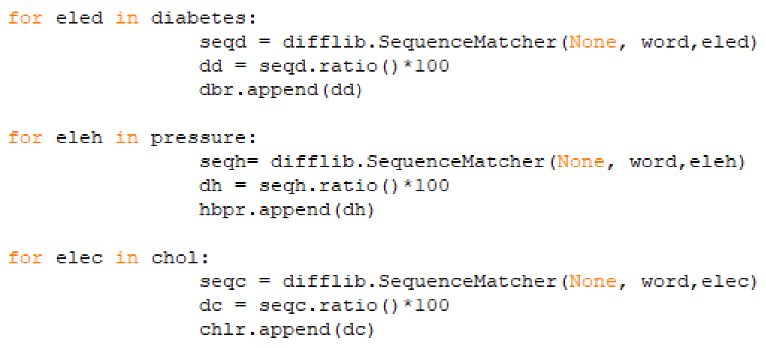
4.3.3 Matching with Knowledge Base

After the recognition NN, the output will be the recognized word. To match it against the knowledge base, first the knowledge base is loaded to seperate lists using the csv files.

```
dbcol =('d')
hbpcol =('p')
chlcol =('c')
db = pandas.read_csv('Hypo-Hyperglycemia.csv',names=dbcol)
hbp = pandas.read_csv('Hypertension.csv',names=hbpcol)
chl = pandas.read_csv('Hyperlipidemia.csv',names=chlcol)
diabetes = db.d.tolist()
pressure = hbp.p.tolist()
chol = chl.c.tolist()
```

Fig. 4.10: Loading the Knowledge Base

After the knowledge base is loaded, the recognized word is match with all the entries in the knowledge base and a matching ration is calculated for each separate lists.



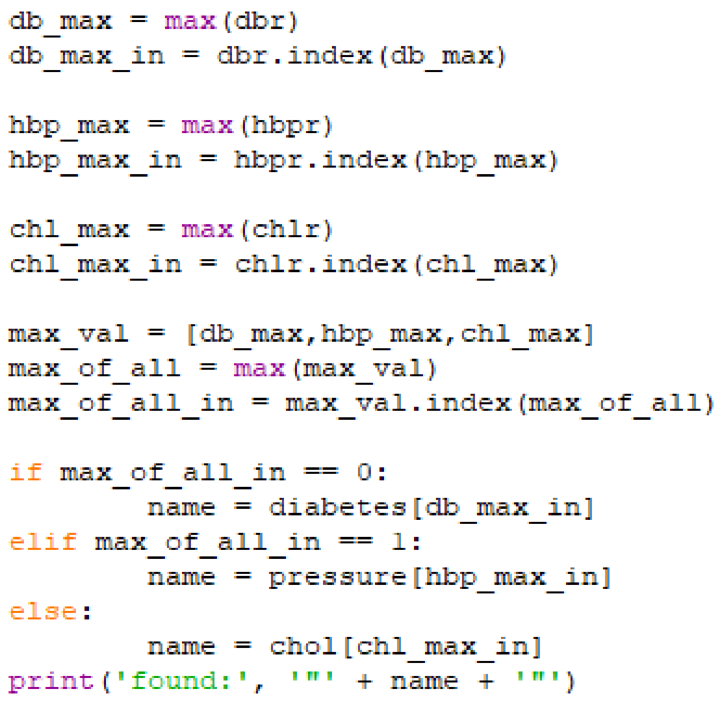
```
for eled in diabetes:
                seqd = difflib.SequenceMatcher(None, word,eled)
                dd = seqd.ratio()*100
                dbr.append(dd)

for eleh in pressure:
                seqh= difflib.SequenceMatcher(None, word,eleh)
                dh = seqh.ratio()*100
                hbpr.append(dh)

for elec in chol:
                seqc = difflib.SequenceMatcher(None, word,elec)
                dc = seqc.ratio()*100
                chlr.append(dc)
```

Fig. 4.11: Calculate the matching ratio

After the ratio is calculated as stored, maximum value from the each three lists will be obtained since the relevent illness of the corresponding drug is not known in advance. Among the three maxumum values, again the maximum of all is calculated to obtain the most suitable drug name. After the value is obtained, the corresponding drug name is retrieved from the list and output as the final result.



```
db_max = max(dbr)
db_max_in = dbr.index(db_max)

hbp_max = max(hbpr)
hbp_max_in = hbpr.index(hbp_max)

chl_max = max(chlr)
chl_max_in = chlr.index(chl_max)

max_val = [db_max,hbp_max,chl_max]
max_of_all = max(max_val)
max_of_all_in = max_val.index(max_of_all)

if max_of_all_in == 0:
        name = diabetes[db_max_in]
elif max_of_all_in == 1:
        name = pressure[hbp_max_in]
else:
        name = chol[chl_max_in]
print('found:', '"' + name + '"')
```

Fig. 4.12: Obtaining the maximum value and the resulting drug name

4.4 Chapter Summary

This chapter provides the detailed description of the implenetation of the project. A brief description is provided on the technologies used which is followed by a detailed description of the implementation of each of the three subsystems. Results and the evaluation will be discussed in the next chapter.

5. RESULTS AND EVALUATION

5.1 Evaluation Dataset

The recognition network was trained using the IAM Handwritten Character dataset and evaluation using the prescription images dataset. The prescription images dataset consists of the following components.

- 176 Prescription Images
- 412 Segmented Single line images.

5.2 Final Evaluation

Since the evaluation involves external parties, specially domain advisors, evaluation was planned with their availability and feasibility in mind. So as the fisrt step, all the dataset or the set of segmented images are labeled with the advisor's help. After that the preprocessing is done and the recognition process is carried on as usual. As the final evaluation, the labels was compared with the output of the recognition process for each image to evaluate whether the recognition was successfull or not. Following section contains the first 50 images of evaluation dataset and the results.

5.2.1 *Segmented Single Line images*

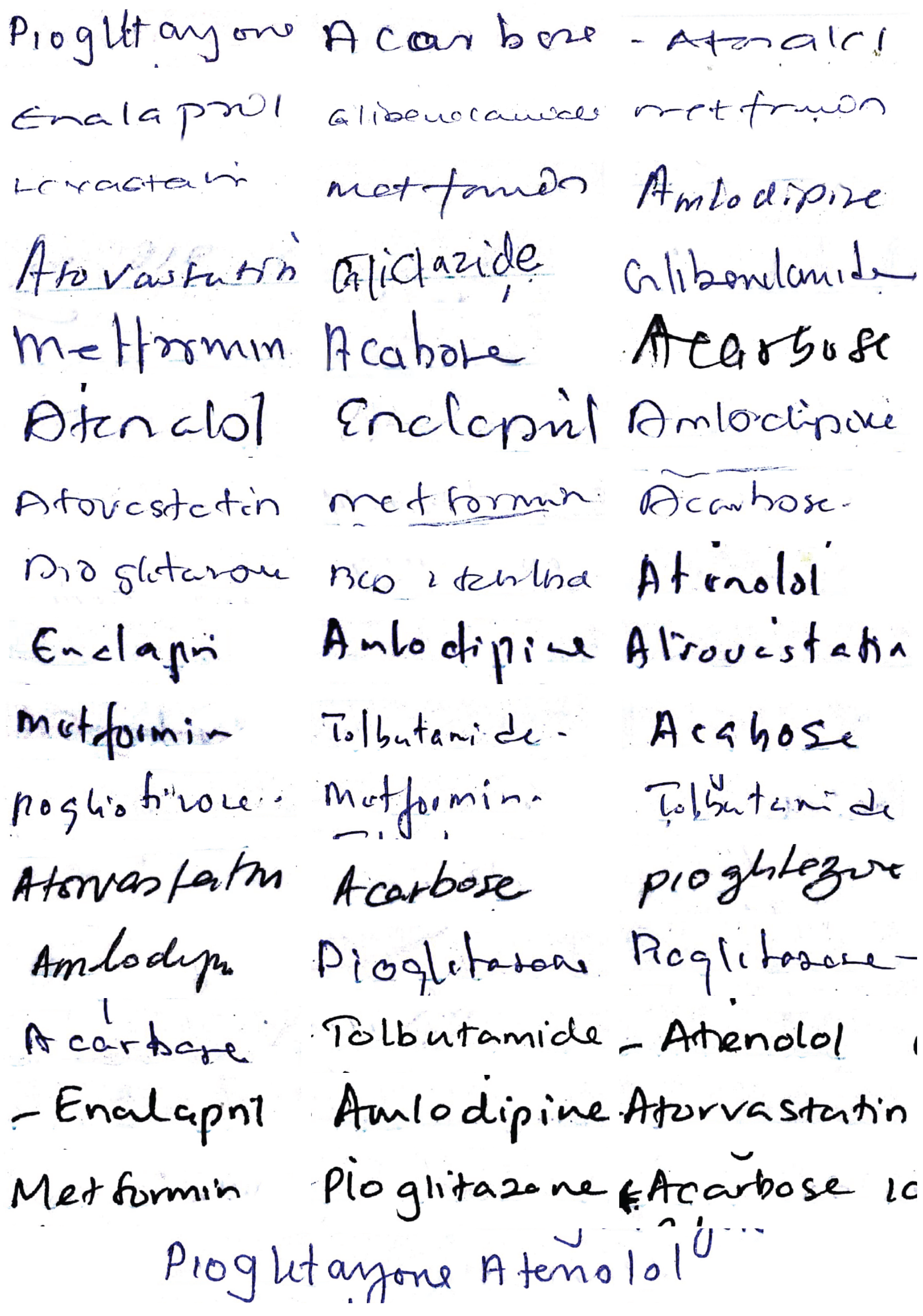


Fig. 5.1: Segmented Single Line images

Tab. 5.1: Results.

Image ID	Label	Recognized	Matched	True/False
1	pioglitazone	piogltanons	pioglitazone	TRUE
2	Acarbose	pcaabos	Acarbose	TRUE
3	Atacand	-tairs	Starlix	FALSE
4	enalapril	inalapoes	enalapril	TRUE
5	-	alimomecamines	mecamylamine	FALSE

Image ID	Label	Recognized	Matched	True/False
6	metformin	rerefrin	ertugliflozin	FALSE
7	-	ecrracrait	verapamil	FALSE
8	metformin	mor-fomon	metformin	TRUE
9	amlodipine	Amboaipive	amlodipine	TRUE
10	atorvastatin	Arvastutsn	atorvastatin	TRUE
11	glipizide	sidmside	torsemide	FALSE
12	glibenclamide	Glbomlomdr	Altocor	FALSE
13	metformin	metfoomn	metformin	TRUE
14	Acarbose	pcabore	Acarbose	TRUE
15	Acarbose	Aearbuse	Acarbose	TRUE
16	atenolol	Sitencll	atenolol	TRUE
17	enalapril	inclopit	fosinopril	FALSE
18	amlodipine	Amlodipue	amlodipine	TRUE
19	atorvastatin	proveatatin	rosuvastatin	FALSE
20	metformin	meosoun	metformin	TRUE
21	Acarbose	Pcctior-	Actos	FALSE
22	pioglitazone	riosltaron	rosiglitazone	FALSE
23		ieodeta	Riomet	FALSE
24	atenolol	Atinoli	atenolol	TRUE
25	enalapril	Enclapris	enalapril	TRUE
26	amlodipine	Auloctipie	amlodipine	TRUE
27	atorvastatin	blirovistatin	lovastatin	FALSE
28	metformin	mnetformi-	metformin	TRUE
29	Tolbutamide	Folbntanide-	Tolbutamide	TRUE
30	Acarbose	Acahose	Acarbose	TRUE
31	pioglitazone	poghstinoue	prazosin	FALSE
32	metformin	Iutformin-	metformin	TRUE
33	Tolbutamide	Plintanide	pramlintide	FALSE
34	atorvastatin	Atorvastatn	atorvastatin	TRUE
35	Acarbose	Acarbose	Acarbose	TRUE
36	pioglitazone	proghtegue	pioglitazone	TRUE
37	amlodipine	Alodn	Amlobenz	FALSE
38	pioglitazone	pioglitatons	pioglitazone	TRUE
39	pioglitazone	Roglitooo-	miglitol	FALSE
40	Acarbose	pcirtore	spironolactone	FALSE
41	Tolbutamide	solbutanide	Tolbutamide	TRUE
42	atenolol	ctiendod	atenolol	TRUE
43	enalapril	EEnalapnt	enalapril	TRUE
44	amlodipine	Aolodipine	amlodipine	TRUE
45	atorvastatin	Arorvastation	atorvastatin	TRUE
46	metformin	Mersormin	metformin	TRUE
47	pioglitazone	piogritasane	pioglitazone	TRUE
48	Acarbose	prarbosee	Acarbose	TRUE
49	pioglitazone	progltanpone	pioglitazone	TRUE
50	atenolol	AtroIoll	metoprolol	FALSE

6. CONCLUTIONS

6.1 Introcution

This chapter focuses on the conclusions drawn upon the completion of the research. The aim of the research as stated in Section 1.2.2 has been accomplished by using Neural networks and knowledge base mapping. These technologies become more and more advanced day by day by the introduction of more sophisticated machine learning techniques. Developing a system that can accurately identify the content of a prescription using these technologies will help the effectiveness of the treatment by reducing misreading and dispensing wrong drug to the patients.

The subsequent sections in this chapter will discuss further the conclusions of this research.

6.2 Conclusion about the Research Question

- How to identify the content of a handwritten medical prescription using the image of the prescription

This research proposes method to accuratly identify the content of a medical prescription using image processing and neural networks. The system is developed under the specifications mentioned in Section 4.3. Methodology proposed in this research utilizes the CNN's ability to extract features of a handwritten character image and RNN's ability to correctly identify them. To make the output more accurate, a mapping with a domain knowledge base in included so that the result would be an actual drug name which is most suitable for the recognized image rather than a simply recognized word, since the recognition system may not output the actual drug name. Evaluating on the dataset, the proposed method is able to identify the drug name with 63.10% accuracy. To conclude this question, this study present a method to accuratly identify the content of a medical prescription using image processing and neural networks. The study has further avenues to be explored that will be discussed in Section 6.5.

6.3 Conclusion about the Research Problem

This research has implemented a method to accuralty identify the content of a medical prescription. Using this method, the users can read the prescriptions even though they do not have a domain knowledge on the prescriped drugs.

6.4 Limitations

There are several limitations to this research.

- Segmentation of the prescription image cannot be done using any other method than manual
- Content that are identified will be only the entries from the knowledge base.
- Only single line segmented images can be identified by the NN.
- Entries in the knowledge base is limited to three illnesses.

6.5 Implications for further research

This research can be further improved in many ways. First and formost, the segmentation and the preprocessing phase needed to be automated. Then the content of the knowledge base can be expanded to identify more drugs than the current amount. Also this system can be improved to read the whole prescription rather than a single line. Also this research need to expanded id identify drugs prescriobed for other illnesses than the mentioned.

7. BIBLIOGRAPHY

1. J.Pradeep, E.Srinivasan, S.Himavathi "Diagonal Based Feature Extraction For Handwritten Character Recognition System Using Neural Network"
2. Y. Le Cun, 0. Matan, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel and H. S. Baird "Handwritten Zip Code Recognition with Multilayer Networks'
3. Gregory Dzuba, Alexander Filatov, Alexander Volguninc "Handwritten ZIP Code Recognition"
4. Jonathan J. Hull, Sargur N. Srihari, Ed Cohen, Leonard Kuan, Peter Cullen and Paul Palumbo "A Blackboard-based Approach to Handwritten ZIP Code Recognition"
5. F. Gmura, M. Shridhar and Z. Chen "Improvements of a Lexicon Directed Algorithm for Recognition of Unconstrained Handwritten Words"
6. Sargur N. Srihari, Venu Govindaraju and Ajay Shelihawat "Interpretation of Handwritten Addresses in US Mailstream"
7. A. Rajavelu, M. T. Musavi, And M. V. Shirvaikar "A Neural Network Approach to Character Recognition"
8. Soharab Hossain Shaikh · Asis Kumar Maiti ·Nabendu Chaki "A new image binarization method using iterative partitioning"
9. Nobuyuki Otsu "A Threshold Selection Method from Gray-Level Histograms"
10. Qiang Chena,, Quan-sen Suna, Pheng Ann Hengb,c, De-shen Xiaa "A double-threshold image binarization method based on edge detector"
11. B. Gatos, I. Pratikakis, S.J. Perantonis "Adaptive degraded document image binarization"
12. H. Brits, A. Botha, L. Niksch, R. Terblanché, K. Venter & G. Joubert ""Illegible handwriting and other prescription errors on prescriptions at National District Hospital, Bloemfontein"
13. J. Sauvola , M. PietikaKinen "Adaptive document image binarization"
14. Reza FarrahiMoghaddam n, MohamedCheriet "AdOtsu: An adaptive and parameterless generalization of Otsu's method for document image binarization"
15. Jaakko Sauvola, Tapio Seppanen, Sami Haapakoski and Matti Pietikiiinen "Adaptive Document Binarization"

16. Dan Claudiu Cires¸an and Ueli Meier and Luca Maria Gambardella and J urgen Schmidhuber “”Convolutional Neural Network Committees For Handwritten Character Classification”

17. Kunihiko Fukushima “Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition”

18. OIVIND DUE TRIER,? ANIL K. JAIN§ and TORFINN TAXT: “”Feature Extraction Methods For Character Recognition–a Survey”

19. “Illegible handwriting and other prescription errors on prescriptions at National District Hospital, Bloemfontein” https://www.tandfonline.com/doi/full/10.1080/20786

20. https://en.wikipedia.org/wiki/List_of_abbreviations_used_in_medical_prescriptions “List of abbreviations used in medical prescriptions”

21. https://en.wikipedia.org/wiki/Medical_prescription “Medical prescription”

22. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2723200/ “Medication errors: prescribing faults and prescription errors”

23. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3035877/ “Prescription Drug Labeling Medication Errors: A Big Deal for Pharmacists”

24. https://academic.oup.com/qjmed/article/102/8/513/1598923 “Medication errors: what they are, how they happen, and how to avoid them”

25. http://onlinelibrary.wiley.com/doi/10.1002/psb.96/pdf “A misread abbreviation thatled to a digoxin overdose”

26. https://stackoverflow.com/questions/34981144/split-text-lines-in-scanned-document “Split text lines in scanned document”

27. https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5

28. https://www.drugs.com/condition/hypoglycemia.html

29. https://www.drugs.com/condition/hypertension.htmlsort=drug&order=asc&page_size=25&

30. https://www.rxlist.com/high_blood_pressure_hypertension_medications/drugs-condition.htm

31. https://www.healthline.com/health/diabetes/medications-list#other-drugs

32. https://www.webmd.com/diabetes/diabetes-medications

APPENDICES

CODE LISTING

7.1 dataloader.py

```
from \_\_future\_\_ import division
from\_\_future\_\_ import print\_function

import os
import random
import numpy as np
import cv2
from SamplePreprocessor import preprocess


class Sample:
        "sample from the dataset"
        def \_\_init\_\_(self, gtText, filePath):
                self.gtText = gtText
                self.filePath = filePath



class Batch:
        "batch containing images and ground truth texts"
        def \_\_init\_\_(self, gtTexts, imgs):
                self.imgs = np.stack(imgs, axis=0)
                self.gtTexts = gtTexts
class DataLoader:
        "loads data which corresponds to IAM
         format, see: http://www.fki.inf.unibe.ch/

         databases/iam-handwriting-database"

        def \_\_init\_\_(self, filePath, batchSize,
         imgSize, maxTextLen):
                "loader for dataset at given location
                 preprocess images and text according
                  to parameters"

                assert filePath[-1]=='/'

                self.dataAugmentation = False
                self.currIdx = 0
        self.batchSize = batchSize
        self.imgSize = imgSize
        self.samples = []

        f=open(filePath+'words.txt')
        chars = set()
        bad\_samples = []
        bad\_samples\_reference =

        ['a01-117-05-02.png', 'r06-022-03-05.png']
        for line in f:
                \# ignore comment line
                if not line or line[0]=='\#':
                        continue
                lineSplit = line.strip().split(' ')
                assert len(lineSplit) >= 9

                \# filename: part1-part2-part3 -->
                 part1/part1-part2/part1-part2-part3.png
                fileNameSplit = lineSplit[0].split('-')
                fileName = filePath + 'words/' +
                fileNameSplit[0] + '/' + fileNameSplit[0] +
                 '-' + fileNameSplit[1] + '/' + lineSplit[0] +

                \# GT text are columns starting at 9
                gtText = self.truncateLabel(' '.join(lineSplit[8

                maxTextLen)
                chars = chars.union(set(list(gtText)))

                \# check if image is not empty
                if not os.path.getsize(fileName):
                        bad\_samples.append(lineSplit[0] + '.png
                        continue

                \# put sample into list
                self.samples.append(Sample(gtText, fileName))
\# some images in the IAM dataset are known to be damaged,
don't show warning for them
        if set(bad\_samples) != set(bad\_samples\_reference):
                print("Warning, damaged images found:", bad\_sam
                print("Damaged images expected:", bad\_samples\

        \# split into training and validation set: 95\% - 5\%
        splitIdx = int(0.95 * len(self.samples))
        self.trainSamples = self.samples[:splitIdx]
        self.validationSamples = self.samples[splitIdx:]
        \# put words into lists
        self.trainWords = [x.gtText for x in self.trainSamples]
        self.validationWords = [x.gtText for x in self.validatio

        \# number of randomly chosen samples per epoch for trair
        self.numTrainSamplesPerEpoch = 25000

        \# start with train set
        self.trainSet()

        \# list of all chars in dataset
        self.charList = sorted(list(chars))

        def truncateLabel(self, text, maxTextLen):
        \# ctc\_loss can't compute loss if it cannot find a map
\# labels. Repeat letters cost double because of the blank symb
        \# If a too-long label is provided, ctc\_loss returns ar
        cost = 0
        for i in range(len(text)):
                if i != 0 and text[i] == text[i-1]:
                        cost += 2
                else:
                        cost += 1
                if cost > maxTextLen:
                        return text[:i]
        return text

def trainSet(self):
        "switch to randomly chosen subset of training set"
        self.dataAugmentation = True
        self.currIdx = 0
        random.shuffle(self.trainSamples)
        self.samples = self.trainSamples[:self.numTrainSamplesP


def validationSet(self):
        "switch to validation set"
        self.dataAugmentation = False
        self.currIdx = 0
        self.samples = self.validationSamples


def getIteratorInfo(self):
        "current batch index and overall number of batches"
        return (self.currIdx // self.batchSize + 1, len(self.sar


def hasNext(self):
        "iterator"
        return self.currIdx + self.batchSize <= len(self.samples
        def getNext(self):
        "iterator"
        batchRange = range(self.currIdx, self.currIdx + self.ba
        gtTexts = [self.samples[i].gtText for i in batchRange]
        imgs = [preprocess(cv2.imread(self.samples[i].filePath,
        cv2.IMREAD\_GRAYSCALE), self.imgSize, self.dataAugmenta
         for i in batchRange]
        self.currIdx += self.batchSize
        return Batch(gtTexts, imgs)
```

7.2 samplepreprocessor.py

```
from\_\_future\_\_ import division
from \_\_future\_\_ import print\_function

import random
import numpy as np
import cv2


def preprocess(img, imgSize, dataAugmentation=False):
        "put img into target img of size imgSize, transpose for
        TF and normalize gray-values"

        \# there are damaged files in IAM dataset - just use black image
        if img is None:
                img = np.zeros([imgSize[1], imgSize[0]])

        \# increase dataset size by applying random stretches to the im
        if dataAugmentation:
                stretch = (random.random() - 0.5) \#
                -0.5 .. +0.5
                wStretched = max(int(img.shape[1] * (1 + stretch)),
                1) \# random width, but at least 1
                img = cv2.resize(img, (wStretched, img.shape[0]))
                \# stretch horizontally by factor 0.5 .. 1.5

        \# create target image and copy sample image into it
        (wt, ht) = imgSize
        (h, w) = img.shape
        fx = w / wt
        fy = h / ht
        f = max(fx, fy)
        newSize = (max(min(wt, int(w / f)), 1), max(min(ht,
         int(h / f)), 1)) \# scale according to f (result at least
         1 and at most wt or ht)
        img = cv2.resize(img, newSize)
        target = np.ones([ht, wt]) * 255
        target[0:newSize[1], 0:newSize[0]] = img

        \# transpose for TF
        img = cv2.transpose(target)
        \# normalize
        (m, s) = cv2.meanStdDev(img)
        m = m[0][0]
        s = s[0][0]
        img = img - m
        img = img / s if s>0 else img
        return img
```

7.3 Model.py

```
from \_\_future\_\_ import division
from \_\_future\_\_ import print\_function

import sys
import tensorflow as tf


class DecoderType:
        BestPath = 0
        BeamSearch = 1
        WordBeamSearch = 2


class Model:
        "minimalistic TF model for HTR"

        \# model constants
        batchSize = 50
        imgSize = (128, 32)
        maxTextLen = 32

        def \_\_init\_\_(self, charList, decoderType=
        DecoderType.BestPath, mustRestore=False):
                "init model: add CNN, RNN and CTC and initialize TF"
                self.charList = charList
                self.decoderType = decoderType
                self.mustRestore = mustRestore
                self.snapID = 0

                \# CNN
                self.inputImgs = tf.placeholder(tf.float32,
                shape=(Model.batchSize, Model.imgSize[0],
                 Model.imgSize[1]))
                cnnOut4d = self.setupCNN(self.inputImgs)


                \# RNN
                rnnOut3d = self.setupRNN(cnnOut4d)

                \# CTC
                (self.loss, self.decoder) = self.setupCTC(rnnOut3d)

                \# optimizer for NN parameters
                self.batchesTrained = 0
                self.learningRate = tf.placeholder(tf.float32,
                shape=[])
                self.optimizer = tf.train.RMSPropOptimizer
                (self.learningRate).minimize(self.loss)

                \# initialize TF
                (self.sess, self.saver) = self.setupTF()


        def setupCNN(self, cnnIn3d):
                "create CNN layers and return output of these layers"
                cnnIn4d = tf.expand\_dims(input=cnnIn3d, axis=3)

                \# list of parameters for the layers
                kernelVals = [5, 5, 3, 3, 3]
                featureVals = [1, 32, 64, 128, 128, 256]
                strideVals = poolVals = [(2,2), (2,2), (1,2),
                 (1,2), (1,2)]
                numLayers = len(strideVals)

                \# create layers
                pool = cnnIn4d \# input to first CNN layer
                for i in range(numLayers):
                        kernel = tf.Variable(tf.truncated\_norma
                        l([kernelVals[i], kernelVals[i], featureVals[i]
                        featureVals[i + 1]], stddev=0.1))
                        conv = tf.nn.conv2d(pool, kernel,
                        padding='SAME',  strides=(1,1,1,1))
                        relu = tf.nn.relu(conv)
                        pool = tf.nn.max\_pool(relu, (1, poolVals
                        [i][0], poolVals[i][1], 1), (1, strideVals[i][0]
                         strideVals[i][1], 1), 'VALID')

                return pool
```

```
def setupRNN(self, rnnIn4d):
"create RNN layers and return output of these layers"
rnnIn3d = tf.squeeze(rnnIn4d, axis=[2])

\# basic cells which is used to build RNN
numHidden = 256
cells = [tf.contrib.rnn.LSTMCell(num\_units=
numHidden, state\_is\_tuple=True) for \_ in
range(2)] \# 2 layers

\# stack basic cells
stacked = tf.contrib.rnn.MultiRNNCell(cells,
state\_is\_tuple=True)

\# bidirectional RNN
\# BxTxF -> BxTx2H
((fw, bw), \_) = tf.nn.bidirectional\_dynamic\_
rnn(cell\_fw=stacked, cell\_bw=stacked, inputs=
rnnIn3d, dtype=rnnIn3d.dtype)

\# BxTxH + BxTxH -> BxTx2H -> BxTx1X2H
concat = tf.expand\_dims(tf.concat([fw, bw],
 2), 2)

\# project output to chars (including blank):
 BxTx1x2H -> BxTx1xC -> BxTxC
kernel = tf.Variable(tf.truncated\_normal([1, 1
, numHidden * 2, len(self.charList) + 1], stddev

=0.1))
return tf.squeeze(tf.nn.atrous\_conv2d(value=
concat, filters=kernel, rate=1, padding='SAME'),

axis=[2])
def setupCTC(self, ctcIn3d):
"create CTC loss and decoder and return them"
\# BxTxC -> TxBxC
ctcIn3dTBC = tf.transpose(ctcIn3d, [1, 0, 2])
\# ground truth text as sparse tensor
self.gtTexts = tf.SparseTensor(tf.placeholder
(tf.int64, shape=[None, 2]) , tf.placeholder(tf.int32,
 [None]), tf.placeholder(tf.int64, [2]))
\# calc loss for batch
self.seqLen = tf.placeholder(tf.int32, [None])
loss = tf.nn.ctc\_loss(labels=self.gtTexts,
 inputs=ctcIn3dTBC, sequence\_length=

 self.seqLen, ctc\_merge\_repeated=True)
```

```
\# decoder: either best path decoding or beam search dec
if self.decoderType == DecoderType.
BestPath:
        decoder = tf.nn.ctc\_greedy\_decoder
        (inputs=ctcIn3dTBC, sequence\_length=self.seqLen
elif self.decoderType == DecoderType
.BeamSearch:
        decoder = tf.nn.ctc\_beam\_search\_
        decoder(inputs=ctcIn3dTBC, sequence
        length=self.seqLen, beam\_width=50, merge\_repea
elif self.decoderType == DecoderType.
WordBeamSearch:
        \# import compiled word beam search
        operation (see https://github.com/githubharald

        /CTCWordBeamSearch)
        word\_beam\_search\_module = tf.load

        \_op\_library('TFWordBeamSearch.so')

        \# prepare information about language
        (dictionary, characters in dataset, characters
         forming words)
        chars = str().join(self.charList)
        wordChars = open('../model/wordCharList.txt').
        read().splitlines()[0]
        corpus = open('../data/corpus.txt').read()

        \# decode using the "Words" mode of word
         beam search
        decoder = word\_beam\_search\_module.
        word\_beam\_search(tf.nn.softmax(ctcIn3d
        TBC, dim=2), 50, 'Words', 0.0, corpus.encode
        ('utf8'), chars.encode('utf8'), wordChars.encod
        e('utf8'))

\# return a CTC operation to compute the loss and

a CTC operation to decode the RNN output
return (tf.reduce\_mean(loss), decoder)

def setupTF(self):
"initialize TF"
print('Python: '+sys.version)
print('Tensorflow: '+tf.\_\_version\_\_)

sess=tf.Session() \# TF session
```

```
saver = tf.train.Saver(max\_to\_keep=1) \#
 saver saves model to file
modelDir = '../model/'
latestSnapshot = tf.train.latest\_checkpoint(modelDir)
 \# is there a saved model?

\# if model must be restored (for inference),
there must be a snapshot
if self.mustRestore and not latestSnapshot:
        raise Exception('No saved model found
        in: ' + modelDir)

\# load saved model if available
if latestSnapshot:
        print('Init with stored values from ' +
         latestSnapshot)
        saver.restore(sess, latestSnapshot)
else:
        print('Init with new values')
        sess.run(tf.global\_variables\_initializer())

return (sess,saver)
def toSparse(self, texts):
"put ground truth texts into sparse tensor for ctc\_loss
indices = []
values = []
shape = [len(texts), 0] \# last entry must be
 max(labelList[i])



\# go over all texts
for (batchElement, text) in enumerate(texts):
        \# convert to string of label (i.e. class-ids)
        labelStr = [self.charList.index(c) for c in text
        \# sparse tensor must have size of max.
         label-string
        if len(labelStr) > shape[1]:
                shape[1] = len(labelStr)
        \# put each label into sparse tensor
        for (i, label) in enumerate(labelStr):
                indices.append([batchElement, i])
                values.append(label)

return (indices, values, shape)

def decoderOutputToText(self, ctcOutput):
"extract texts from output of CTC decoder"
```

```
        \# contains string of labels for each batch
        element
        encodedLabelStrs = [[] for i in range
        (Model.batchSize)]

        \# word beam search: label strings terminated by blank
        if self.decoderType == DecoderType.WordBeamSearch:
                blank=len(self.charList)
                for b in range(Model.batchSize):
                        for label in ctcOutput[b]:
                                if label==blank:
                                        break
                                encodedLabelStrs[b].append(label

        \# TF decoders: label strings are contained in sparse te
        else:
                \# ctc returns tuple, first element is SparseTe
                decoded=ctcOutput[0][0]

                \# go over all indices and save mapping: batch -
                idxDict = { b : [] for b in range(Model.batchSi
                for (idx, idx2d) in enumerate(decoded.indices):
                        label = decoded.values[idx]
                        batchElement = idx2d[0] \# index accordi
                        encodedLabelStrs[batchElement].append(la

        \# map labels to chars for all batch elements
        return [str().join([self.charList[c] for c in labelStr])
        for labelStr in encodedLabelStrs]


def trainBatch(self, batch):
        "feed a batch into the NN to train it"
        sparse = self.toSparse(batch.gtTexts)
        rate = 0.01 if self.batchesTrained < 10 else
         (0.001 if self.batchesTrained < 10000 else
         0.0001) \# decay learning rate
        (\_, lossVal) = self.sess.run([self.optimizer,
        self.loss], { self.inputImgs : batch.imgs, self.
        gtTexts : sparse , self.seqLen : [Model.max

        TextLen] * Model.batchSize, self.learningRate : rate} )
        self.batchesTrained += 1
        return lossVal

        def inferBatch(self, batch):
        "feed a batch into the NN to recngnize the
         texts"
```

```
                decoded = self.sess.run(self.decoder, {

                 self.inputImgs : batch.imgs, self.seqLen :
                 [Model.maxTextLen] * Model.batchSize } )
                return self.decoderOutputToText(decoded)


        def save(self):
                "save model to file"
                self.snapID += 1
                self.saver.save(self.sess, '../model/snapshot',

                 global\_step=self.snapID)
```

7.4 main.py

```
from \_\_future\_\_ import division
from \_\_future\_\_ import print\_function

import sys
import argparse
import cv2
import editdistance
import pandas
import difflib
from DataLoader import DataLoader, Batch
from Model import Model, DecoderType
from SamplePreprocessor import preprocess


class FilePaths:
        "filenames and paths to data"
        fnCharList = '../model/charList.txt'
        fnAccuracy = '../model/accuracy.txt'
        fnTrain = '../data/'
        fnInfer = '../data/images/54-01.png'
        fnCorpus = '../data/corpus.txt'


def train(model, loader):
        "train NN"
        epoch = 0 \# number of training epochs
        since start
        bestCharErrorRate = float('inf') \# best
        valdiation character error rate
        noImprovementSince = 0 \# number of
```

```
 epochs no improvement of character
 error rate occured
earlyStopping = 5 \# stop training after
this number of epochs without improvement
while True:
        epoch += 1
        print('Epoch:', epoch)

        \# train
        print('Train NN')
        loader.trainSet()
        while loader.hasNext():
                iterInfo = loader.getIteratorInfo()
                batch = loader.getNext()
                loss = model.trainBatch(batch)
                print('Batch:', iterInfo[0],'/',
                 iterInfo[1], 'Loss:', loss)
                \# validate
        charErrorRate = validate(model, loader)

        \# if best validation accuracy so far, save
         model parameters
        if charErrorRate < bestCharErrorRate:
                print('Character error rate improved,
                save model')
                bestCharErrorRate = charErrorRate
                noImprovementSince = 0
                model.save()
                open(FilePaths.fnAccuracy, 'w').write
                ('Validation character error rate of saved mode

                %f%%' % (charErrorRate*100.0))
        else:
                print('Character error rate not improved')
                noImprovementSince += 1

        \# stop training if no more improvement in the last x ep
        if noImprovementSince >= earlyStopping:
                print('No more improvement since
                 %d epochs. Training stopped.' % earlyStopping)
                break
def validate(model, loader):
"validate NN"
print('Validate NN')
loader.validationSet()
numCharErr = 0
numCharTotal = 0
numWordOK = 0
```

```
        numWordTotal = 0
        while loader.hasNext():
                iterInfo = loader.getIteratorInfo()
                print('Batch:', iterInfo[0],'/', iterInfo[1])
                batch = loader.getNext()
                recognized = model.inferBatch(batch)

                print('Ground truth -> Recognized')
                for i in range(len(recognized)):
                        numWordOK += 1 if batch.
                        gtTexts[i] == recognized[i] else 0
                        numWordTotal += 1
                        dist = editdistance.eval
                        (recognized[i], batch.gtTexts[i])
                        numCharErr += dist
                        numCharTotal += len
                        (batch.gtTexts[i])
                        print('[OK]' if dist==0 else
                        '[ERR:%d]' % dist,'"' +
                        batch.gtTexts[i] + '"', '->', '"'
                         + recognized[i] + '"')

        \# print validation result
        charErrorRate = numCharErr /
        numCharTotal
        wordAccuracy = numWordOK /
        numWordTotal
        print('Character error rate: %f%%.
        Word accuracy: %f%%.' % (charErrorRate*
        100.0, wordAccuracy*100.0))
        return charErrorRate

        def infer(model, fnImg):
        "recognize text in image provided by
        file path"
        img = preprocess(cv2.imread(fnImg,
         cv2.IMREAD\_GRAYSCALE), Model.imgSize)
        batch = Batch(None, [img] * Model.
        batchSize) \# fill all batch elements with
         same input image
        recognized = model.inferBatch(batch)
        \# recognize text
        \#print('Recognized:', '"' +
        recognized[0] + '"') \# all batch elements hold same result
        return recognized[0]


def main():
```

```
"main function"
\# optional command line args
parser = argparse.ArgumentParser()
parser.add\_argumen
t("--train", help="train the NN",
action="store\_true")
parser.add\_argument
("--validate", help="validate the NN",
action="store\_true")
parser.add\_argument
("--beamsearch", help="use beam search
 instead of best path decoding", action="store\_true")

parser.add\_argument("--wordbeamsearch",
help="use word beam search instead of best
path decoding", action="store\_true")
args = parser.parse\_args()

decoderType = DecoderType.BestPath
if args.beamsearch:
        decoderType = DecoderType.BeamSearch
elif args.wordbeamsearch:
        decoderType = DecoderType.WordBeamSearch

\# train or validate on IAM dataset
if args.train or args.validate:
        \# load training data, create TF model
        loader = DataLoader(FilePaths.fnTrain,
        Model.batchSize, Model.imgSize, Model.maxTextLen)

        \# save characters of model for inference mode
        open(FilePaths.fnCharList, 'w').write(str()
        .join(loader.charList))

        \# save words contained in dataset into file
        open(FilePaths.fnCorpus, 'w').write(str(' ').
        join(loader.trainWords + loader.validationWords))

        \# execute training or validation
        if args.train:
                model = Model(loader.charList, decoderType)
                train(model,loader)
        elif args.validate:
                model = Model(loader.charList, decoderType,
                mustRestore=True)
                validate(model, loader)

\# infer text on test image
```

```
else:

                                model = Model(open(FilePaths.fnCharList
                                read(), decoderType, mustRestore=True)
                                word = infer(model, FilePaths.fnInfer)
                                print(FilePaths.fnInfer)
                                print('Recognized:', '"' + word + '"')
                                dbcol =('d')
                                hbpcol =('p')
                                chlcol =('c')
                                db = pandas.read\_
                                csv('Hypo-Hyperglycemia.csv',names=dbco
                                hbp = pandas.read\_
                                csv('Hypertension.csv',names=hbpcol)
                                chl = pandas.read\_
                                csv('Hyperlipidemia.csv',names=chlcol)
                                diabetes = db.d.tolist()
                                pressure = hbp.p.tolist()
                                chol = chl.c.tolist()
                                dbr = []
                                hbpr = []
                                chlr = []
                                for eled in diabetes:
                                            seqd = difflib.
                                            SequenceMatche
                                            r(None, word,eled)
                                            dd = seqd.ratio()*100
                                            dbr.append(dd)

                                for eleh in pressure:
                                            seqh= difflib.
                                            SequenceMatcher
                                            (None, word,eleh)
                                            dh = seqh.ratio()*100
                                            hbpr.append(dh)

                                for elec in chol:
                                            seqc = difflib.
                                            SequenceMatcher(None, w
                                            dc = seqc.ratio()*100
                                            chlr.append(dc)
                                db\_max = max(dbr)
                                db\_max\_in = dbr.index(db\_max)

                                hbp\_max = max(hbpr)
                                hbp\_max\_in = hbpr.index(hbp\_max)

                                chl\_max = max(chlr)
                                chl\_max\_in = chlr.index(chl\_max)

                                max\_val = [db\_max,hbp\_max,chl\_max]
                                max\_of\_all = max(max\_val)
                                max\_of\_all\_in = max\_val.index(max\_

                                if max\_of\_all\_in == 0:
                                        name = diabetes[db\_max\_in]
                                elif max\_of\_all\_in == 1:
                                        name = pressure[hbp\_max\_in]
                                else:
                                        name = chol[chl\_max\_in]
                                print('found:', '"' + name + '"')
if \_\_name\_\_ == '\_\_main\_\_':
        main()
```