HANDWRITTEN TEXT RECOGNITION

A Major Project Report submitted
in partial fulfillment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING

By

1. 18B01A0563 - A.S.S.Sanjana
2. 18B01A0577 – M.Persis Edith
3. 18B01A0586 – U.Aasin
4.18B01A05B2 - P.Deepthi Bhatnakar

Under the esteemed guidance of

Mr. P. SUNIL M.Tech

Assistant professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SHRI VISHNU ENGINEERING COLLEGE FOR WOMEN(A)

(Approved by AICTE, Accredited by NBA & NAAC, Affiliated to JNTU Kakinada

BHIMAVARAM – 534202

2021 – 2022

SHRI VISHNU ENGINEERING COLLEGE FOR WOMEN(A)

(Approved by AICTE, Accredited by NBA & NAAC, Affiliated to JNTU Kakinada)

BHIMAVARAM – 534 202

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

*This is to certify that the Mini Project-II entitled "**HANDWRITTEN TEXT RECOGNITION**" , is being submitted by **A. S. S. Sanjana, M. Persis Edith, U. Aasin, P. Deepthi Bhatnakar** bearing the **Regd. No. 18B01A0563,18B01A0577,18B01A0586,18B01A05B2** in partial fulfillment of the requirements for the award of the degree of "**Bachelor of Technology** in **Computer Science & Engineering**" is a record of bonafide work carried out by them under my guidance and supervision during the academic year **2021 – 2022** and it has been found worthy of acceptance according to the requirements of the university.*

Internal Guide **Head of the Department**

External Examiner

ACKNOWLEDGMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant encouragement and guidance has been a source of inspiration throughout the course of this project. I take this opportunity to express our gratitude to all those who have helped us in this project.

We wish to place our deep sense of gratitude to **Sri. K. V. Vishnu Raju, Chairman of SVES**, for his constant support on each and every progressive work of mine.

We wish to express our sincere thanks to **Dr. G. Srinivasa Rao, Principal of SVECW** for being a source of an inspirational and constant encouragement.

We wish to express our sincere thanks to **Dr. P. Srinivasa Raju, Vice-Principal of SVECW** for being a source of an inspirational and constant encouragement.

We wish to place our deep sense of gratitude to **Dr. P. Kiran Sree, Head of the Department of Computer Science & Engineering** for his valuable pieces of advice in completing this project successfully.

Our deep sense of gratitude and sincere thanks to **Mr. P. Sunil Assistant Professor** for his unflinching devotion and valuable suggestion throughout my project work.

Project Associates

1. **A. S. S. SANJANA - 18B01A0563**
2. **M. PERSIS EDITH - 18B01A0577**
3. **U. AASIN - 18B01A0586**
4. **P. DEEPTHI - 18B01A05B2**

Abstract

Handwriting recognition has been one of the active and challenging research areas in the field of image processing and pattern recognition. Handwriting Detection is a technique or ability of a Computer to receive and interpret intelligible handwritten input from sources such as paper documents, notes, touch screens, photographs, etc. Handwritten Text recognition is one of the areas of pattern recognition.

The purpose of pattern recognition is to categorize or classification data or objects of one of the classes or categories. The goal of handwriting is to identify input characters or image correctly then analyzed them to many automated process systems. The development of handwriting is more sophisticated, which is found in various kinds of handwritten characters such as digit, numeral, cursive script, symbols, and scripts including English and other languages.

The automatic recognition of handwritten text can be extremely useful in many applications where it is necessary to process large volumes of handwritten data, such as recognition of addresses and postcodes on envelopes, doctor prescriptions, interpretation of amounts on bank checks, document analysis, and verification of signatures.

Research in the handwriting recognition field is focused on deep learning techniques and has achieved breakthrough performance in the last few years. Still, the rapid growth in the amount of handwritten data and the availability of massive processing power demands improvement in recognition accuracy and deserves further investigation. Convolutional neural networks (CNNs) are very effective in perceiving the structure of handwritten characters/words in ways that help in the automatic extraction of distinct features and make CNN the most suitable approach for solving handwriting recognition problems. This system will be applied to detect the writings of different formats.

Contents

1.INTRODUCTION

Despite the abundance of technological writing tools, many people still choose to take their notes traditionally: with pen and paper. However, there are drawbacks to handwriting text. It's difficult to store and access physical documents in an efficient manner, search through them efficiently and to share them with others.

Thus, a lot of important knowledge gets lost or does not get reviewed because of the fact that documents never get transferred to digital format. We have thus decided to tackle this problem in our project because we believe the significantly greater ease of management of digital text compared to written text will help people more effectively access, search, share, and analyze their records, while still allowing them to use their preferred writing method.

The aim of this project is to further explore the task of classifying handwritten text and to convert handwritten text into the digital format.

The purpose of this project is to take handwritten English text image as input, process each character, train the neural network algorithm, to recognize the pattern and modify the character to a beautified version of the input.

This project is aimed at developing software which will be helpful in recognizing characters of English language. It can be further developed to recognize the text of different languages. It engulfs the concept of neural network.

One of the primary means by which computers are endowed with humanlike abilities is through the use of a neural network. Neural networks are particularly useful for solving problems that cannot be expressed as a series of steps, such as recognizing patterns, classifying them into groups, series prediction and data mining. Pattern recognition is perhaps the most common use of neural networks.

The neural network is presented with a target vector and also a vector which contains the pattern information, this could be an image and hand written data. The neural network then attempts to determine if the input data matches a pattern that the neural network has memorized. A neural network trained for classification is designed to take input samples and classify them into groups. These groups may be fuzzy, without clearly defined boundaries. This project concerns detecting free handwritten text.

2. SYSTEM ANALYSIS

2.1 EXISTING SYSTEM: -

- Handwritten Text Recognition exists in the form of software Google Lens and also in the form of hardware is OCR scanners.
- In existing models, they have recognized either digits or characters, trained the model with the help of the MNIST dataset.

2.2 PROPOSED SYSTEM: -

- The aim of our project is to make an model for handwritten text recognition for application in healthcare and personal care that can recognize handwriting using concepts of deep learning.
- This model is used to convert the text into different forms ie, mainly Text documents and files.
- **Objectives**
    - To provide an easy user interface to input the handwritten text image.
    - User should be able to upload the image.
    - System should be able to pre-process the given input to suppress the background.
    - System should detect text regions present in the image.
    - System should retrieve text present in the image and display them to the user

2.3 FEASIBILITY STUDY: -

Generally, the feasibility study is used for determining the resource cost, benefits and whether the proposed system is feasible with respect to the organization. The proposed system feasibility could be as follows. There are six types of feasibility which are equally important are:

Technical Feasibility: -

Technical feasibility deals with the existing technology, software and hardware requirements for the proposed system. The proposed system "HANDWRITTEN TEXT RECOGNITION" is planned to run on Python.

Economic Feasibility: -

This method is most frequently used for evaluating the effectiveness of a Python. It is also called as benefit analysis. In this project "HANDWRITTEN TEXT RECOGNITION" is developed on current equipment, existing software technology.

Behavioural Feasibility: -

This proposed system "HANDWRITTEN TEXT RECOGNITION" Application has much behavioural feasibility because users are provided with a better facility.

3. SYSTEM REQUIREMENTS SPECIFICATION

3.1 SOFTWARE REQUIREMENTS: -

Tool : Jupyter Notebook, Google Colab, Sublime Text Editor

Programming Language : Python

3.2 HARDWARE REQUIREMENTS: -

RAM : 4GB.

Processor : Intel core i3 or above.

Hard disk space : 120GB.

3.3 FUNCTIONAL REQUIREMENTS: -

The developed system should recognize handwritten text characters present in the image. The system shall show the error message to the user when the given input is not in the required format. The system must provide the quality of service to the user. The system must provide spell correction for text recognition.

3.4 NON-FUNCTIONAL REQUIREMENTS: -

Non-functional requirements describe user-visible aspects of the system that are not directly related to the functional behavior of the system.

- **Performance**: Handwritten text in the input image will be recognized with an accuracy of about 90
- **Availability**: This system will retrieve the handwritten text regions only if the image contains written text in it.

- **Flexibility**: It provides the users to load the image easily.
- **Learnability**: The software is very easy to use and reduces the learning work.

4.SYSTEM DESIGN

4.1 Introduction

4. 1.1 Purpose

The purpose of this design is to explore the logical view of architecture design, sequence diagram, data flow diagram, user interface design of the software for performing the operations such as pre-proccssing, extracting features and displaying the text present in the images.

4. 1.2 Scope

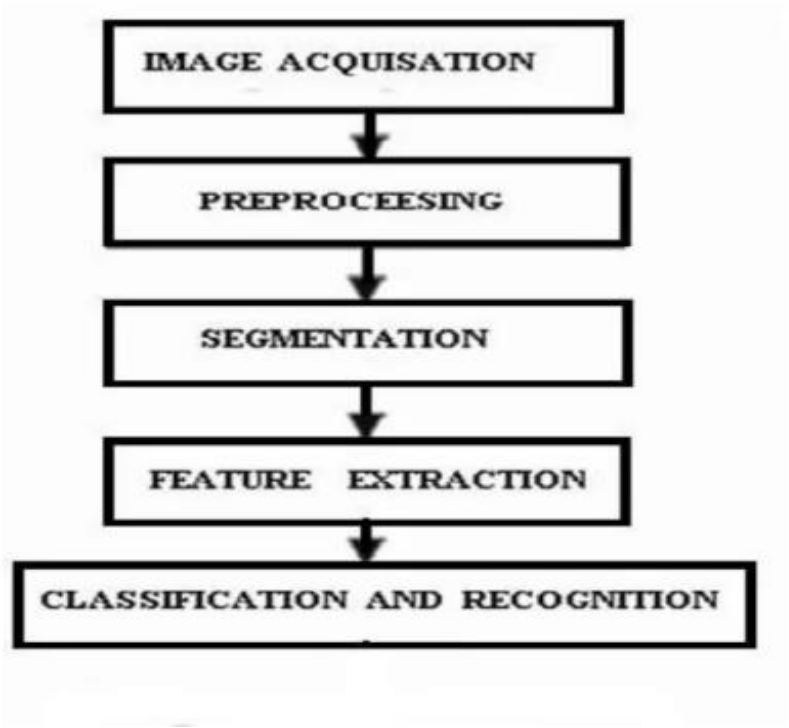
The scope of this design is to achieve the features of the system such as pre-processing the images, feature extraction, segmentation and display the text present in the image.

4. 1.3 Block Diagram and Algorithm

The proposed methodology uses some techniques to remove the background noise, andfeaturcs extraction to detect and classify the handwritten text.

The proposed method comprises of 4 phases:

1. Pre-processing.
2. Segmentation.
3. Feature Extraction.
4. Classification and Recognition.



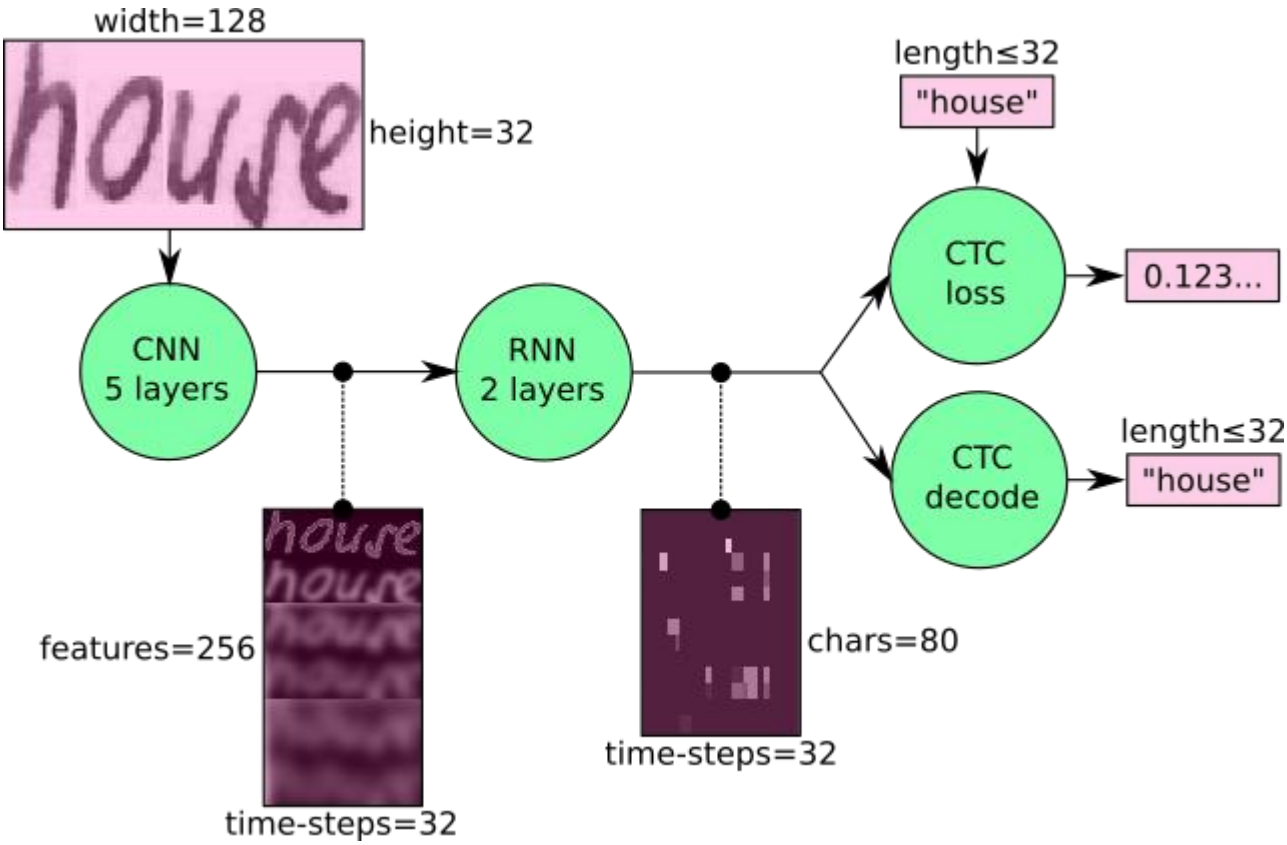
1. Pre-processing

The pre-processing is a series of operations performed on scanned input image. It essentially enhances the image rendering it suitable for segmentation. The role of pre-processing is to segment the interesting pattern from the background. Generally, noise filtering, smoothing and normalization should be done in this step. The pre-processing also defines a compact representation of the pattern. Binarization process converts a gray scale image into a binary image. Dilution of edges in the binarized image is done using sobel technique.

2. Segmentation

In the segmentation stage, an image of sequence of characters is decomposed into sub-images of individual character. The pre-processed input image is segmented into isolated characters by assigning a number to each character using a labelling process. This labelling provides information about number of characters in the image. Each individual character is uniformly resized into pixels. Normalization: After extracting the character we need to normalize the size of the characters. There are large variations in the sizes of each Character hence we need a method to normalize the size.

We use a NN for our task. It consists of convolutional NN (CNN) layers, recurrent NN (RNN) layers and a final Connectionist Temporal Classification (CTC) layer.

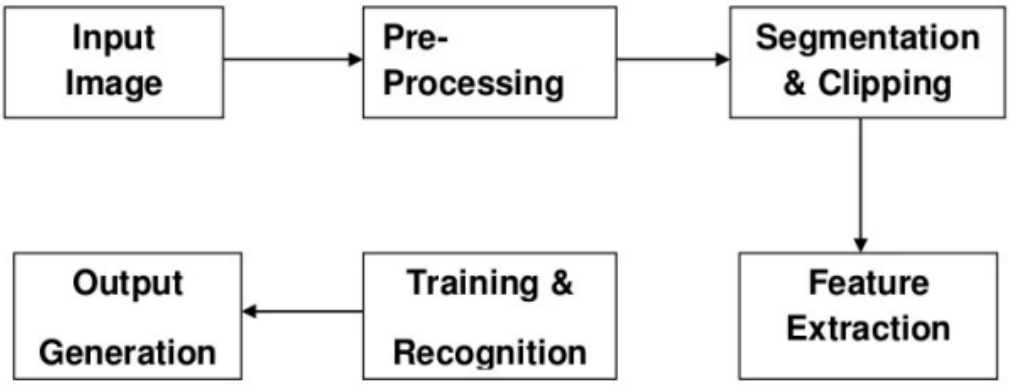


System design is the process of designing the elements of a system such as the architecture, modules and components, the different interfaces of those components and the data that goes through that system.

We can also view the NN in a more formal way as a function (see Eq. 1) which maps an image (or matrix) M of size W×H 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).

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

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



Architecture of the System

System Analysis is the process that decomposes a system into its component pieces for the purpose of defining how well those components interact to accomplish the set requirements. The purpose of the System Design process is to provide sufficient detailed data and information about the system. The purpose of the design phase is to plan a solution of the problem specified by the requirement document. This phase is the first step in moving from problem domain to the solution domain. The design of a system is perhaps the most critical factor affecting the quality of the software, and has a major impact on the later phases, particularly testing and maintenance.

A design methodology is a systematic approach to creating a design by application of set of techniques and guidelines. Most methodologies focus on system design. The two basic principles used in any design methodology are problem partitioning and abstraction. Abstraction is a concept related to problem partitioning

4.2 UML DIAGRAMS

UMLDiagrams is a rich visualizing model for representing the system architecture and design.These diagrams help us to know the flow of the system. To model a system, the most important aspect is to capture the dynamic behavior. Dynamic behavior means the behavior of the system when it is running/operating.

Only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. In UML, there are five diagrams available to model the dynamic nature and use case diagram is one of them.

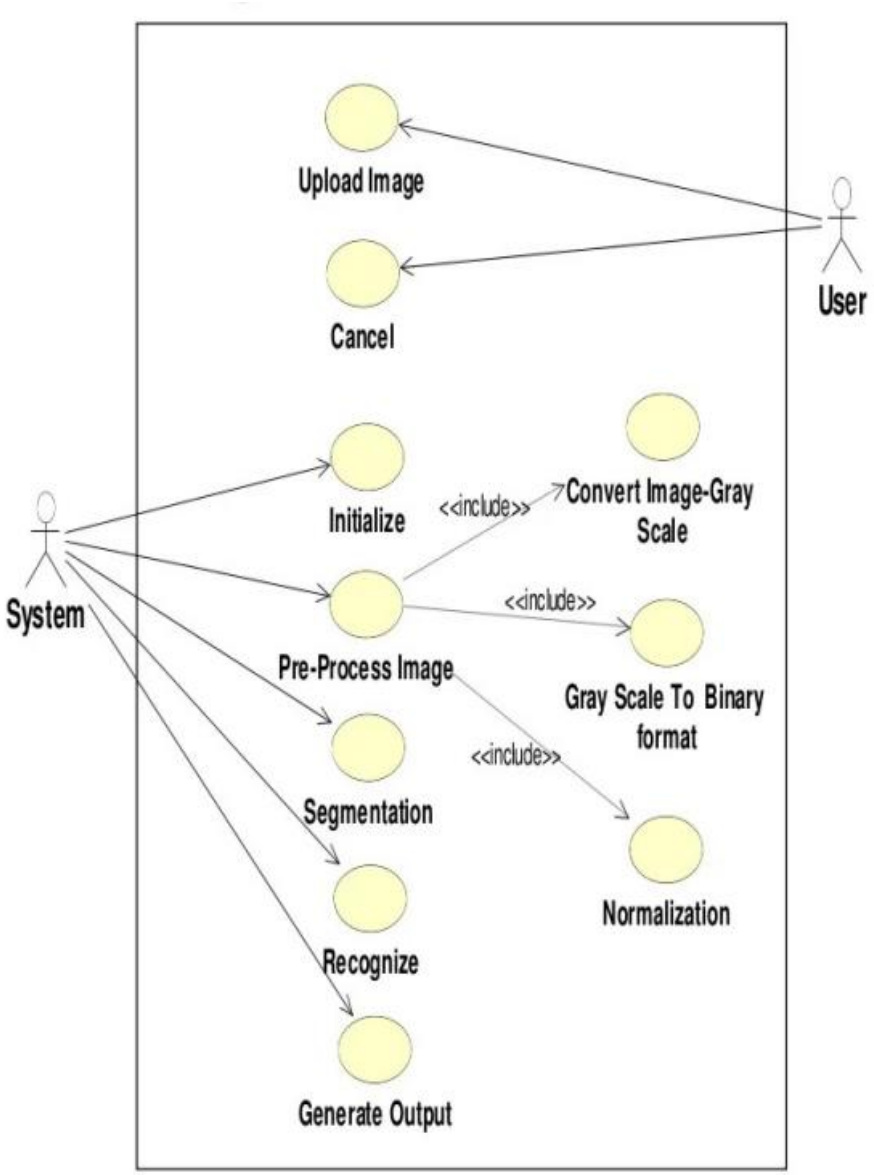
USECASE DIAGRAM:

These internal and external agents are known as actors. Use case diagrams consists of actors, use cases and their relationships. The diagram issued to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system .Hence to model the entire system, a number of use case diagrams are used.

ACTORS IN OUR HANDWRITTEN TEXT RECOGNITION PROJECT ARE:

User: User uploads an handwritten text image and obtains the recognized text from the image.

System: Actor pre-process the data, perform model training and provides the user recognized text



INTERACTION DIAGRAMS:

From the term Interaction, it is clear that the diagram is used to describe some type of interactions among the different elements in the model. This interaction is a part of dynamic behavior of the system.

This interactive behavior is represented in UML by two diagrams known as **Sequence diagram** and **Collaboration diagram**. The basic purpose of both the diagrams are similar.

Sequence diagram emphasizes on time sequence of messages and collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

The purpose of interaction diagrams is to visualize the interactive behavior of the system. Visualizing the interaction is a difficult task. Hence, the solution is to use different types of models to capture the different aspects of the interaction.

Sequence and collaboration diagrams are used to capture the dynamic nature but from a different angle.

The purpose of interaction diagram is –

- To capture the dynamic behaviour of a system.
- To describe the message flow in the system.
- To describe the structural organization of the objects.
- To describe the interaction among objects.

The main purpose of both the diagrams are similar as they are used to capture the dynamic behavior of a system. However, the specific purpose is more important to clarify and understand.

Sequence diagrams are used to capture the order of messages flowing from one object to another.

Collaboration diagrams are used to describe the structural organization of the objects taking part in the interaction. A single diagram is not sufficient to describe the dynamic aspect of an entire system, so a set of diagrams are used to capture it as a whole.

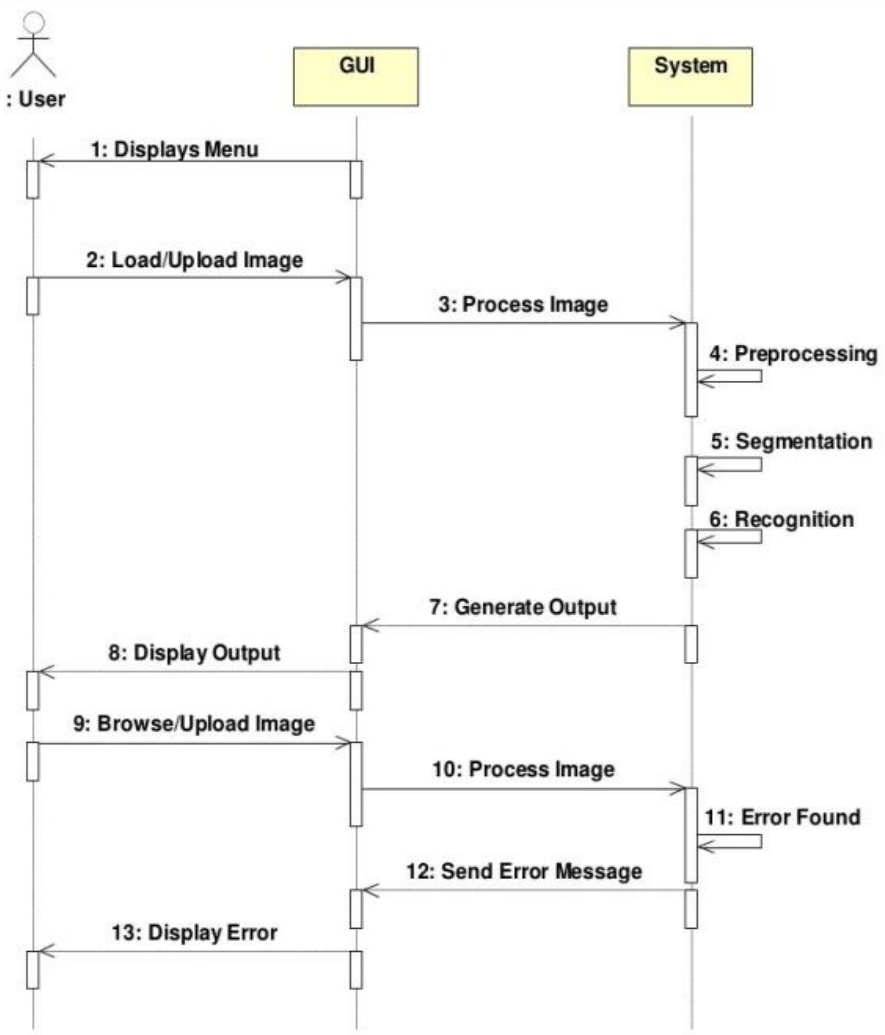
Interaction diagrams are used when we want to understand the message flow and the structural organization. Message flow means the sequence of control flow from one object to another. Structural organization means the visual organization of the elements in a system.

Interaction diagrams can be used –

- To model the flow of control by time sequence.
- To model the flow of control by structural organizations.
- For forward engineering.

SEQUENCE DIAGRAM:

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

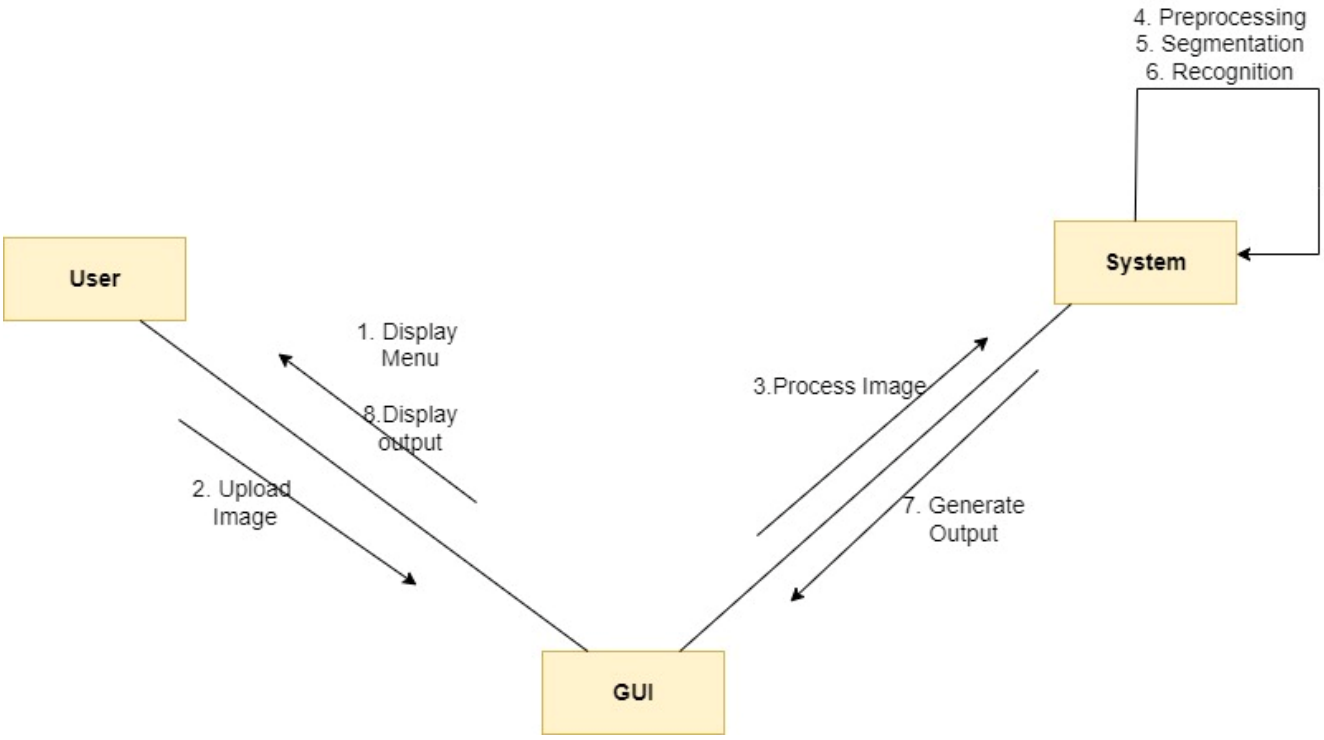


COLLABORATION DIAGRAM:

The second interaction diagram is the collaboration diagram. It shows the object organization as seen in the following diagram. In the collaboration diagram, the method call sequence is indicated by some numbering technique. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram.

Method calls are similar to that of a sequence diagram. However, difference being the sequence diagram does not describe the object organization, whereas the collaboration diagram shows the object organization.

To choose between these two diagrams, emphasis is placed on the type of requirement. If the time sequence is important, then the sequence diagram is used. If organization is required, then collaboration diagram is used.



ACTIVITY DIAGRAM:

Activity diagram is defined as a UML diagram that focuses on the execution and flow of the behavior of a system instead of implementation. It is also called **object-oriented flowchart**. Activity diagrams consist of activities that are made up of actions which apply to behavioralmodeling technology.

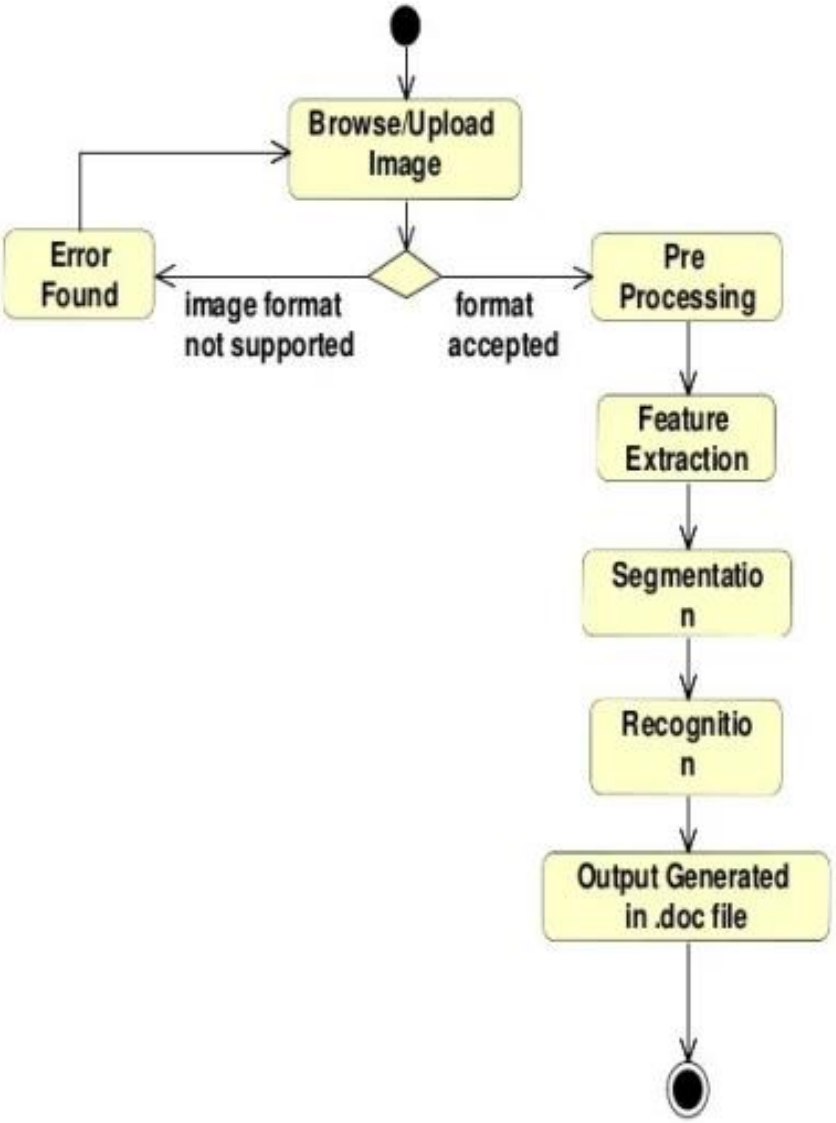
Activity diagrams are used to model processes and workflows. The essence of a useful activity diagram is focused on communicating a specific aspect of a system's dynamic behavior. Activity diagrams capture the dynamic elements of a system.

Activity diagram is similar to a flowchart that visualizes flow from one activity to another activity. Activity diagram is identical to the flowchart, but it is not a flowchart. The flow of activity can be controlled using various control elements in the UML diagram. In simple words, an activity diagram is used to activity diagrams that describe the flow of execution between multiple activities.

Activity Diagram Notations: -

Activity diagrams symbol can be generated by using the following notations:

- Initial states: The starting stage before an activity takes place is depicted as the initial state
- Final states: The state which the system reaches when a specific process ends is known as a Final State
- State or an activity box:
- Decision box: It is a diamond shape box which represents a decision with alternate paths. It represents the flow of control.


Browse/Upload Image
Error Found
image format not supported
format accepted
Pre Processing
Feature Extraction
Segmentatio n
Recognitio n
Output Generated in .doc file

5.SYSTEM IMPLEMENTATION

5.1 INTRODUCTION:

- A lot of important knowledge gets lost or does not get reviewed because of the fact that documents never get transferred to digital format.
- We have thus decided to tackle this problem in our project because we believe the significantly greater ease of management of digital text compared to written text will help people more effectively access, search, share, and analyze their records, while still allowing them to use their preferred writing method.

5.2 PROJECT MODULES:

- **Data Preprocessing:**

- First we will look into the data set for preprocessing. We have searched for the best dataset in order to get better results and preprocesss the data. Dataset we selected is IAM Handwriting Database. The dataset 13,353 images of handwritten text written by 657 writers.
- IAM Dataset we considered having images of individual words and images of single line sentences. We will preprocess the image starting from converting to gray scale image, removing background noise, proper filtering and smoothing of the image.

- **Model Building:**

- **CNN**: 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 5×5 in the first two layers and 3×3 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 32×256.

- **RNN**: 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 32×80. 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.

- **CTC**: 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.

- **Data**
- **Input**: it is a gray-value image of size 128×32. Usually, the images from the dataset do not have exactly this size, therefore we resize it until it either has a width of 128 or a height of 32. Then, we copy the image into a target image of size 128×32. Finally, we normalize the gray-values of the image 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.

- **Model Evaluation:**

- **CNN output**: Output of the CNN layers. 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).

- **RNN output**: The RNN output matrix 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:;?ABCDEFGHIJKLMNO PQRSTUVWXYZabcdefghijklmnopqrstuvwxyz". 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: we just take the most probable character from each time-step, this forms the so called best path, then we throw away repeated characters and finally all blanks: "l---ii--t-t--l-…-e" → "l---i--t-t--l-…-e" → "little".

- **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

- **CNN**
- For each CNN layer, create a kernel of size k×k to be used in the convolution operation.

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

- Then, feed the result of the convolution into the RELU operation and then again to the pooling layer with size px×py and step-size sx×sy.
- 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)

- 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 32×256, which we later concatenate along the feature-axis to form a sequence of size 32×512. Finally, it is mapped to the output sequence (or matrix) of size 32×80 which is fed into the CTC layer.

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

- **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.

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

- 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)

- **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.

- **Improving the model**

- Data augmentation: increase dataset-size by applying further (random) transformations to the input images
- Remove cursive writing style in the input images
- Increase input size (if input of NN is large enough, complete text-lines can be used)
- Adding more CNN layers
- Replacing LSTM by 2D-LSTM
- Decoder: use token passing or word beam search decoding to constrain the output to dictionary words
- Text correction: if the recognized word is not contained in a dictionary, search for the most similar one

- **Conclusion**
- The NN consists of 5 CNN and 2 RNN layers and outputs a character-probability matrix. This matrix is either used for CTC loss calculation or for CTC decoding. An implementation using TF is provided and some important parts of the code were presented. Finally, hints to improve the recognition accuracy were given.

- ## Model Deployment:
- We integrated this model with web flask for better user experience(GUI).

5.3 SCREENS:

index.html | upload.py | DataLoader.py | lines.txt

```python
import os
from datetime import datetime
from flask import Flask, request, render_template, send_from_directory
from main import infer_by_web
app = Flask(__name__)
APP_ROOT = os.path.dirname(os.path.abspath(__file__)) # project abs path

@app.route("/")
def index():
    return render_template("index.html")


@app.route("/upload_page", methods=["GET"])
def upload_page():
    return render_template("upload.html")


@app.route("/upload", methods=["POST"])
def upload():
    # folder_name = request.form['uploads']
    target = os.path.join(APP_ROOT, 'static/')
    print(target)
    if not os.path.isdir(target):
        os.mkdir(target)
    print(request.files.getlist("file"))
    option = request.form.get('optionsPrediction')
    print("Selected Option:: {}".format(option))
    for upload in request.files.getlist("file"):
        print(upload)
        print("{} is the file name".format(upload.filename))
        filename = upload.filename
        # This is to verify files are supported
        ext = os.path.splitext(filename)[1]
        if (ext == ".jpg") or (ext == ".png"):
            print("File supported moving on...")
        else:
            render_template("Error.html", message="Files uploaded are not supported...")
        savefname = datetime.now().strftime('%Y-%m-%d_%H_%M_%S') + "."+ext
        destination = "/".join([target, savefname])
        print("Accept incoming file:", filename)
        print("Save it to:", destination)
        upload.save(destination)
        result = predict_image(destination, option)
        print("Prediction: ", result)
    # return send_from_directory("images", filename, as_attachment=True)
    return render_template("complete.html", image_name=savefname, result=result)


def predict_image(path, type):
    print(path)
    return infer_by_web(path, type)


if __name__ == "__main__":
    app.run(port=4555, debug=True)
```

Goto Tools Project Preferences Help

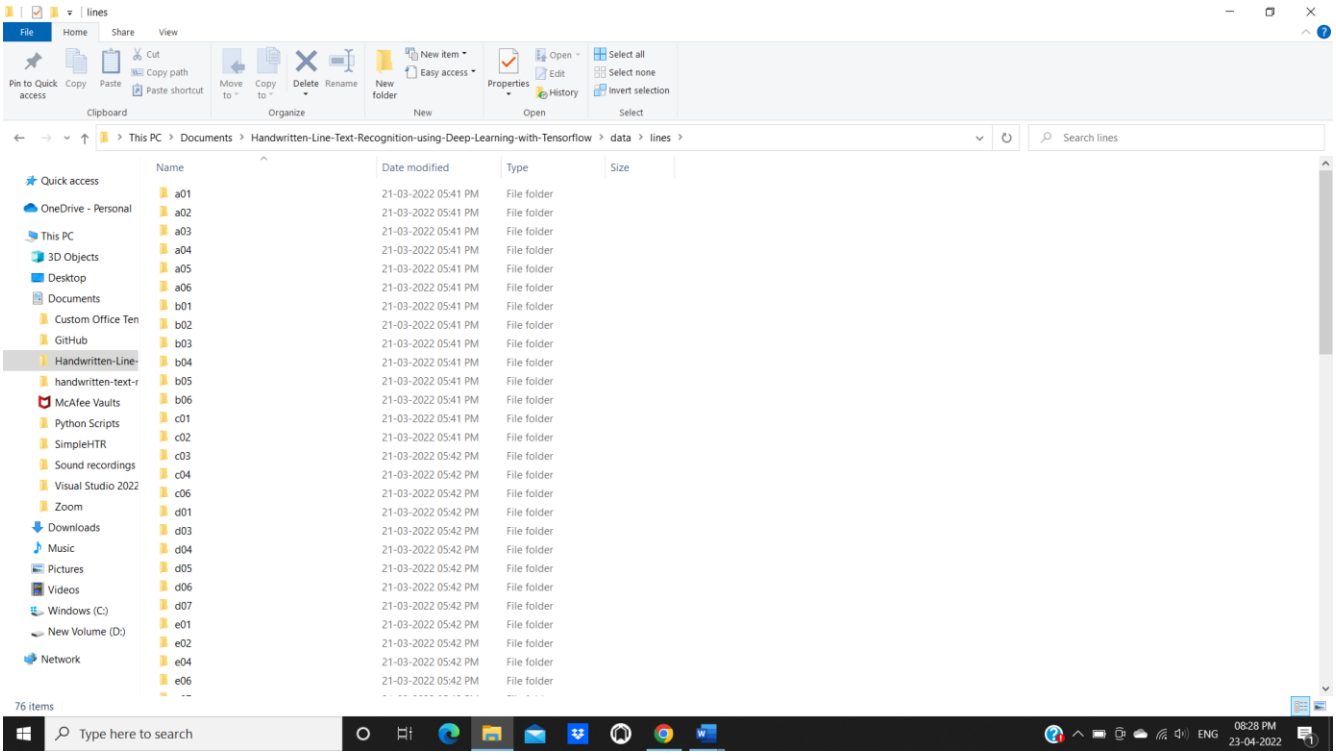
index.html | upload.html | upload.py | main.py | charList.txt | complete.html | SamplePreprocessor.py

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>HomePage</title>
    <link rel="stylesheet" type="text/css" href="{{url_for('static',filename='css/bootstrap.min.css') }}">
    <link rel="stylesheet" type="text/css" href="{{url_for('static',filename='css/jumbotron.css') }}">

</head>
</head>
<body>

<div class="container-fluid">
    {% include "logo_include.html" %}

    <div class="header clearfix">
        <nav>
          <ul class="nav nav-pills pull-right">
            <li role="presentation" class="active"><a href="#">Home</a></li>
            <li role="presentation"><a href="#">About</a></li>
            <li role="presentation"><a href="#">Contact</a></li>
          </ul>
        </nav>
        <h3 class="text-muted">Handwritten Line Text Recognition</h3>
    </div>
    <div class="jumbotron">
        <h1>Handwritten Text Recognition</h1>
        <p class="lead">Input the Handwritten Text here as file upload and convert into digital text.</p>
                <p><a class="btn btn-lg btn-success" href="upload_page" role="button">INPUT HERE</a></p>

    </div>
</div>

<footer>
    <script src="{{url_for('static', filename='js/jquery.js')}}"></script>
    <script src="{{url_for('static', filename='js/bootstrap.min.js')}}"></script>
</footer>
</body>
</html>
```

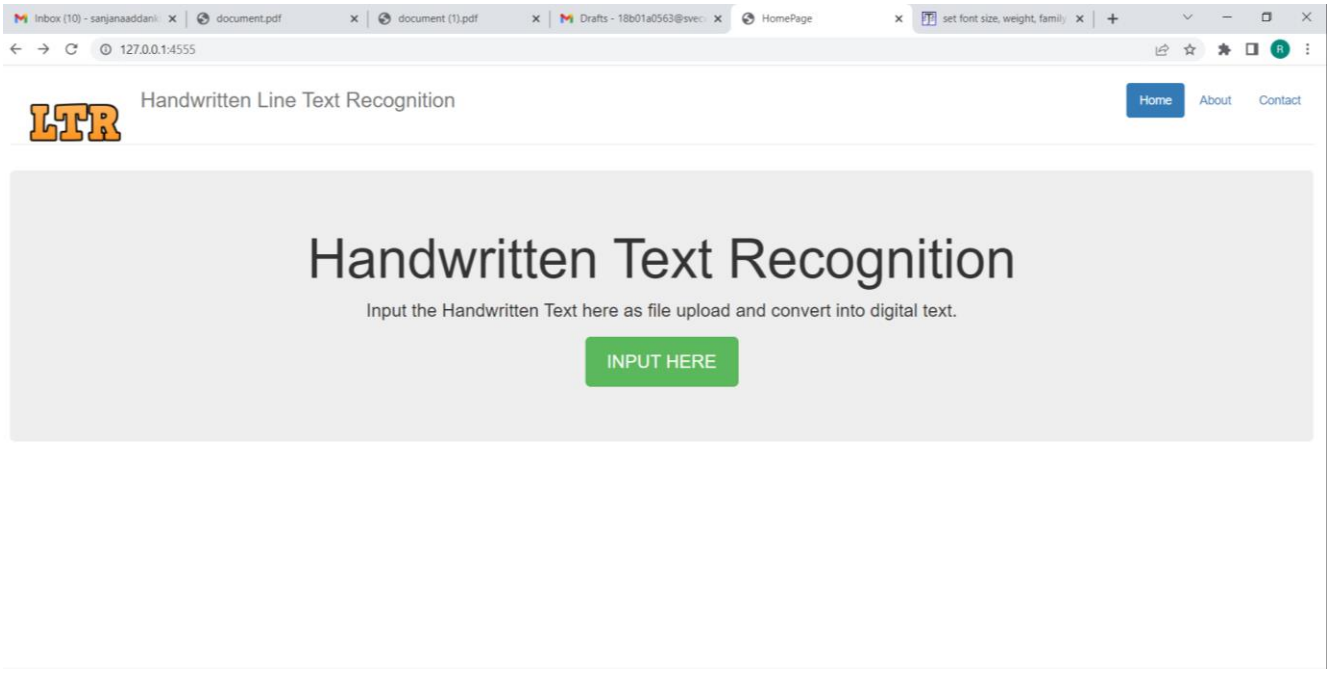
index.html | upload.html | upload.py | main.py | charList.txt | complete.html | SamplePreprocessor.py

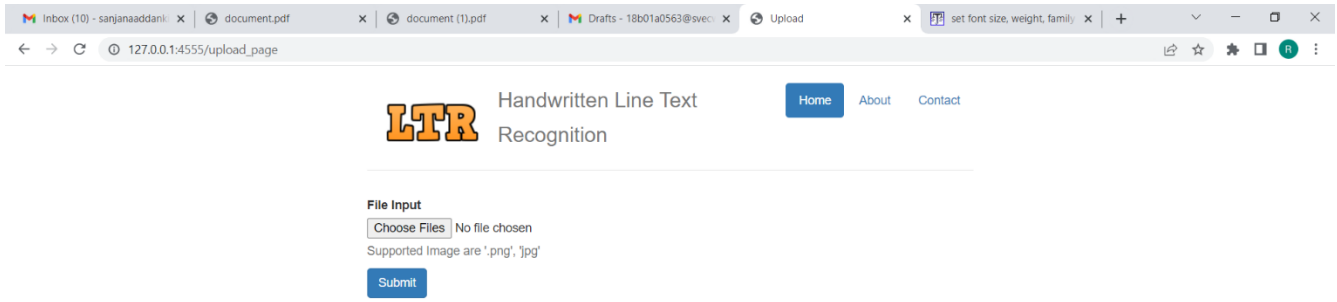
```
<head>
    <meta charset="UTF-8">
    <title>Complete Show</title>
    <link rel="stylesheet" type="text/css" href="{{url_for('static',filename='css/bootstrap.css') }}">
    <script src="{{url_for('static', filename='js/jquery.js')}}"></script>
    <script src="{{url_for('static', filename='js/bootstrap.min.js')}}"></script>

</head>
<style>
textarea {
 width: 300px;
 height: 100px;
 background-color: yellow;
 font-size: 2em;
 font-weight: bold;
 font-family: Verdana, Arial, Helvetica, sans-serif;
 border: 1px solid black;
}
</style>
<body>
    <div class="container">
        <div class="alert alert-warning alert-dismissible fade show" role="alert">
          <strong>Congrats!</strong> The file <b></b> is uploaded Sucessful.
          <button type="button" class="close" data-dismiss="alert" aria-label="Close">
            <span aria-hidden="true">&times;</span>
          </button>
        </div>
        <!--Static Folder is predefined in Jinja-->
        <!--<img src="{{url_for('static', filename=image_name)}} ">-->
        <figure class="figure">
          <img src="{{url_for('static', filename=image_name)}} " class="figure-img img-fluid rounded" alt="Upload Image." width="300" height="300">
          <figcaption class="figure-caption">Recently Uploaded Photo.</figcaption>
        </figure>
        <p class="bg-success">
            <b>Recognized Text</b><br>
        </p>
        <textarea class="form-control" id="textPred" rows="10" cols="24"></textarea>

    </div>
</body>
<script>
    document.getElementById("textPred").value = `{{result}}`


</script>

</html>
```

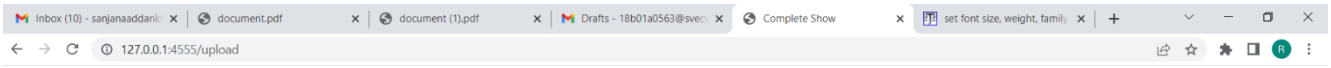
DATA SET


lines
File
Home
Share
View
Pin to Quick access
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New folder
New item
Easy access
New
Properties
Open
Edit
History
Open
Select all
Select none
Invert selection
Select
This PC > Documents > Handwritten-Line-Text-Recognition-using-Deep-Learning-with-Tensorflow > data > lines >
Search lines
Quick access
OneDrive - Personal
This PC
3D Objects
Desktop
Documents
Custom Office Ten
GitHub
Handwritten-Line-
handwritten-text-r
McAfee Vaults
Python Scripts
SimpleHTR
Sound recordings
Visual Studio 2022
Zoom
Downloads
Music
Pictures
Videos
Windows (C:)
New Volume (D:)
Network
Name
Date modified
Type
Size
a01 21-03-2022 05:41 PM File folder
a02 21-03-2022 05:41 PM File folder
a03 21-03-2022 05:41 PM File folder
a04 21-03-2022 05:41 PM File folder
a05 21-03-2022 05:41 PM File folder
a06 21-03-2022 05:41 PM File folder
b01 21-03-2022 05:41 PM File folder
b02 21-03-2022 05:41 PM File folder
b03 21-03-2022 05:41 PM File folder
b04 21-03-2022 05:41 PM File folder
b05 21-03-2022 05:41 PM File folder
b06 21-03-2022 05:41 PM File folder
c01 21-03-2022 05:41 PM File folder
c02 21-03-2022 05:41 PM File folder
c03 21-03-2022 05:42 PM File folder
c04 21-03-2022 05:42 PM File folder
c06 21-03-2022 05:42 PM File folder
d01 21-03-2022 05:42 PM File folder
d03 21-03-2022 05:42 PM File folder
d04 21-03-2022 05:42 PM File folder
d05 21-03-2022 05:42 PM File folder
d06 21-03-2022 05:42 PM File folder
d07 21-03-2022 05:42 PM File folder
e01 21-03-2022 05:42 PM File folder
e02 21-03-2022 05:42 PM File folder
e04 21-03-2022 05:42 PM File folder
e06 21-03-2022 05:42 PM File folder
76 items
Type here to search
ENG
08:28 PM
23-04-2022

OUTPUT SCREENS



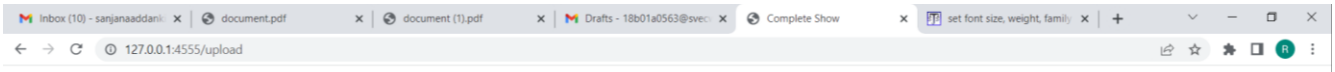




Recently Uploaded Photo.

Recognized Text

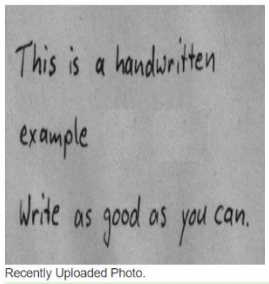
Batch 14 projects



Recently Uploaded Photo.

Recognized Text

Final yea projects

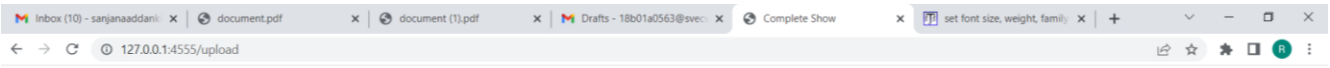


Recently Uploaded Photo.

Recognized Text

This is a handwritten
example

Write as qooal as you can.



Recently Uploaded Photo.

Recognized Text

Final yea SVE CW

6. SYSTEM TESTING

6.1. INTRODUCTION:

Software Testing is an important element of the software quality assurance and represents the ultimate review of specification, design and coding. The increasing feasibility of software as a system and the cost associated with the software failures are motivated forces for III planned through testing.

TESTING OBJECTIVES

These are several rules that can save as testing objectives:

- Testing is a process of executing program with the intent of finding an error.
- A good testcase is one that has a high probability of finding an undiscovered error.

TEST LEVELS

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or darkness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product. Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

6.2. TESTING METHODS:

6.2.1 Unit Testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application.

6.2.2 Integration Testing

Integration tests are designed to test integrated software components to determine if they run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields.

6.2.3 Functional Testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals. Organization and preparation of functional tests is focused on requirements, key functions, or special test cases.

6.2.4 System Testing

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test.

6.2.5 White Box Test

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

6.2.6 Black Box Test

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document.

6.2.7 Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

6.2.8 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

6.2.9 Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user.

7.Conclusion

Handwritten text recognition using deep learning techniques, as CNN performs well for recognition and with the aid of hyper parameter tuning, accuracy or recognition rate can be improved. As a result, in the proposed scheme, we used CNN for handwritten text recognition. The images will be taken with a camera scanning the handwritten text during the image acquisition stage and the recognition process will be done using the CNN algorithm after preprocessing. The machine results with a recognized text.

8.BIBLIOGRAPHY

DATA SET:

https://fki.tic.heia-fr.ch/databases/iam-handwriting-database

REFERENCES :

[1]. Chirag I Patel, Ripal Patel, Palak Patel, "Handwritten Character Recognition Using Neural Networks", International Journal of Scientific & Engineering Research Volume 2, Issue 5, May-2011.

[2]. Kauleshwar Prasad, Devvrat C Nigam, AshmikaLakhotiya, DheerenUmre. "Character Recognition Using Matlab's Neural Toolbox", International Journal of u- and e- Serv ice, Science and Technology Vol. 6, No. 1, Fcbruary, 2013.

[3]. AshutoshAggarwal, Rajneesh Rant, RenuDhir, "Handwritten Character Recognition Using Gradient Features", International Journal or" Advanced Research in Computer Science and Software Engineering, Volume 2, Issue 5, May 2012.

[4]. Vinita Dutt, Sunil Dutt, "Handwritten Character Recognition Using Artificial Neural Network", Advances in Computing: 2011; l1(1): 18-23.

[5]. Rahul Kala, Harsh Vazirani, AnupamShukla, RiluTiwari, "Offline Handwriting Recognition", International Journal or' Computer Science issues, volume 7, March- 2010.

[6]. Dinesh Dileep, "A Feature Extraction Technique Based on Character Geometry for Character Recognition".

[7]. Alexander J. Faaborg, "Using Neural Networks to Create an Adaptive Character Recognition System", Cornell University, Ithaca NY, (May 14, 2002).

[8]. Swapnil A. Vaidya, Balaji R. Bombade "A Novel Approach of Handwritten Character Recognition using Positional Feature Extraction",IJCSMC, Vol. 2, Issue. 6, June 2013.

Sheng Wang "A Review or" Gradient-Based and Edge-Based Feature Extraction Methods ior Object Detection",Computer and Information Technology (CIT), 2011 IEEE 11 th International Conference.

9.APPENDIX

9.1 Python

Python is an open source, high-level programming language developed by Guido van Rossum in the late 1980s and presently administered by Python Software Foundation. It came from the ABC language that he helped create early on in his career. Python is a powerful language that you can use to create games, write GUIs, and develop web applications.

It is a high-level language. Reading and writing codes in Python is much like reading and writing regular English statements. Because they are not written in machine-readable language, Python programs need to be processed before machines can run them. Python is an interpreted language. This means that every time a program is run, its interpreter runs through the code and translates it into machine readable byte code.

Python is an object-oriented language that allows users to manage and control data structures or objects to create and run programs. Everything in Python is, in fact, first class. All objects, data types, functions, methods, and classes take equal position in Python. Programming languages are created to satisfy the needs of programmers and users for an effective tool to develop applications that impact lives, lifestyles, economy, and society. They help make lives better by increasing productivity, enhancing communication, and improving efficiency. Languages die and become obsolete when they fail to live up to expectations and are replaced and superseded by languages that are more powerful. Python is a programming language that has stood the test of time and has remained relevant across industries and businesses and among programmers, and individual users. It is a living, thriving, and highly useful language that is highly recommended as a first programming language for those who want to dive into and experience programming. Advantages of Using Python Here are reasons why you would prefer to learn and use Python over other high-level languages:

Readability

Python programs use clear, simple, and concise instructions that are easy to read even by those who have no substantial programming background. Programs written in Python are, therefore, easier to maintain, debug, or enhance.

Higher productivity

Codes used in Python are considerably shorter, simpler, and less verbose than other high-level programming languages such as Java and C++. In addition, it has well-designed built-in features and standard library as well as access to third party modules and source libraries.
These features make programming in Python more efficient.

Less learning time

Python is relatively easy to learn. Many find Python a good first language for learning programming because it uses simple syntax and shorter codes. Python works on Windows, Linux/UNIX, Mac OS X, other operating systems and small form devices. It also runs on microcontrollers used in appliances, toys, remote controls, embedded devices, and other similar devices.

9.2 Deep Learning

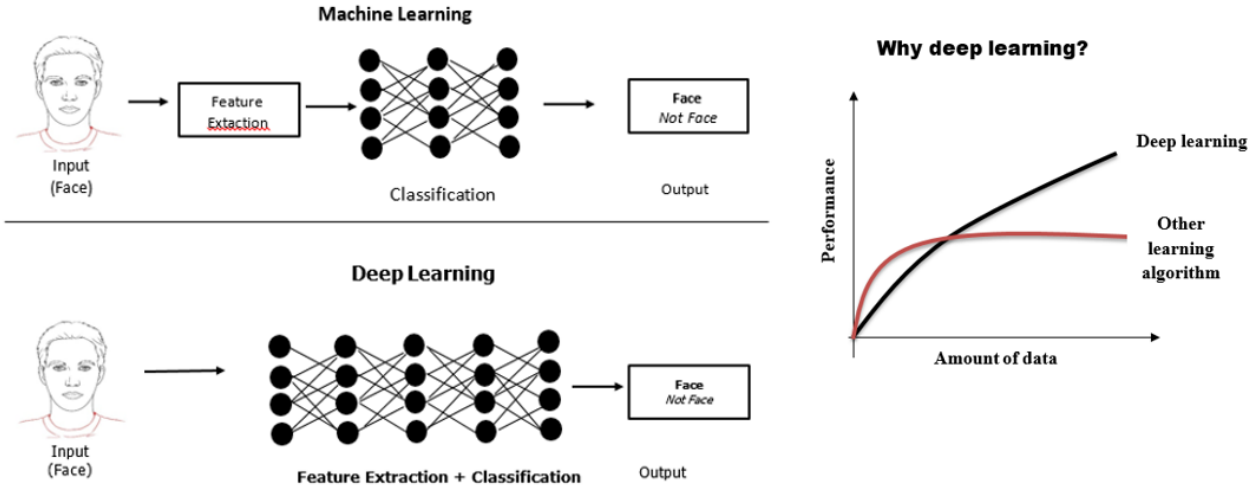


Fig1: Deep Learning vs Machine Learning & Deep Learning performance with Other Algorithm.

Machine Learning needs *early Feature Extraction as features* and performed classification on it. But Deep Learning acts as a "black box" which do feature extraction and classification on its own. In above figure shows that the main task is to classify the given image as face or non-face. In the case of machine learning, it needs a feature of images such as edges, color, shape, etc., and performs classification on its own.

But deep learning extracts feature and perform classification on its own. Example of this given image to the Convolutional Neural Networks that every layer of CNN takes the feature and finally Fully Connected Layer perform classification

The main conclusion is Deep Learning self extracts features with deep neural networks and classifies itself. Compare to traditional Algorithms its performance increase with the Amount of Data. This article is all for building your own handwritten recognition system with TensorFlow. It covers detailed intuition about architecture and how I reach the solution and increase accuracy. Note that you can build a handwritten recognition system in any language where the architecture remains the same. In the end, you can build a handwritten recognition system in your own language provided the dataset to it. Let's get started!

Basic Intuition on How it Works.

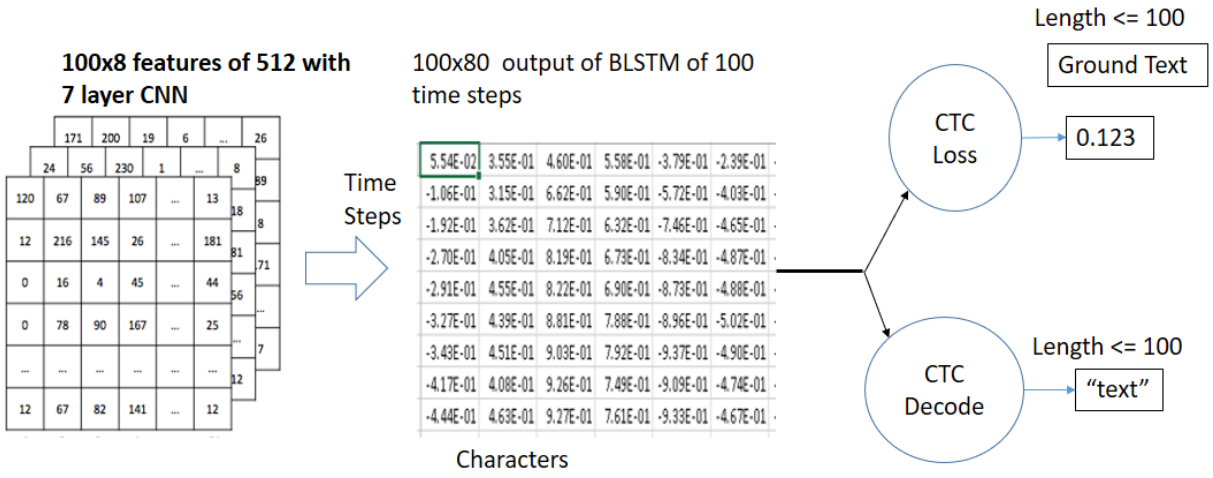


Fig2: Visualization Output from each layer

- *First Use Convolutional Recurrent Neural Network to extract the important features from the handwritten line text Image.*
- *The output before the CNN FC layer (512x100) is passed to the BLSTM which is for sequence dependency and time-sequence operations. The output of BLSTM is 100x80 i.e 100 timesteps and 80 characters including blank.*
- *Then CTC LOSS Alex Graves is used to train the RNN which eliminates the Alignment problem in Handwritten since handwritten have a different alignment of every writer. We just gave them what is written in the image (Ground Truth Text) and BLSTM output, then it calculates loss simply as log("gtText"); aim to minimize the negative maximum likelihood path.*
- *Then, CTC finds out the possible paths from the given labels. Loss is given by for (X, Y) pair is:*

$$p(Y \mid X) = \sum_{A \in \mathcal{A}_{X,Y}} \prod_{t=1}^{T} p_t(a_t \mid X)$$

The CTC conditional **probability** | **marginalizes** over the set of valid alignments | computing the **probability** for a single alignment step-by-step.

Fig3: CTC loss formula

- *Finally, CTC Decode is used to decode the output during Prediction.*

Detail Architecture

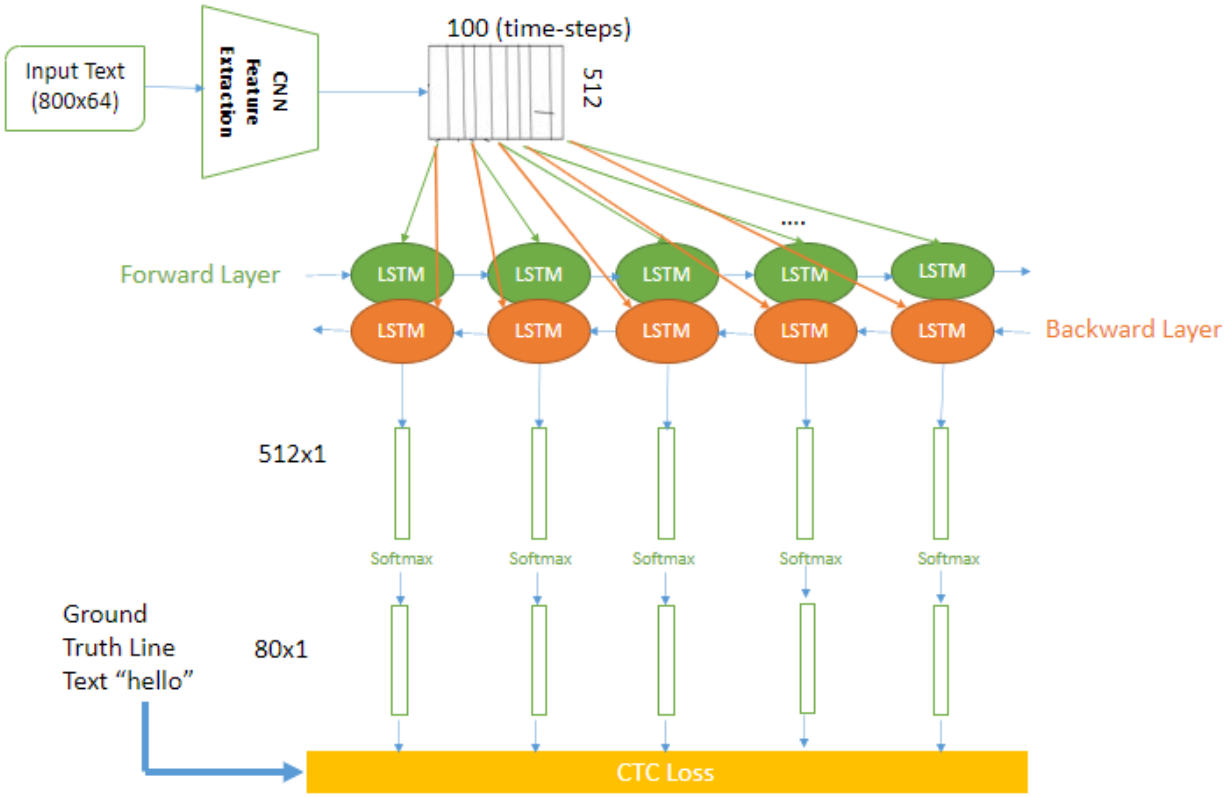


Fig4: Detail Architecture

Altogether there are 3 steps:

1. Multi-scale feature Extraction → Convolutional Neural Network 7 Layers
2. Sequence Labeling (BLSTM-CTC) → Recurrent Neural Network (2 layers of LSTM) with CTC
3. Transcription → Decoding the output of the RNN (CTC decode)

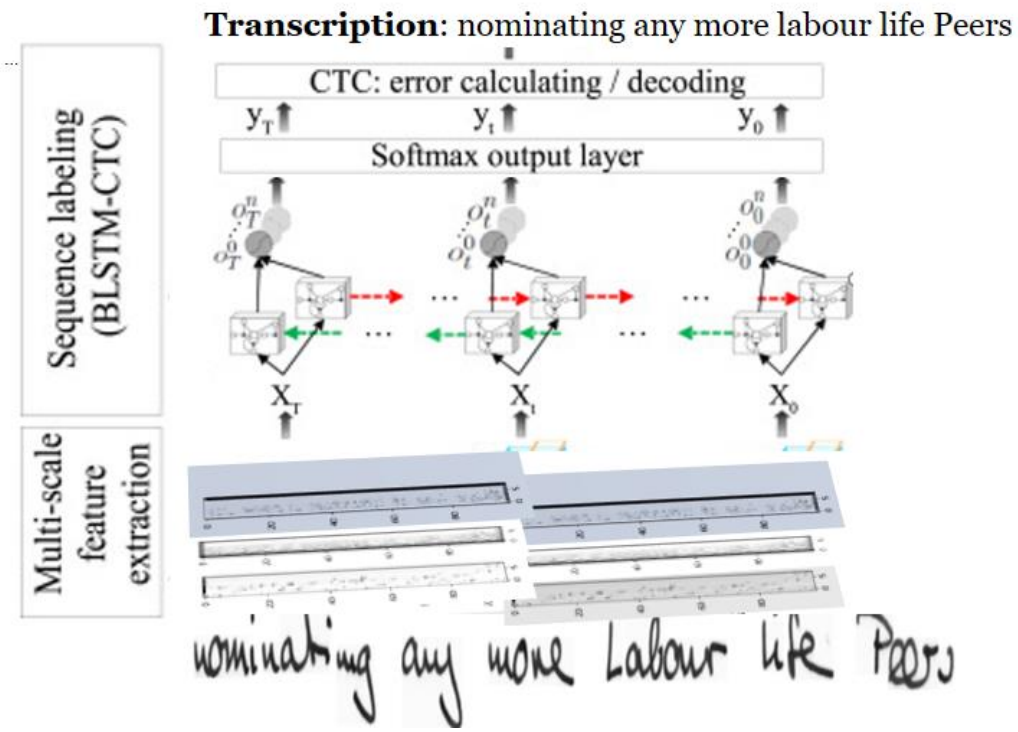


Fig5: 3 steps used in handwritten recognition

In deep learning, a **convolutional neural network** (**CNN/ConvNet**) is a class of deep neural networks, most commonly applied to analyze visual imagery. Now when we think of a neural network we think about matrix multiplications but that is not the case with ConvNet. It uses a special technique called Convolution. Now in mathematics **convolution** is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.

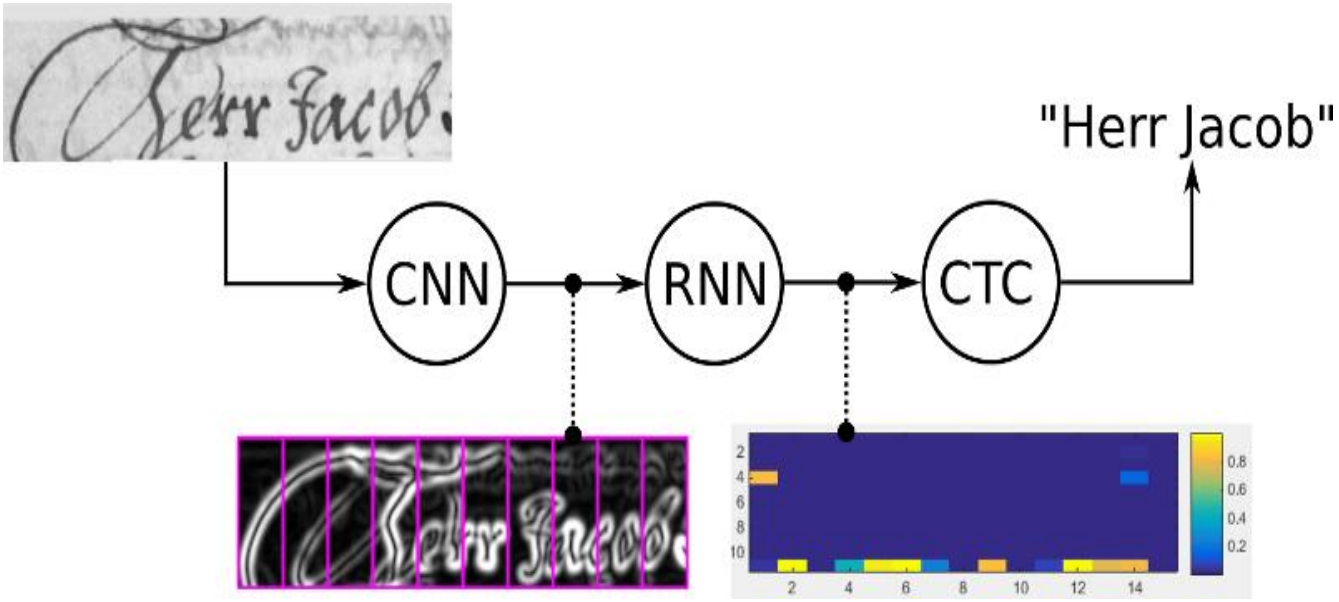
Convolutional neural networks are composed of multiple layers of artificial neurons. Artificial neurons, a rough imitation of their biological counterparts, are mathematical functions that calculate the weighted sum of multiple inputs and outputs an activation value. When you input an image in a ConvNet, each layer generates several activation functions that are passed on to the next layer.

The first layer usually extracts basic features such as horizontal or diagonal edges. This output is passed on to the next layer which detects more complex features such as corners or combinational edges. As we move deeper into the network it can identify even more complex features such as objects, faces, etc.

Based on the activation map of the final convolution layer, the classification layer outputs a set of confidence scores (values between 0 and 1) that specify how likely the image is to belong to a "class." For instance, if you have a ConvNet that detects cats, dogs, and horses, the output of the final layer is the possibility that the input image contains any of those animals.

9.3 Connectionist Temporal Classification

Let's have a closer look at the CTC operation and discuss how it works without hiding the clever ideas it is based on behind complicated formulas. At the end, I will point you to references where you can find Python code and the formulas.



We could, of course, create a data-set with images of text-lines, and then specify for each horizontal position of the image the corresponding character as shown in the figure. Then, we could train a NN to output a character-score for each horizontal position. However, there are two problems with this naive solution:

it is very time-consuming (and boring) to annotate a data-set on character-level.

we only get character-scores and therefore need some further processing to get the final text from it. A single character can span multiple horizontal positions, e.g. we could get " ttooo" because the "o" is a wide character as shown in Fig. 2. We have to remove all duplicate "t"s and "o"s. But what if the recognized text would have been "too"? Then removing all duplicate "o"s gets us the wrong result. How to handle this?

CTC solves both problems for us:

we only have to tell the CTC loss function the text that occurs in the image. Therefore we ignore both the position and width of the characters in the image.

no further processing of the recognized text is needed.

As already discussed, we don't want to annotate the images at each horizontal position (which we call time-step from now on). The NN-training will be guided by the CTC loss function. We only feed the output matrix of the NN and the corresponding ground-truth (GT) text to the CTC loss function. But how does it know where each character occurs? Well, it does not know. Instead, it tries all possible alignments of the GT text in the image and takes the sum of all scores. This way, the score of a GT text is high if the sum over the alignment-scores has a high value.

Encoding the text

There was the issue of how to encode duplicate characters (you remember what we said about the word "too"?). It is solved by introducing a pseudo-character (called blank, but don't confuse it with a "real" blank, i.e. a white-space character). This special character will be denoted as "-" in the following text. We use a clever coding schema to solve the duplicate-character problem: when encoding a text, we can insert arbitrary many blanks at any position, which will be removed when decoding it. However, we must insert a blank between duplicate characters like in "he**ll**o". Further, we can repeat each character as often as we like.

Let's look at some examples:

- "to" → "---ttttttooo", or "-t-o-", or "to"
- "too" → "---ttttto-o", or "-t-o-o-", or "to-o", but **not** "too"

As you see, this schema also allows us to easily create different alignments of the same text, e.g. "t-o" and "too" and "-to" all represent the same text ("to"), but with different alignments to the image. The NN is trained to output an encoded text (encoded in the NN output matrix).

Loss calculation

We need to calculate the loss value for the training samples (pairs of images and GT texts) to train the NN. You already know that the NN outputs a matrix containing a score for each character at each time-step. A minimalistic matrix is shown in Fig. 3: there are two time-steps (t0, t1) and three characters ("a", "b" and the blank "-"). The character-scores sum to 1 for each time-step.

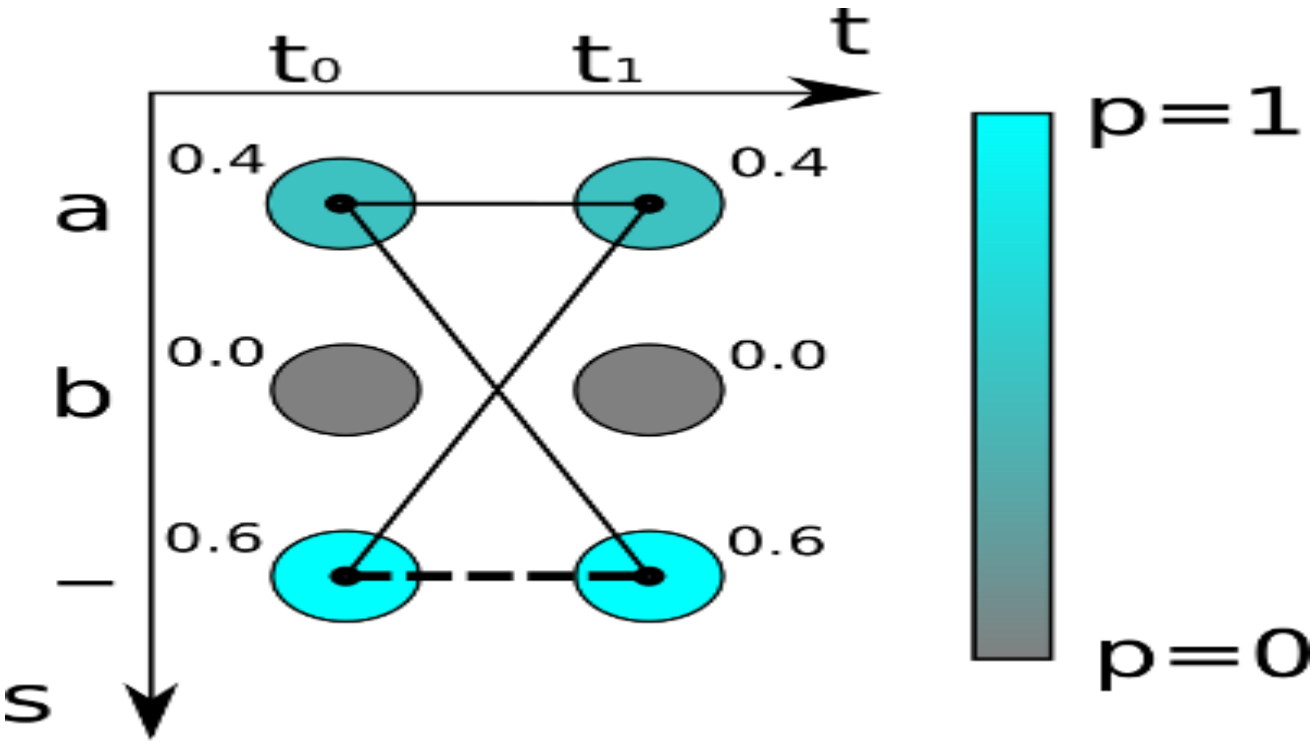


Fig. 3: Output matrix of NN. The character-probability is color-coded and is also printed next to each matrix entry. Thin lines are paths representing the text "a", while the thick dashed line is the only path representing the text "".

Further, you already know that the loss is calculated by summing up all scores of all possible alignments of the GT text, this way it does not matter where the text appears in the image.

The score for one alignment (or **path**, as it is often called in the literature) is calculated by multiplying the corresponding character scores together. In the example shown above, the score for the path "aa" is 0.4·0.4=0.16 while it is 0.4·0.6=0.24 for "a-" and 0.6·0.4=0.24 for "-a". To get the score for a given GT text, we sum over the scores of all paths corresponding to this text. Let's assume the GT text is "a" in the example: we have to calculate all possible paths of length 2 (because the matrix has 2 time-steps), which are: "aa", "a-" and "-a". We already calculated the scores for these paths, so we just have to sum over them and get 0.4·0.4+0.4·0.6+0.6·0.4=0.64. If the GT text is assumed to be "", we see that there is only one corresponding path, namely "--", which yields the overall score of 0.6·0.6=0.36.

We are now able to compute the probability of the GT text of a training sample, given the output matrix produced by the NN. The goal is to train the NN such that it outputs a high probability (ideally, a value of 1) for correct classifications. Therefore, we maximize the product of probabilities of correct classifications for the training dataset. For technical reasons, we re-formulate into an equivalent problem: minimize the loss of the training dataset, where the loss is the negative sum of log-probabilities. If you need the loss value for a single sample, simply compute the probability, take the logarithm, and put a minus in front of the result. To train the NN, the gradient of the loss with respect to the NN parameters (e.g., weights of convolutional kernels) is computed and used to update the parameters.

Decoding

When we have a trained NN, we usually want to use it to recognize text in previously unseen images. Or in more technical terms: we want to calculate the most likely text given the output matrix of the NN. You already know a method to calculate the score of a given text. But this time, we are not given any text, in fact, it is exactly this text we are looking for. Trying every possible text would work if there are only a few time-steps and characters, but for practical use-cases, this is not feasible.

A simple and very fast algorithm is **best path decoding** which consists of two steps:

1. it calculates the best path by taking the most likely character per time-step.
2. it undoes the encoding by first removing duplicate characters and then removing all blanks from the path. What remains represents the recognized text.

An example is shown in Fig. 4. The characters are "a", "b" and "-" (blank). There are 5 time-steps. Let's apply our best path decoder to this matrix: the most likely character of t0 is "a", the same applies for t1 and t2. The blank character has the highest score at t3. Finally, "b" is most likely at t4. This gives us the path "aaa-b". We remove duplicate characters, this yields "a-b", and then we remove any blank from the remaining path, which gives us the text "ab" which we output as the recognized text.

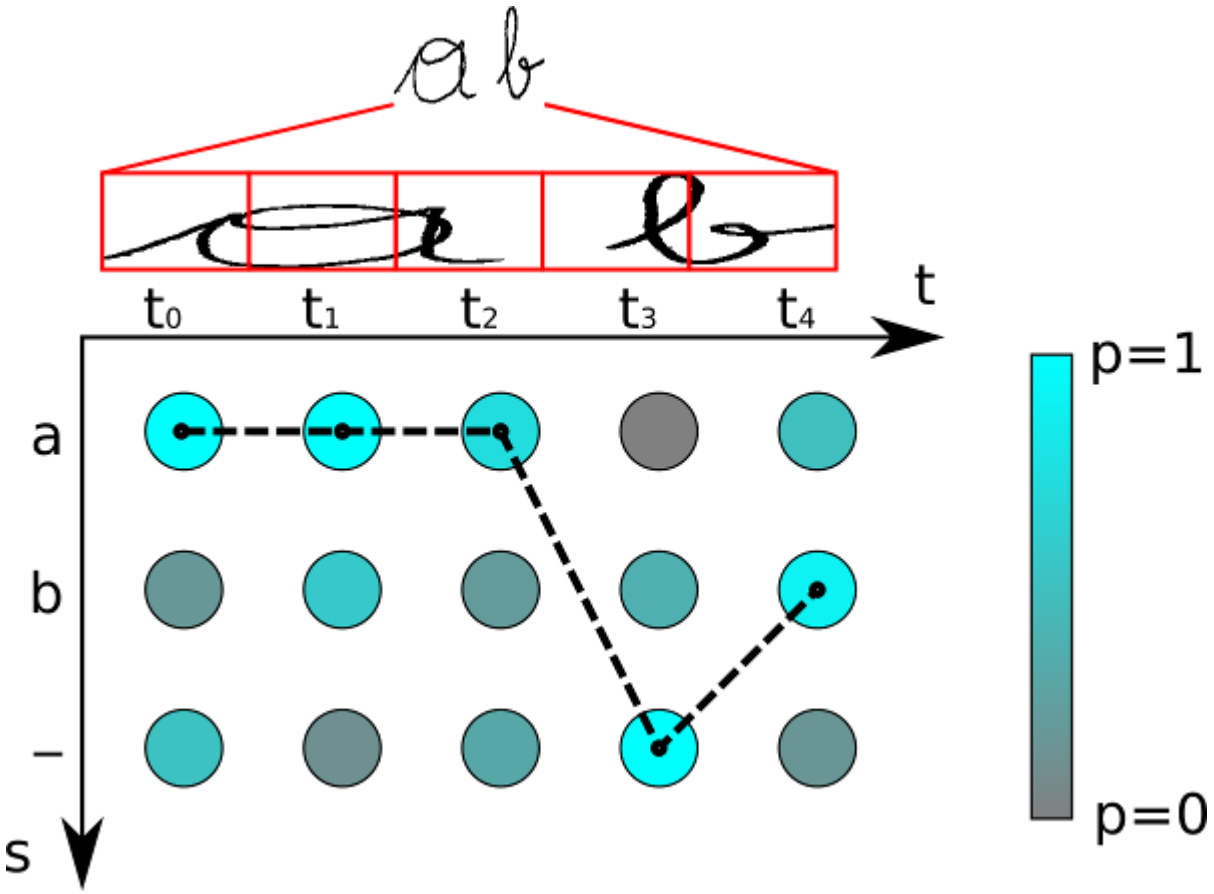


Fig. 4: Output matrix of NN. The thick dashed line represents the best path.

Best path decoding is, of course, only an approximation. It is easy to construct examples for which it gives the wrong result: if you decode the matrix from Fig. 3, you get "" as the recognized text. But we already know that the probability of "" is only 0.36 while it is 0.64 for "a". However, the approximation algorithm often gives good results in practical situations. There are more advanced decoders such as beam-search decoding, prefix-search decoding or token passing, which also use information about language structure to improve the results.

Conclusion

First, we looked at the problems arising with a naive NN solution. Then, we saw how CTC is able to tackle these problems. We then examined how CTC works by looking at how it encodes text, how loss calculation is done and how it decodes the output of a CTC-trained NN.

9.4 Word Beam Search: A CTC Decoding Algorithm

Improve text recognition results: avoid spelling mistakes, allow arbitrary numbers and punctuation marks and make use of a word-level language model

Let's suppose we have a neural network (NN) for text recognition (OCR or HTR) which is trained with the connectionist temporal classification (CTC) loss function. We feed an image containing text into the NN and get the digitized text out of it. The NN simply outputs the characters it sees in the image. This purely optical model might introduce errors because of similar looking characters like "a" and "o".

Problems like the one shown in Fig. 1 will look familiar to you if you have already worked on text-recognition: the result is of course wrong, however, when we look closely at the handwritten text, we can imagine why the NN confuses "a" with "oi" in the word "random".

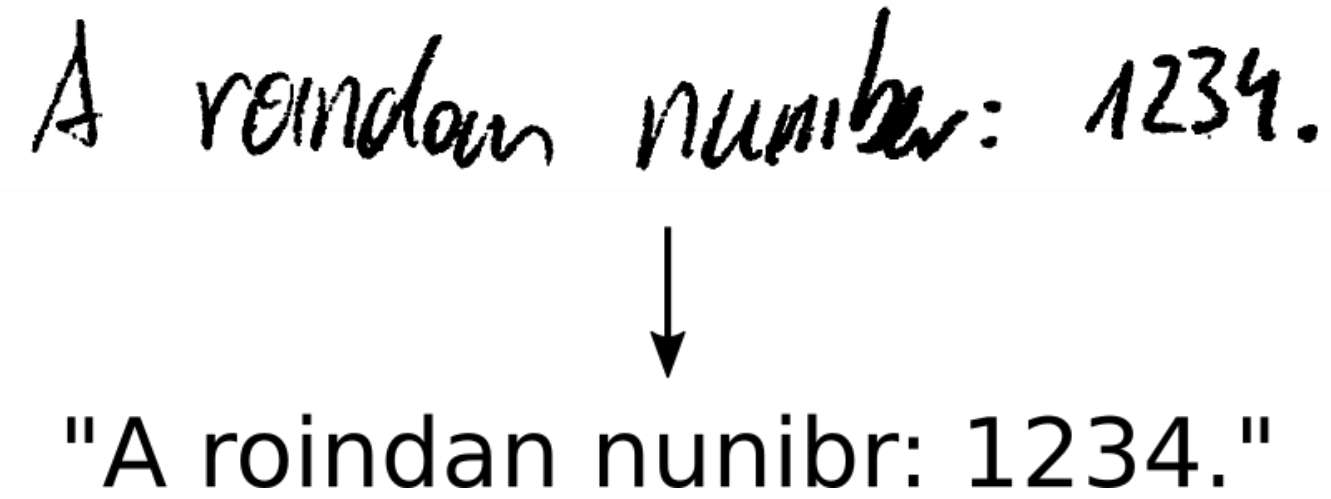


Fig. 1: Why does the NN make such mistakes?

Can we improve our result?

So, how to tackle such situations? There are different decoding algorithms available, some also include a language model (LM). Let's look what the algorithms output for our example:

- **Best path decoding:** "A roindan nunibr: 1234.". This is the result shown in Fig. 1. Best path decoding only uses the output of the NN (i.e. no language model) and computes an approximation by taking the most likely character at each position.

- **Beam search:** "A roindan numbr: 1234.". It also only uses the NN output, but it uses more information from it and therefore produces a more accurate result.
- **Beam search with character-LM:** "A randan number: 1234." It additionally scores character-sequences (e.g. "an" is more probable than "oi") which further improves the result.
- **Token passing:** "A random number". This algorithm uses a dictionary and word-LM. It searches for the most probable sequence of dictionary words in the NN output. But it can't handle arbitrary character sequences (numbers, punctuation marks) like ": 1234.".

None of the algorithms gets it right. But we observe nice properties of beam search and token passing:

- Beam search allows **arbitrary character strings**, which is needed to decode **numbers and punctuation** marks.
- Token passing limits its output to **dictionary words**, which **avoids spelling mistakes**. Of course, the dictionary must contain all words which have to be recognized.

A combination of both properties would be nice: when we see a word, we only allow words from a dictionary, but in any other case, we allow arbitrary character strings. Let's look at an algorithm which implements our desired behavior: word beam search.

Proposed algorithm: Word Beam Search

We use the vanilla beam search algorithm as a starting point. This algorithm iterates through the NN output and creates text candidates (called beams) which are scored. Fig. 2 shows an illustration of the evolution of beams: we start with the empty beam, then add all possible characters (we only have "a" and "b" in this example) to it in the first iteration and only keep the best scoring ones. The beam width controls the number of surviving beams. This is repeated until the complete NN output it processed.

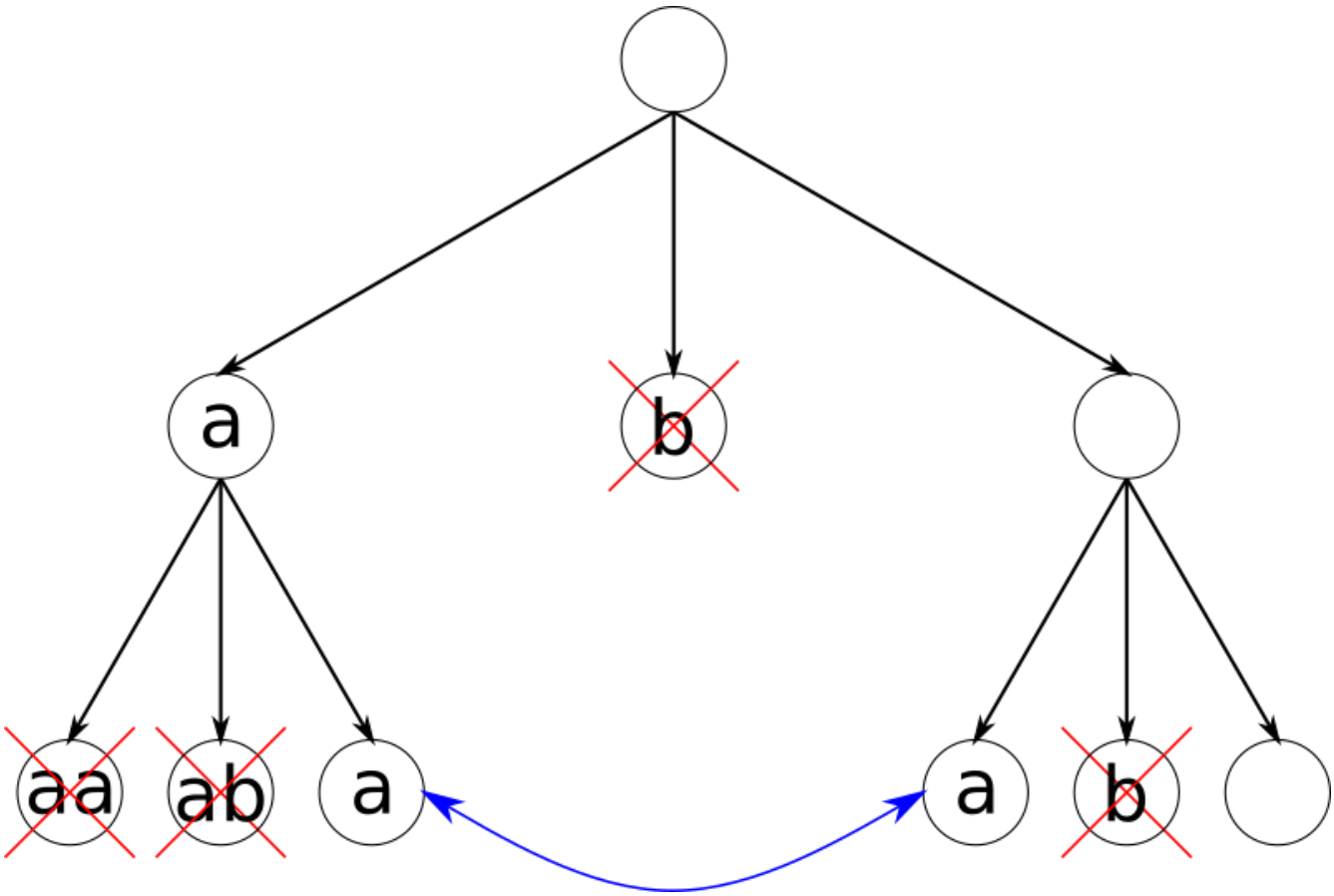


Fig. 2: Beams are iteratively created (from top to bottom), equal beams are merged and only the best 2 beams (beam width=2) per iteration are kept.

Let's look at how we have to adapt vanilla beam search to get the desired behavior. We want the algorithm to behave differently when it recognizes a word and when it recognizes a number or punctuation marks. Therefore we add a state variable to each beam (state diagram see Fig. 3). A beam is either in word-state or in non-word-state. If the beam-text currently is "He**l**", then we are in word-state and only allow adding word-characters until we have a complete word like "Hell" or "Hello". A beam currently in non-word-state could look like this: "She is **3**".

Going from word-state to non-word-state is allowed when a word is completed: then we can add a non-word-character to it, like if we have "Hello" and add " " to it we get "Hello ". The other direction from non-word-state to word-state is always allowed, like in "She is 33 ", where we may add the word-character "y" to get "She is 33 y" (and later, maybe, will get "She is 33 years old.").

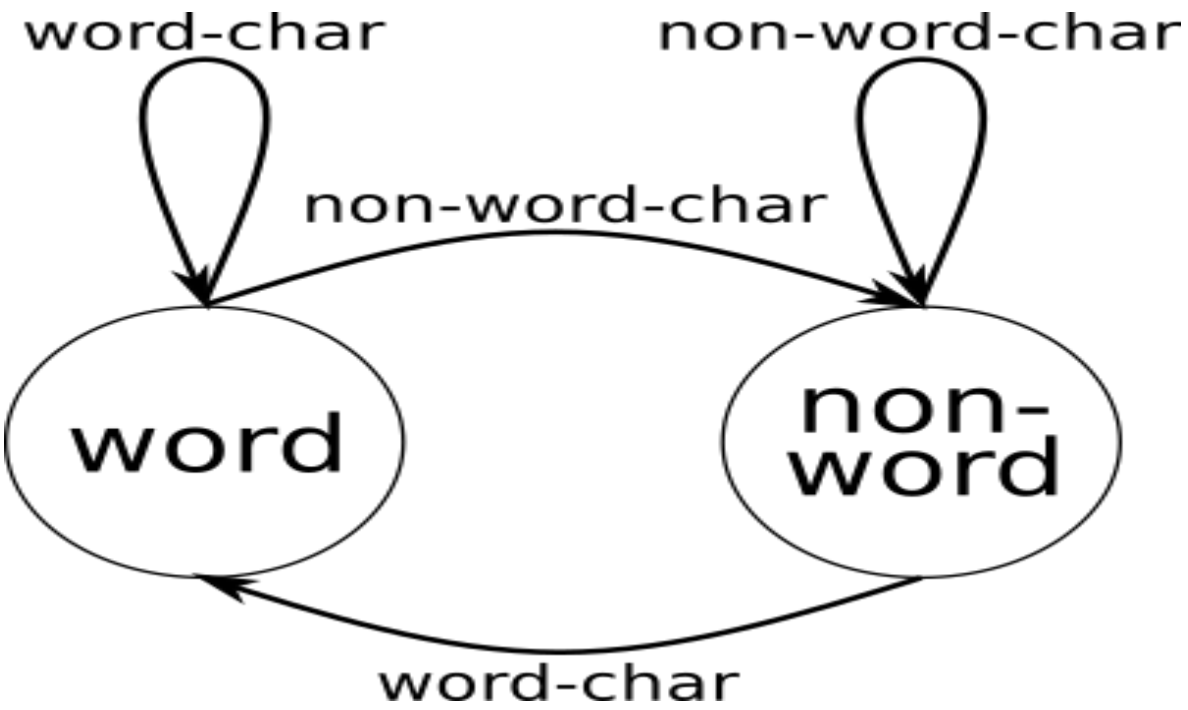


Fig. 3: Each beam can be in one of two states.

When in word-state, we only allow adding characters which eventually will form words. In each iteration, we at most add one character to each beam, so how do we know which characters will, after adding enough characters, form a word? A prefix tree as shown in Fig. 4 can solve this task: all words from the dictionary are added to the tree. When adding a word like "too", we start at the root node, add (if it does not yet exist) an edge labeled with the first character "t" and a node, then add an edge "o" to this new node, and again add an "o". The final node gets marked to indicate the end of a word (double circles). If we repeat this for the words "a", "to", "too", "this" and "that", we get the tree shown in Fig. 4.

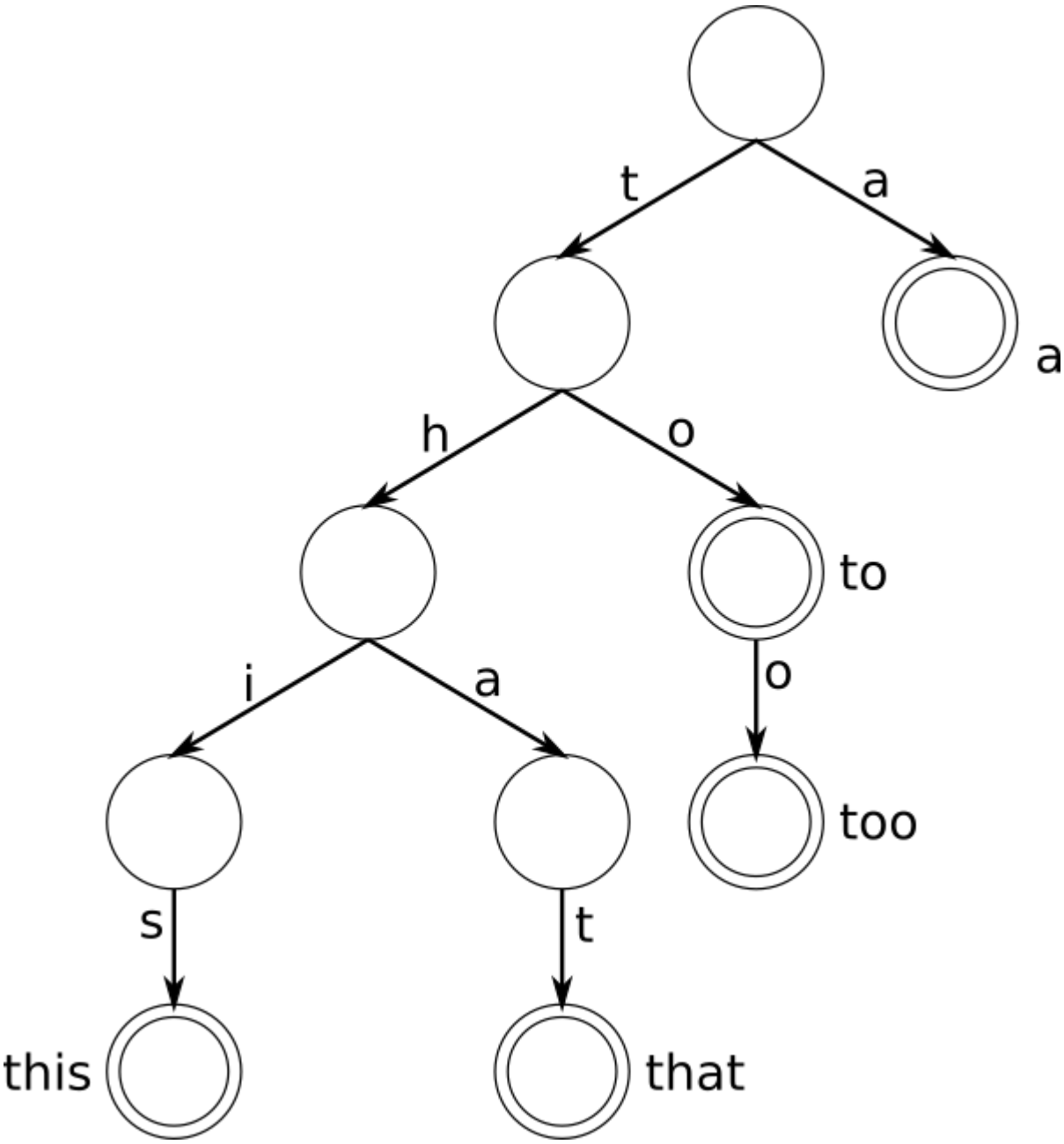


Fig. 4: Prefix tree containing words “a”, “to”, “too”, “this” and “that”.

It is now easy to answer two questions:

- Given a prefix, **which characters can be added** to eventually form a word? Simply go to the prefix’s node in the prefix tree by following the edges labeled with the prefix’s characters, and then collect the labels of the outgoing edges of this node. If we are given the prefix “th”, then the possible next characters are “i” and “a”.

- Given a prefix, **which words can be created** out of it? From the prefix's node, collect all descendant nodes which are marked as a word. If we are given the prefix "th", then the possible words are "this" and "that".

With this information, we can constrain the characters added to beams (which are in word-state): we only add characters which eventually will form a word. See Fig. 5 for an illustration.

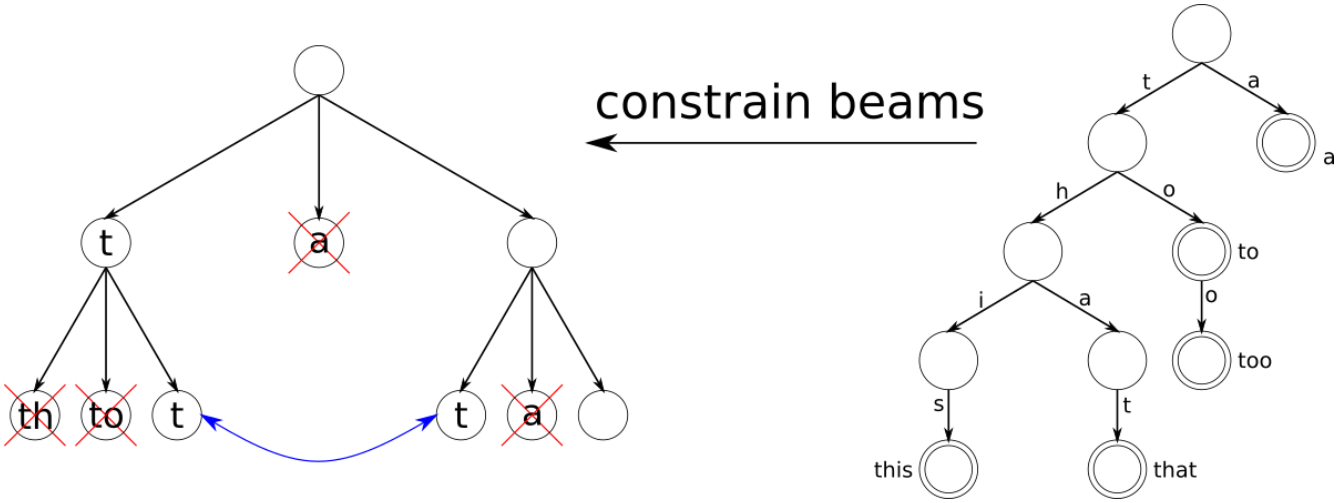


Fig. 5: The beams (left) are constrained by the prefix tree (right).

We only keep the best-scoring beams per iteration. The score depends on the NN output. But we can also incorporate a word-LM with different scoring modes (the provided abbreviations for these modes are used in the results section):

- **Don't use the LM at all (W)**, only constrain the words by a dictionary. This is what we have discussed so far.
- **Score by LM each time a word is fully recognized (N)**. Suppose we have a beam "My nam". As soon as an "e" is added and therefore the last word "name" is finished, the LM scores seeing "My" and "name" next to each other.
- **Look into the future (N+F)**: with the help of the prefix tree, we know which words might occur in later iterations. A beam "I g" might be extended to "I go", "I get", … So instead of waiting until a word is finished to apply the LM-score to a beam, we can also do it in each iteration by summing over the LM-scores of all possible words, this would be "I" and "go", "I" and "get", … in the mentioned example. A small performance improvement can be achieved by only taking a **random subset (N+F+S)** of the words.

We now have an algorithm which is able to recognize the correct text "A random number: 1234." shown in Fig. 1. The words are constrained by a dictionary, but all the other characters are just added in the way the NN sees them.

How well does Word Beam Search perform?

It is nice that the algorithm is able to correctly recognize the sample from Fig. 1. But we are more interested in how well the algorithm performs on a complete dataset. We will use the Bentham HTR dataset. It contains handwritten text from the time around the year 1800, a sample is shown in Fig. 6.

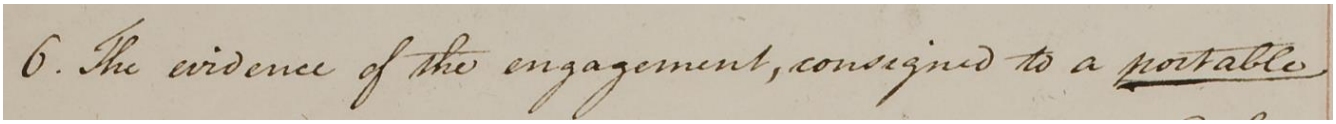


Fig. 6: Sample from Bentham dataset.

We feed the same NN output into all decoding algorithms. This way, a fair comparison is possible. We use two different training texts for the LM: 1.) the text from the test-set which is an ideal training text as it contains all words to be recognized and 2.) a rudimentary training text created from the text of the training-set concatenated with a word list containing 370000 words. The error measures are character error rate (CER) and word error rate (WER). CER is the edit distance between ground truth text and recognized text, normalized by the ground truth length. WER is defined the same way, but on word-level.

Results are shown in Table 1. Word beam search outperforms the other algorithms on this dataset. If we have a good training text for the LM, it makes sense to use it to score the beams (modes N, N+F or N+F+S). Otherwise, it is best to just constrain the words and don't use the LM to score the beams (mode W).

If we don't know which words we have to recognize, or if we don't have to recognize words at all, other algorithms will perform better.

	Perfect LM	Rudimentary LM
Best Path	5.60 / 17.06	5.60 / 17.06
Token Passing	8.16 / 9.24	(not feasible)
VBS	5.35 / 16.02	5.55 / 16.39
WBS W	4.22 / 7.90	**5.47** / 14.09
WBS N	4.07 / **7.08**	6.15 / **13.90**
WBS N+F	**4.05** / 7.36	6.76 / 18.00
WBS N+F+S	4.06 / 7.39	6.75 / 18.06

Table 1: Results given as CER [%] / WER [%] for different algorithms (VBS: vanilla beam search, WBS: word beam search, W: no LM used, N: LM used, N+F: LM with forecasting, N+F+S: LM with forecasting and sampling) and training texts for the LM (perfect and rudimentary LM). A small CER/WER means that we have a well performing algorithm.