### A PROJECT REPORT ON

HANDWRITTEN TO TEXT DOCUMENT CONVERTER

Submitted In Fulfilment of The Requirements for The Award of The Degree of

### BACHELOR OF ENGINEERING IN

**ELECTRONICS AND COMMUNICATION ENGINEERING**

Submitted by

| **KEERTHANA MALLAVARAPU** | **1602-17-735-075** |
| --- | --- |
| **MD SHADAAB FARHAN** | **1602-17-735-083** |
| **PRASHANTH ASIREDDY** | **1602-17-735-316** |

UNDER THE GUIDANCE OF **Mrs. S. ARUNA DEEPTHI** ASSISTANTPROFESSOR

ELECTRONICS AND COMMUNICATION ENGINEERING



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING VASAVI COLLEGE OF ENGINEERING

(AFFILIATED TO OSMANIA UNIVERSITY) IBRAHIMBAGH, HYDERABAD-500031 2021

### DEPARTMENT OF

**ELECTRONICS AND COMMUNICATION ENGINEERING**



**CERTIFICATE**

This is to certify that the Project Report entitled: “HANDWRITTEN TO TEXT DOCUMENT CONVERTER” is a bonafide work done and submitted by

| **KEERTHANA MALLAVARAPU** | **1602-17-735-075** |
| --- | --- |
| **MD SHADDAB FARHAN** | **1602-17-735-083** |
| **PRASHANTH ASIREDDY** | **1602-17-735-316** |

In fulfilment of requirement for the award of Bachelor of Engineering degree in Electronics and Communication Engineering during the year 2020-2021.The result embodied in this project report has not been submitted to any other university or institute for the award of any degree

Head of the Department Internal Guide

### Dr . E.SREENIVASA RAO Mrs. S. ARUNA DEEPTHI

Professor and HOD, ECE Dept. Asst Professor, ECE Dept

External Examiner

### DECLARATION

We hereby declare that the results embodied in this dissertation entitled “HANDWRITTEN TO TEXT DOCUMENT CONVERTER” is carried out by us during the academic year 2020-2021 in fulfilment of the requirement for the award of B.E. (Electronics and Communication Engineering) from “**Vasavi College of Engineering**”.

We have not submitted the same to any other university or organization for the award of any other degree.

| **KEERTHANA MALLAVARAPU** | **1602-17-735-075** |
| --- | --- |
| **MD SHADDAB FARHAN** | **1602-17-735-083** |
| **PRASHANTH ASIREDDY** | **1602-17-735-316** |

### ACKNOWLEDGEMENT

This satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mentioning of the people whose constant guidance and encouragement made it possible. We take pleasure in presenting before you, our project, which is result of studied blend of both research and knowledge.

It is our privilege to express our earnest gratitude and venerable regards to our internal guide **Mrs. S. Aruna Deepthi**, M.E, ASST PROFESSOR, E.C.E. Department, Vasavi College of Engineering, Ibrahimbagh, for abounding and able guidance during the preparation and execution of the project work. We are grateful for her cooperation and her valuable suggestions.

We record with pleasure our deep sense of gratitude to **Dr. E. SREENIVASA RAO**, M.Tech(ECE), M.Tech(CS), Ph.D., Head of the Department, E.C.E. for his simulating guidance and profuse assistance we have received, which helped throughout the project.

We also thank our Project coordinator, Professor **Dr. N. SIVA SANKAR REDDY**, M.Tech, Ph.D., Department of ECE and Professor **Dr. K. VEERA SWAMY**, M.Tech, Ph.D., Department of ECE for their continuous guidance and valuable suggestions throughout our project.

We also acknowledge and thank one and all for the technical help extended in shaping up our project.

### With Regards and Gratitude

**KEERTHANA MALLAVARAPU**

**MD SHADAAB FARHAN**

**PRASHANTH ASIREDDY**

**ABSTRACT**

Handwriting recognition, also recognized as handwriting OCR (Optical Character Recognition) is an OCR applied science that translates the handwritten letters to analogs digital text. In this paper, we are providing a software solution to automatically convert the handwritten image into text. Almost every sector and all the organizations require information. Organizations like educational institutions, medical and healthcare sectors, IT organizations, banks, small scale and large-scale industries and businesses require the customer or the client to fill out a handwritten form. Later this information is uploaded into the database manually which is tedious and time consuming. In this project, we implemented CNN and Bidirectional RNN with the help of open-source libraries like OpenCV, TensorFlow and Matlibplot in Python language to perform segmentation, normalization and feature extraction.

Many handwritings recognition tools have been in the market since 70’s, but still, there are not many tools that give maximum accuracy as each person owns a unique style of writing, pressure and tilt. There are two ways to recognize characters: Markov model and ANN (Artificial Neural Networks). Deep learning enables us to train and test the model with huge datasets, thus leading to maximum accuracy. In this model, we have used neural network consisting of 5 CNN layers and 2 RNN layers and CTC model to decode

**List of contents:**

| **Chapter no.** | **Title of chapter/section** | **Page no.** |
| --- | --- | --- |
|  | **Chapter-1: Introduction** | **1-9** |
| 1.1 | Introduction to the topic | 1 |
| 1.2 | Software used | 3 |
| 1.3 | Flowchart | 6 |
| 1.4 | Objective | 8 |
| 1.5 | Thesis Structure | 8 |
|  | **Chapter-2: Literature Survey** | **10-13** |
|  | **Chapter-3: Methodology and Model Description** | **14-22** |
| 3.1 | Convolutional Neural Networks | 14 |
| 3.2 | Recurrent Neural Network | 17 |
| 3.3 | Long short-term memory | 18 |
| 3.4 | Connectionist Temporal Classification | 19 |
|  | **Chapter-4: Experiment Results** | **23-31** |
| 4.1 | Dataset | 23 |
| 4.2 | CNN | 23 |
| 4.3 | RNN | 23 |
| 4.4 | CTC | 25 |
| 4.5 | Training | 25 |
| 4.6 | Model Evaluation | 26 |
| 4.7 | Output images obtained on testing | 28 |
| 4.8 | Deployment | 31 |
|  | **Chapter-5: Conclusion & Future scope** | **32-33** |
| 5.1 | Conclusion | 32 |
| 5.2 | Future scope | 32 |

**REFERENCES 34-35**

**ANNEXURE 36-52**

**List of figures:**

| **Figure no** | **Caption of figure** | **Page no** |
| --- | --- | --- |
| Figure 1.1 | Instance of HTR: a digital text is resulted by the HTR system for an  image carrying text. | 2 |
| Figure 1.2 | Flow chart of handwritten to text convertor using deep learning | 6 |
| Figure 3.1 | Typical processing of image using CNN | 14 |
| Figure 3.2 | Example of Pooling | 15 |
| Figure 3.3 | Representation of typical Activation Function | 15 |
| Figure 3.4 | Difference between unidirectional and bidirectional | 16 |
| Figure 3.5 | Block diagram of LSTM | 18 |
| Figure 3.6 | Problem that can be avoided using CTC | 19 |
| Figure 3.7 | The Neural Network's output matrix. At each time step, it displays the character probability. | 20 |
| Figure 4.1 | Code snippet of CNN | 23 |
| Figure 4.2 | Code snippet of RNN | 24 |
| Figure 4.3 | Code snippet of CTC | 25 |
| Figure 4.4 | Code snippet of Training | 25 |
| Figure 4.5 | Plot of loss function | 26 |
| Figure 4.6 | Loss vs Epoch Graph | 27 |

**List of Tables**

| **Table no.** | **Caption of table** | **Page no** |
| --- | --- | --- |
| Table 5.1 | Dataset details | 22 |
| Table 5.2 | The test images along with output digitalized text | 27 |

**CHAPTER 1**

**INTRODUCTION**

**1.1 Introduction to the topic:**

**DEEP LEARNING:**

Deep Learning is a man-made function that mimics the performance of a human brain to manage information and create designs and utilize it to make decisions. The practical definition of Deep Learning is ‘It’s a sub-domain of ML (machine learning) algorithms in the type of a neural network that applies a cascade of layers of working units to derive features and make perceptive approximations about new data. Deep learning is also known as deep neural network.

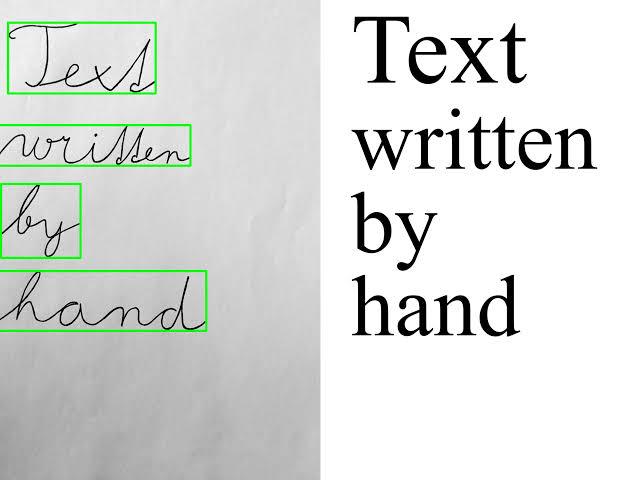
**CONVOLUTIONAL NEURAL NETWORK:**

A Convolutional Neural Network is a Deep Learning algorithm which is capable of taking in an input image, assign significance to different aspects/objects within the image and be capable of differentiating from one to other. Filters are hand-engineered in basic approaches, with ample training, ConvNets must be able to pick up on these filters/characteristics. The design of a ConvNet is sensibly comparable to that of the connectivity rules of Neurons inside of the Human Brain, and was influenced by organization of the Visual Cortex. Individual neurons react in response to inputs only in a restricted region of the visual field called as the Receptive Field. A cluster of such patches encircles the whole visual field.

An image is nothing but a cast of pixel values. A ConvNet is capable to effectively obtaining the Spatial and Temporal colonies with the use of appropriate filters in a picture. The design performs a more accurate match to the picture dataset due to the decrease the number of variables included and re-usability of weights. In alternative terms, the network can be prepared to better comprehend how the image is progressing. The portion of ConvNet is to decrease the images into a format that is easier to work with, without sacrificing crucial aspects for obtaining a great prediction.

Handwritten Text Recognition (HTR) is nothing but translating handwritten content into digital text. There are 2 types of handwritten text recognition, i.e., online recognition and offline recognition. Online recognition is carried out as the text that has to be identified is being written (e.g., by a pressure detection), as a result geometric and temporal data are on-hand. Offline recognition, on the contrary, is done after the material has been written. The content is apprehended and the resulting pictures are taken care of (e.g., by a scanner). Online recognition is said to be the a less challenging problem. Challenges with respect to HTR incorporate the running-hand nature of penmanship, the assortment of each alphabet in estimate and shape and huge terminologies. Presently, enormous sums of manually written reports are scattered in libraries accessible via the internet. Interpreting and tabulating make these records easily available to all.

A case of HTR within the area of records is given in Figure 1.1 which shows one text-line translation.



**Figure 1.1: Instance of HTR: a digital text is resulted by the HTR system for an**

**image carrying text.**

This study revolves around the classifier, its parameters, and the options for pre-processing of the data image as input and a textual output post-processing strategy. Offline HTR is examined since data images of handwritten text carry out as input. The examined classifier is based on Artificial Neural Networks. HTR tasks are connected to Hidden Markov Models. Nevertheless, the proclaimed results of HTR contends to appear that Artificial Neural Networks outperform Hidden Markov Models, consequently Artificial Neural Networks have been chosen to participate in this research. Paper review strategies distinguishing the text on a page or dividing a page into lines are not acceptable. The only exemption is word-segmentation, which is since the datasets are annotated on a line-by-line basis. Hence, one plausibility is to divide text-lines into words and segregate each word on its own, while another one is to refrain from segmentation and bolster overall number of lines in the classifier.

The proposed model makes use of ANNs. Multiple Convolutional Neural Network layers are prepared to extricate important features from the input data image. These layers yield feature matrix which is entrusted to the Recurrent Neural Network layers. The Recurrent Neural Network proliferates data through the categorization. Later, the result of the Recurrent Neural Network is positioned on a matrix it provides a score for each character within each sequence element. As the Artificial Neural Network is trained employing a particular coding plot, a decoding algorithm must be used on the Recurrent Neural Network to obtain the final text. Connectionist Temporal Classification operation is done to do training and decoding. Translating will be able to take advantage of a Language Model.

There are couple of reasons for pre-processing. Firstly, putting the issue simpler for the classifier and data augmentation. Secondly, dispatching the tilt from the text and word division are talked about as well as solutions that make things easier. Data augmentation is implemented by irregular shifts to the dataset images. The derived text can consist of spelling mistakes, hence a text-postprocessing routine is used in order to account for them.

**1.2 SOFTWARES USED:**

**Python:**

Python is a high-level programming language. It is interpreted common-purpose programming language. Python was developed by Guido van Rossum and first made public in 1991, Python's architecture logic focuses on code readability and its outstanding utilization of noteworthy indentation. It is a language that develops object-oriented approach point of view to help software engineers write clear, coherent code for every project. Python is interactively written and supports garbage-collection. It underpins numerous programming ideal models, counting organized, object-oriented and useful programming. It is regularly portrayed as a "batteries included" language due to its overall standard library. A worldwide social-unit of software engineers create and maintain CPython, an open-source reference usage. A non-profit organization, manages and coordinates assets for Python and CPython development. Python interpreters are accessible for numerous working frameworks

It is a common-purpose high level programming language that is mostly used in data science and for writing deep learning algorithms and neural network models. Python supports many libraries.

**Libraries/Modules Used:**

**OpenCV:**

This is a cross-platform library using this we will be able to build real-time computer vision projects. Its primary objective is on image processing, video capture and drawing out conclusions including features like object and face detection. By using this, one can analyse images and videos to spot faces, features, objects and even handwriting. When it combined with various libraries, like NumPy, python is able to process the OpenCV array structure for analysis.

Using this library, one can −

• Analyze the video, i.e., estimate the motion in it, subtract the background and track objects

• Perform feature detection

• Capture and save video Process images

• Detect specific objects such as faces, iris, eyes, cars, human activities in the videos or images.

**TensorFlow:**

This is an end-to-end open-source platform for machine learning. It features a absolute, adaptable environment of tools, community and libraries resources that lets analysts thrust the cutting-edge in ML and designers effectively construct and convey Machine Learning fuelled applications. TensorFlow offers multiple levels of deliberation so you'll select the proper one for your needs. Construct and train models by utilizing the high-level Keras API, which makes getting begun with TensorFlow and machine learning simple. In the event that you wish more adaptability, enthusiastic execution permits for quick emphasis and instinctive investigating. For expansive ML preparing assignments, utilize the Dispersion Technique API for taken preparing on distinctive equipment arrangements without changing the show definition

Features of TensorFlow:

1. **Models can be created effortlessly:** TensorFlow bolsters high-level APIs, through

which Machine Learning models can be built effortlessly utilizing Neural Networks.

1. **Complex Numeric Computations can be done:** As the input dataset is tremendous,

the numerical computations/calculations can be done easily.

1. **Consists of Machine Learning APIs:** TensorFlow is wealthy in Machine Learning

APIs that are of both low-level and high-level. Steady APIs are accessible in Python and C. Directly, working on APIs for Java, JavaScript, Julia, MATLAB, R, etc.

1. **Easy sending and computation utilizing CPU, GPU:** TensorFlow bolster preparing

and building models on CPU and GPU. Computations can be done on both CPU and GPU and can be compared too.

1. **Contains pre-trained models and datasets:** Google has included numerous datasets

and pre-trained models in TensorFlow. Datasets incorporate mnist, vggface2, ImageNet, coco etc.

1. **Pretrained models for mobiles, inserted gadgets, and generation:** The Machine

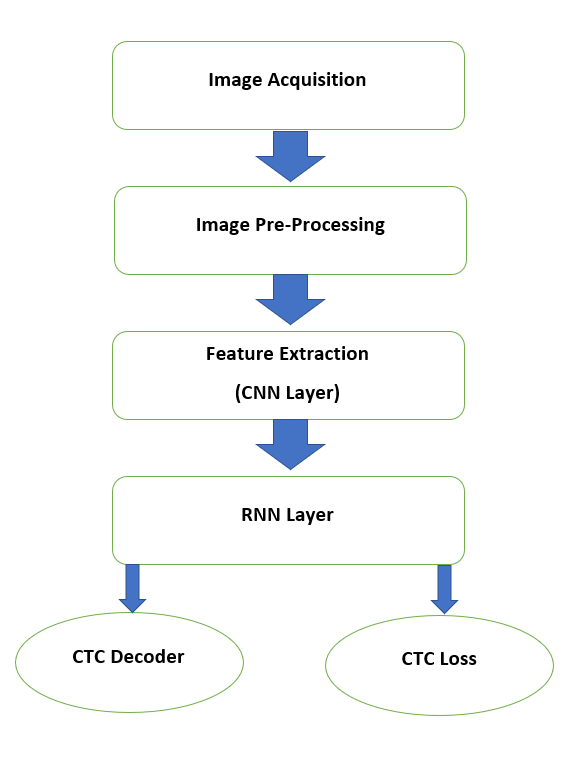
models can be deployed on portable and embedded gadgets utilizing TensorFlow. Pre-trained models can be specifically utilized for production.

**NumPy:**

This is a library of the Python programming language, including aid for large, multidimensional arrays together with a huge accumulation of high level of numerical functions to work on these matrices.

**Matlibplot:**

This is a multi-platform data visualization library built on cluster of NumPy. It was made public by John Seeker within the year 2002. Most prominent benefits of visualization are that it permits us visual access to tremendous sums of information in effortlessly understandable visuals. Matplotlib comprises of several plots like line, bar, diffuse, histogram etc.

**Flowchart:** 

**Figure 1.2: Flow chart of handwritten to text convertor using deep learning**

**Image acquisition:**

The first step of HCR system is the image acquisition. In image acquisition, the scanned picture or image of the handwritten content is determined. This can be done using offline HTR system by using a scanner, photocopies or by directly writing on computer using stylus.

**Image pre-processing:**

The next step of handwritten character recognition is image pre-processing. It is crucial for better recognition rate. Its main objective is to normalise strokes and remove any variations in the handwritten text which would otherwise reduce recognition rate and complicate recognition. These variations or distortions are of many types such as tilt in the handwriting, jitters in the text, irregular size of the text, distance between word to word or letter to letter, missing points during pen movement etc. Image pre-processing consists of 5 steps:

1. Size normalization and focusing
2. Missing points interpolation
3. Smoothing
4. Slant correction
5. Resampling of points.

**Feature Extraction:**

Features are unique attributes of a word or a character that helps us to recognise a character/word and distinguish it. Features are classified into 2 main types: statistical features and structural features. Structural features represent the structure or shape whereas the numerical estimates of pixel intensities generated across pictures are known as statistical features. They include Fourier descriptors, pixel densities, moments, directions, etc. The main purpose of feature extraction step is to derive the most prevalent pattern suitable for classification. To extract the characteristics of individual characters, feature extraction techniques such as zoning, Chain Code (CC), histogram and Gradient based features can be used. The system can be trained from these features. Or model. In our project, feature extraction is done using 5 Convolutional Neural Networks (CNN) which will be discussed in detail in the coming chapters.

**RNN layer:**

RNN (Recurrent Neural Network)/LSTM (Long Short-term Memory) deals with chronological data to detect temporal patterns and produce results. Bi-RNN connect the same output to the two hidden layers in opposite directions. The output layer of this type of productive deep learning can acquire knowledge from the both future and past states at the same time.

**CTC:**

The previous layers i.e., CNN layers are used to derive a chronological succession of features and the RNN layers are used to travel data through this sequence. Now, the output of the RNN layer is the score of each character for each element in the sequence which is illustrated in the form of a matrix. Now, we can do 2 things using this matrix

1. train: To train the NN, determine the loss value.
2. infer: decode the matrix in order to extract the text from the input image

**1.3 Objectives:**

The main objective of this paper is to identify or recognise the scanned handwritten images and convert them into digitalized text using deep learning, i.e., using Convolutional Neural Networks, Recurrent Neural Networks and CTC to get maximum accuracy. These models/algorithms are created in Python language using different libraries like TensorFlow, Numpy, OpenCV and Matlibplot.

Thus, the main objectives are:

1. To minimize the burden of updating and uploading the information into the system manually which is tedious and time consuming and computerising it to make it much faster.
2. Identifying the text from ancient scripts and digitalizing it for reserving.

**1.4 Thesis Structure:**

This thesis is mainly divided into 5 chapters. Chapter 1 introduces the project along with brief explanation of flowchart. In chapter 2, various studies have been analysed which are presented in the form of Literature survey. Chapter 3 describes the proposed model which converts any image containing content into digitalised text. The implementation is explained in this chapter. Chapter 4 talks about the experimental results. Conclusion and future scope of the project are written in Chapter 5 followed by References and Annexure.

**CHAPTER 2**

**LITERATURE SURVEY**

In 1959, Grimsdale made a major endeavour in the field of character recognition research. In early 1960s, a method known as the analysis-by-synthesis method, proposed by Eden in 1968, was at the heart of a lot of research. Eden's work was significant because he formally established that all handwritten characters are generated a point that is defined by a finite number of schematic characteristics had previously been implied. This principle was then enforced to all syntactic (structural) character recognition algorithms.

Crane's early work resulted in the first patent for an unique pen device in 1964. These early efforts improved in the early 1970s with a device that employed what was dubbed the "SRI pen" to input characters into a computer. Later that year, a rudimentary method for inputting handwritten Chinese characters was published.

In 1974, the first patent in the field of signature verification was granted. SRI was granted a patent in 1991 for a version of the pen that can capture dynamic data in 5-dimensions.

K. Gaurav, Bhatia P. K. Et al [1], this paper discusses the early pre-processing approaches used in character detection with various types of pictures, ranging from mere handwritten forms to documents with coloured and complicated backgrounds and varying intensities. Various pre-processing approaches are covered, including correction and skew detection and correction, image enhancing methods such as binarization, contrast stretching, noise removal techniques, segmentation and normalisation, and morphological processing approaches. It was determined that we couldn't fully process the image using a single pre-processing technique. However, even with all of the aforementioned strategies in place, it may be impossible to attain 100% accuracy in a pre-processing system.

Salvador España-Boquera et al [2], in this paper, for identifying unconstrained offline handwritten texts, a hybrid Hidden Markov Model (HMM) model is suggested in this paper. Structural element of the visual model was represented with Markov chains, and emission probabilities were estimated with the help of Multilayer Perceptron. With supervised learning methods, multiple strategies are used to eliminate slant and slope from handwritten writing, as well as to normalise the size of text images. The development of a system with high accuracy in pre-processing and recognition, both based on ANNs, was one of the important features of this recognition system.

U. Pal, T. Wakabayashi and F. Kimura et al [3], It is proposed to recognise offline handwritten digits in six prominent Indian scripts using a modified quadratic classifier-based method.

J. Pradeep, E. Srinivasan and S. Himavathi et al [4], For diagonal feature extraction, offline character recognition has been proposed. It is based on the ANN paradigm. This Neural recognition system is built utilising two approaches, one with 54 features and the other with 69 features. To compare the recognition effectiveness of the recommended diagonal strategy of feature extraction, the neural network recognition system is trained using horizontal and vertical feature extraction approaches. The diagonal approach of feature extraction offers a recognition accuracy of 97.8% for 54 features and 98.5 percent for 69 features, according to the findings.

A. Brakensiek, J. Rottland, A. Kosmala, J. Rigoll et al [5], this work describes an off-line cursive handwriting detection system based on HMM that employs discrete and hybrid modelling techniques. Handwriting recognition studies are contrasted using a discrete technique and two alternative hybrid approaches that combine discrete and semi-continuous structures. To develop the system, a segmentation-free technique is being examined. It was discovered that a hybrid modelling methodology for Hidden Markov Models that relies on a neural vector quantizer (hybrid MMI) can increase recognition rate performance when this might be attributed to a tiny data collection as compared to discrete and hybrid Hidden Markov Models based on fatigued mixture structure (hybrid - TP).

Sandhya Arora et al [6], Intersection, chain code histogram, shadow features, and straight line fitting features were employed as feature extraction approaches. Chain code histogram features, Intersection features, and line fitting features are calculated by segregating the character image into distinct segments, while shadow features are calculated for the character image at a global level. The total recognition rate for Devanagari characters was 92.80 percent when tested with a dataset of 4900 samples.

Nafiz Arica et al [7], developed an approach that avoids the majority of pre-processing processes, which results in the loss of critical data. One of the method's significant accomplishments is the creation of a powerful segmentation algorithm. Local maxima and minima, character boundaries, upper and lower baselines, slant angle, stroke breadth and height, and descenders and ascenders are all used to optimise the search method for the best segmentation route on a grayscale image. This strategy reduces over-segmentation. The use of HMM training, not only for model parameter \*estimation, but also for parameter estimation in the global and feature space, is another contribution. Hidden Markov Model probabilities are also utilised to rank the candidate character and quantify the shape information. The HMM shape recognizer's power is increased by portraying a two-dimensional character image in one dimension.

M. Hanmandlu, O.V. Ramana Murthy et al [8], have published a work that shows how to recognise handwritten Hindi and English digits by modelling them as exponential membership functions that act as a fuzzy model. Altering the exponential membership functions outfitted to the fuzzy sets allows for detection. These fuzzy sets are created using the Box technique and features consisting of normalised distances. The membership function is influenced by two structural factors that are calculated by optimising the entropy with the goal of achieving one in the membership function. Primarily, 95 percent of Hindi numbers are recognised, while 98.4% of English numerals are recognised.

T.Som, Sumit Saha et al [9], for HCR, they've discussed a fuzzy membership function-based technique. Normalized character pictures are 20 X 10 pixels. Each character's average picture (fused picture) is made up of ten images. The horizontal and veritcal projections of characters are used to establish the bonding box surrounding the character. The image is downsized to 10 X 10 pixels after being cropped to the bounding box. After that, the procedure is followed, and the thinned image is placed one after another in the row of a 100 X 100 canvas. The test image's similarity score is compared to the fusion image's similarity score, and the characters are categorised.

R. Bajaj, L. Dey, S. Chaudhari et al [10], For the classification of Devanagari numerals, researchers used three different types of characteristics: moment features, density features, and component descriptive features. For handwritten Devanagari numerals, they presented a multi classifier connectionist design to improve recognition dependability, and they achieved 89.6% accuracy.

Mohammed Z. Khedher, Gheith A. Abandah, and Ahmed M. Al Khawaldeh [11] et al, this study explains how the features utilised in character recognition play a big role. A number of characteristics of handwritten Arabic characters are highlighted and examined. On the basis of the selected features, an off-line recognition system was created. With authentic samples of handwritten Arabic characters, the system was taught and evaluated. The significance and precision of the selected features are assessed. The detection accuracy for numerals and letters is 88 percent and 70 percent, respectively, based on the selected attributes. Using weights based on insights derived from the precisions of individual features, further improvements can be made.

Sushree Sangita Patnaik and Anup Kumar Panda May 2011 [12] et al, research suggests the use of bacterial foraging optimization and particle swarm optimization methods to achieve effective harmonic compensation by reducing the undesired losings that occur within the APF. For two different supply conditions, the efficiency and effectiveness of two different techniques are evaluated. By using BFO, the total harmonic distortion in the source current, is a measure of APF performance, is decreased to less than 1%. The derivations show that bacterial foraging optimization outperforms traditional and particle swarm optimization -based techniques by ensuring outstanding APF functionality and early dominance harmonics in the source current, even when the supply is imbalanced.

In [13], A method is presented for creating a handwritten Tamil character by executing a series of strokes. The stroke was displayed as a string of shape features using a structural or shape-based representation. An unknown stroke was detected using this string representation by equating it to a database of strokes with the use of a flexible string mapping process. All of the component strokes were identified to form a complete character.

G. Pirlo and D. Impedovo et al [14], proposed a new class of membership functions for zoning-based classification called Fuzzy membership functions. To maximise classification performance, these FMFs can be simply changed to the specific parameters of a categorization task. In this paper, a real coded genetic algorithm is proposed for finding the optimal FMF and the ideal zoning represented by Voronoi tessellation in a single optimization step. The findings of the experiments, which were conducted in the field of handwritten number and character recognition, show that the optimal Fuzzy membership function outperforms existing membership functions on the basis of an ranked-level, abstract level, and measurement-level weighting models known in the literature.

Yoshimasa Kimura et al [15], presented a paper on how to use a genetic algorithm to choose features for character recognition. With the help of genetic algorithms, the author offers a unique feature selection technique for character recognition. The suggested method picks as candidates for the parent gene only those genes for which the detection rate of training samples crosses over a predefined threshold, and uses a reduction ratio in the number of features used for recognition as the fitness value.

**CHAPTER 3**

**METHODOLOGY AND MODEL DESCRIPTION**

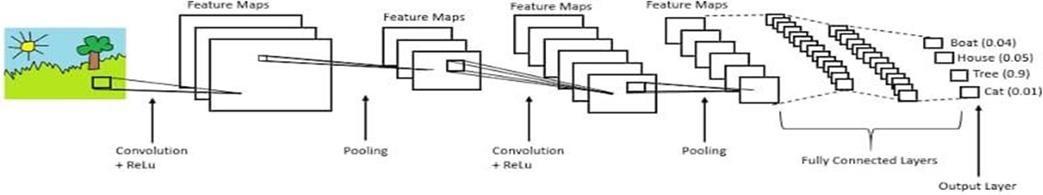
**3.1 CONVOLUTIONAL NEURAL NETWORK (CNN):**

A Convolutional Neural Network is a Deep Learning system that can accept an input image and assign relevance to multiple objects or aspects in the image, as well as differentiate between them. The quantity of pre-processing required by a Convolutional Neural Network is much less than that required by other classification methods. Convolutional Neural Network are capable to pick up these characteristics with adequate training, whereas simple techniques require hand-engineering of filters.

A ConvNet's architecture is inspired by the Visual Cortex's organisation and is related to the connectivity pattern of Neurons in the Human Brain. Individual neurons are only capable to react to stimuli in the Receptive Field, a tiny portion of the optical field. To span the entire visual field, a number of comparable fields can be piled on top of one another.

A ConvNet may successfully capture the Spatial and Temporal relationships in a picture by using appropriate filters. The design provides more desirable fitting to the picture dataset because to the reduced number of attributes involved and the re-usability of weights. Looking at a different approach, the network may be trained to recognise the image's sophistication.

An input and output layer, as well as numerous hidden layers, make up a convolutional neural network. A CNN's hidden layers are usually made up of convolutional layers that convolute with a dot product or multiplication. The activation function is generally preferred as a **RELU layer**, and is consequently followed up by additional convolutions as an example **normalization layers,** **fully connected layers** and **pooling layers**, concerned because the final convolution and activation function masks their inputs and outputs, they are referred to as hidden layers. Though the layers are described as convolutions informally, this is merely a convention. In terms of mathematics, it's a sliding cross-correlation or dot product. It has implications for the appendices in the matrix since it influences how the weighs are calculated at a given index point. The Convolution layer is the central component of a Convolutional Network, and it is responsible for the majority of the computational effort. Pictorial representation of CNN is shown in Figure 3.1 along with all the hidden layers.



### Figure 3.1: Typical processing of image using CNN

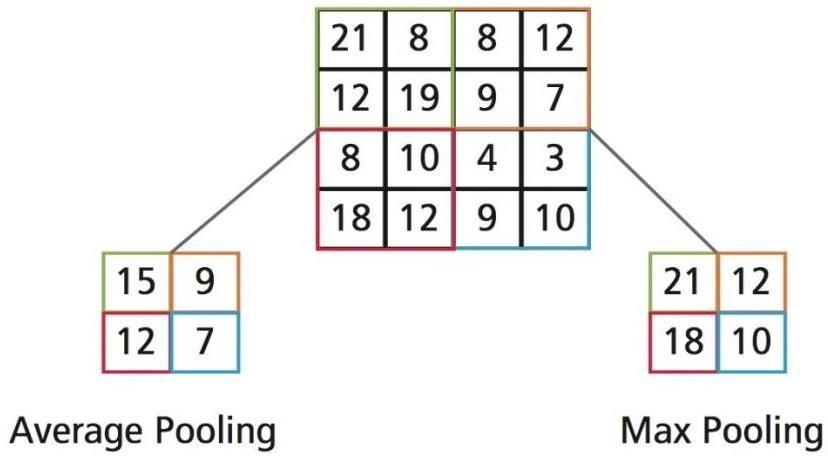
1. **RELU layer:**

The rectified linear activation function, that is basically a piecewise linear function that acts as, if the given input is positive, output is the input directly; else, the output becomes zero. The RELU layer employs the function f(y) = max(y, 0) to all of the input values. In simple terms, the function of the layer is just to change all the negative activations to 0. RELU layer attempts to increase the nonlinear properties of any model and the all-round network without swaying the receptive fields of the conv layer.

1. **Pooling layer:**

Conv networks may comprise global or local pooling layers, which aggregates the findings of one layer's neuron clusters into a single neuron in the next layer. For instance, **max pooling** utilizes the peak value from each of the cluster of neurons at the prior layer. Another example is **average pooling**, which utilizes the average value from each of the cluster of neurons at the prior layer.

An example of how average pooling and max pooling is done is shown in Figure 3.2.



### Figure 3.2: Example of Pooling

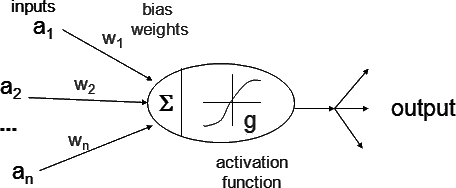
### Max pooling: The maximum value from the portion of the image covered by the Kernel is returned by Max Pooling. Max Pooling works as a Noise Suppressant as well. It removes all noisy activations and conducts de-noising and dimensionality reduction at the same time.

### Average pooling: The average of all the values from the portion of the image covered by the Kernel is returned by Average Pooling. Average Pooling performs dimensionality reduction as a noise suppressing mechanism.

1. **Activation layer:**

Activation functions are important for an ANN to adapt and understand the complicated patterns. The primary function of this layer is to introduce non-linear properties into the network. This layer basically determines the ‘weighted sum’ and adds direction and determines whether to ‘fire’ a particular neuron or not. There are many kinds of non-linear activation functions, few examples are Tanh, Sigmoid, leaky ReLU and ReLU. The non-linear activation function will assist the model to comprehend the complexity and give precise results.

Figure 3.3 represents the typical representation of activation function where inputs are given with weighted biases.



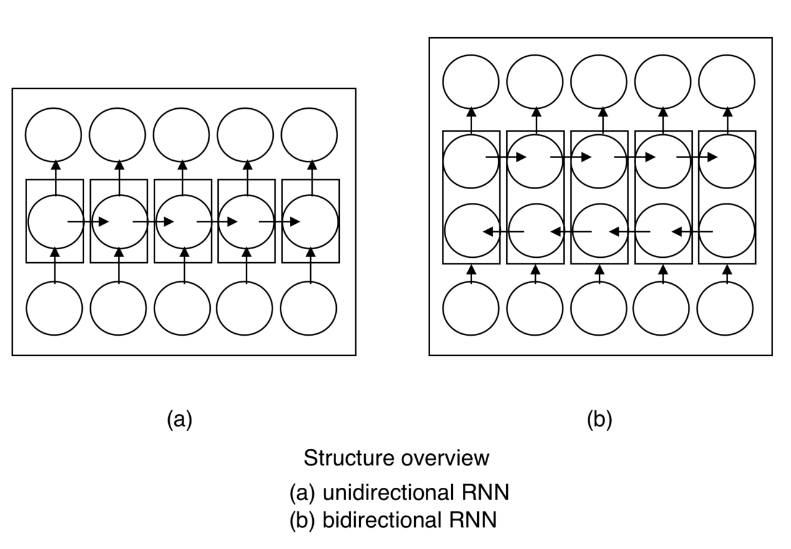
### Figure 3.3: Representation of typical Activation Function

**3.2 Recurrent Neural Network (RNN) & Bidirectional RNN:**

Bidirectional recurrent neural networks (BRNN) are NN that connects same output to two different hidden layer that are facing in opposite directions. The output layer of this type of productive deep learning can collect knowledge from both future and past states at the same time.

The basic idea behind Bidirectional recurrent neural networks is to partition the neurons of a standard Recurrent Neural Network into two directions, one for negative time and the other for positive time. The outputs of those couple of states are not linked to the inputs of the states in the opposite direction. The following figure depicts the general structure of Recurrent Neural Network and BRNN. Unlike normal Recurrent Neural Network, which expects delays for incorporating future information, the input information from the future and past of the recent time frame can be included by using two-time directions.

Because the two directional neurons do not interact, BRNNs can be trained using identical algorithms as RNNs. Back-propagation across time, on the other hand, necessitates the usage of extra processes because updating input and output layers cannot be done simultaneously. The following are the general training procedures: Forward states and backward states are passed first in forward pass, followed by output neurons. Output neurons are passed first in the reverse pass, followed by forward and backward states. The weights are updated after the forward and backward passes are completed. Figure 3.4 shows the pictorial difference between unidirectional RNN and bidirectional RNN.



**Figure 3.4: Difference between unidirectional and bidirectional**

**3.3 Long Short-Term Memory (LSTM):**

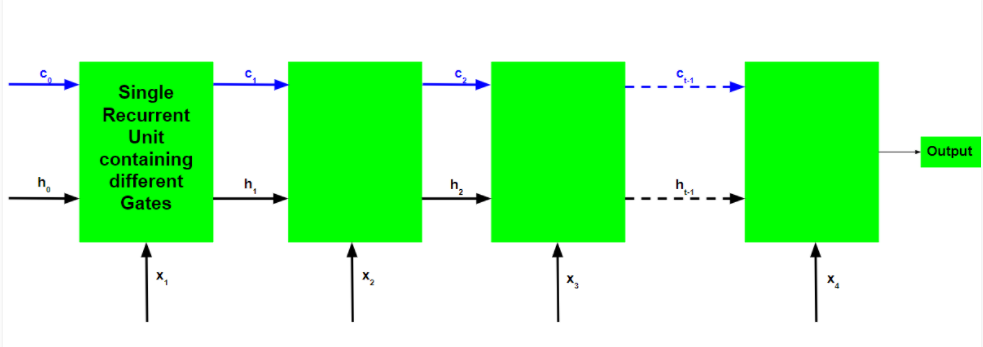
Many variations have been devised to handle the issue of Vanishing and Exploding Gradients in a deep Recurrent Neural Network. The LSTM Network is unique and the most well-known (LSTM). In theory, an LSTM recurrent unit seeks to "remember" all of the network's previous information while "forgetting" unnecessary inputs. This is accomplished by introducing various activation function levels known as "gates" for various aims. For each of the Long Short Term Memory recurrent unit additionally keeps an Internal Cell State vector, which essentially describes the information that the preceding LSTM recurrent unit chose to keep.

A Long Short Term Memory Network is made up of four separate gates, each of which serves a particular purpose, as shown below

1. **The Forget Gate(f)** controls how far the prior data should be forgotten.
2. **Input Gate(i):** The diversity of information to be written onto the Internal Cell State is determined by the Input Gate(i).
3. **Input Modulation Gate(g):** It is frequently regarded as a sub-component of the input gate, and multiple LSTM literatures do not even discuss it, assuming that it is included inside the input gate. It's utilised to add nonlinearity to the data that the Input Gate will store in the Internal State Cell, making it Zero-mean. It is done to shorten the training duration because zero-mean input converges faster. Despite the fact that the actions of this gate are less important compared to those of the others, and it is frequently considered as a finesse-providing notion, it is good practise to include it in the Long Short Term Memory unit's construction.
4. **Output Gate(o):** It controls what is the output the current Internal Cell State should produce.

A LSTM Network's basic work-flow is indistinguishable to that of a Recurrent Neural Network, with the making the exception of that the Internal Cell State is also transmitted along with the Hidden State.

A simple block diagram of LSTM is depicted in Figure 3.5



**Figure 3.5: Block diagram of LSTM**

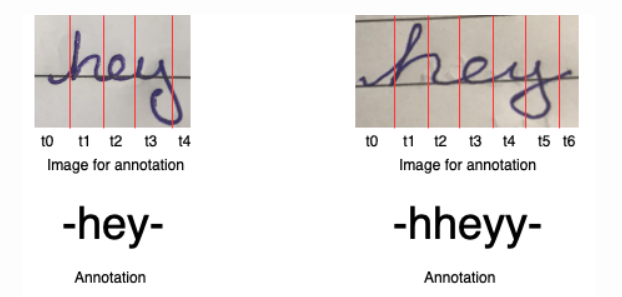
**3.4 Connectionist Temporal Classification (CTC):**

Connectionist Temporal Classification is a Neural Network output that may be used to solve sequence problems such as handwriting and speech recognition where the time fluctuates. Using CTC eliminates the need for a pre-aligned dataset, making the training procedure much simpler. CRNN (Convolutional Recurrent Neural Networks) are the recommended method for building an OCR (Optical Character Reader). For each time step, they generate a character-score, which is represented as a matrix. This matrix must now be used for the following tasks:

1. Calculating the loss and training the Neural Network
2. Decoding the Neural Network's Output

The CTC procedure aids in the completion of both jobs.

CTC is written in such a way that all that is required is the text that appears in the image. The width and position of the text in an image can be ignored. The output of the CTC procedure does not require any post-processing! We can immediately obtain the network's result using decoding techniques. Problem that can be avoided using CTC is depicted in Figure 3.6.



**Figure 3.6:** **Problem that can be avoided using CTC**

**Working of CTC:**

The ideas that CTC works on are:

1. Encoding the text
2. Loss calculation
3. Decoding

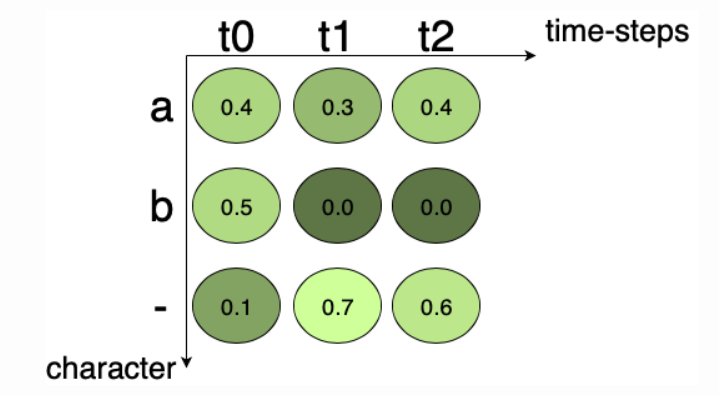
**Encoding the text**

What to do when the character takes more than one time-step in the image is a problem with approaches that do not use CTC. Non-CTC approaches would fail in this situation, resulting in repeated characters. CTC works around this by combining all of the recurring characters into a single character. For example, if the word in the image is 'hey,' the letter 'h' takes three time-steps, whereas the letters 'e' and 'y' each take one time-step. The network's output using CTC will then be 'hhhey,' which will be collapsed to 'hey' according to our encoding approach.

**Loss calculation:**

We must calculate loss given the image and its label in order to train the CRNN. The CRNN provides us with a matrix of each character's score at each time step. Figure 3 illustrates an example of a CRNN output matrix. There are three characters and three time-steps in this game (including one blank). The character score adds up to 1 at each time step.

The ground truth scores of all conceivable alignments are added together to calculate the loss. It makes no difference where the character appears in the image in this way.



**Figure 3.7: The Neural Network's output matrix. At each time step, it displays the character probability**.

The score for one path is determined by multiplying with the character scores together. The score for the path "a–" in Figure 3.7 is 0.4x0.7x0.6 = 0.168, while the score for the path "aaa" is 0.4x0.3x0.4 = 0.048. The scores of all the paths to the appropriate text are added together to generate the score that corresponds to the provided ground truth.

For example, if the ground truth is "a," all feasible pathways for "a" in Fig.3 are "aaa," "a–," "a-," "aa-," "-aa," and "–a." 0.048 + 0.168 + 0.018 + 0.072 + 0.012 + 0.028 = 0.346 when the individual path scores are added together. The likelihood of the ground truth happening is 0.346, not the loss. Because the loss is the negative logarithm of probability, it is simple to compute. This loss can be propagated backwards and the network taught.

**Decoding:**

We want CRNN to give us output on unseen text pictures once it has been trained. To put it another way, we want the most likely text given a CRNN output matrix. Examining all possible text output is one way, but it isn't really practical from a computation standpoint. To solve this problem, the best path algorithm is utilised. It comprises of the two phases listed below:

* Calculates the optimum path by taking the character with the highest probability into account at each time step.
* The actual text is created by deleting blanks and duplicate characters in this phase.

Take, for example, the matrix in Figure 3. In the initial time step, t0, the character with the highest probability is ‘b.' The characters having the highest likelihood for t1 and t2 are ‘-‘ and ‘-‘, respectively. So, after decoding, the output text according to the best path algorithm for matrix in Fig.3.7 is ‘b'.

**CHAPTER 4**

**EXPERIMENTAL RESULTS**

**4.1 Dataset:**

We have to build the Neural Network (NN) which is prepared on line-images from IAM (On-Line Handwriting Database) dataset. As the input layer and therefore all the other layers as well can be kept for small for line-images, NN-training is feasible on the CPU.

**Table 4.1: Dataset details**

| **Dataset** | **Training-set** | **Validation-set** | **Test-set** |
| --- | --- | --- | --- |
| IAM DATASET | 1364 | 80 | 160 |

It's a 128x32 pixel Gray-value image. Since the images in the dataset are rarely exactly this size, we scale them until they are either 128 pixels wide or 32 pixels tall. The image is then copied into a (white) target image with a size of 12832 pixels. Figure 3 depicts this procedure. Finally, we normalise the image's grey levels, making the task easier for the NN. By duplicating the image to irregular spots rather than aligning them to the left, or by randomly scaling the image, data augmentation can be simply implemented.

**4.2 Convolutional Neural Network (CNN):**

The CNN layers are fed the input image. These layers have been trained to extract important information from images. There are three operations in each layer. Firstly, there's the convolution operation is done, which takes the input and employs a filter kernel of size 5X5 in the first couple of layers and a kernel of size 3X3 in the last three layers. Following that, the non-linear RELU function is used. Lastly, a pooling layer condenses image regions and produces a smaller edition of the input. While the image height is reduced by two in each layer, feature maps are added, resulting in a 32x256 feature map (or sequence).

Figure 4.1 has the code snippet for CNN which mentions all the functions in CNN layer like RELU, Pooling, Normalization, 2D convolution.



**Figure 4.1: Code snippet of CNN**

For each Convolutional Neural Network layer, we make a kernel of size kXk to be utilized in the convolution operation.

Then the output after kernel operation, is feed into the RELU operation and then further carried forward again to the pooling layer with size px×py and step-size sx×sy.

These same steps are followed for all layers in a for-loop.

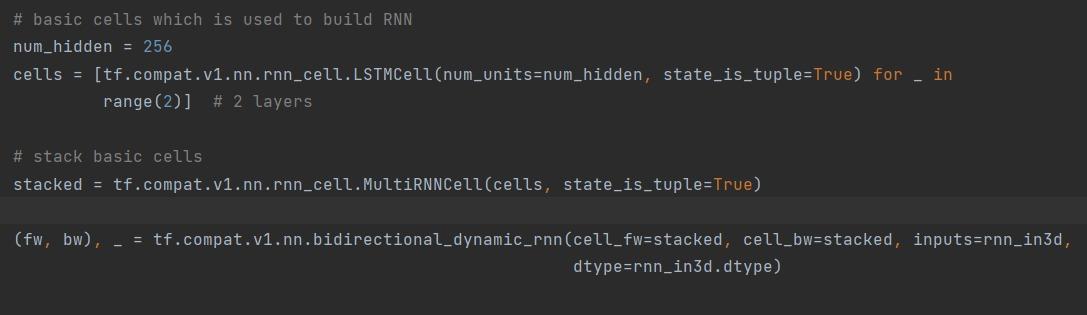
**CNN output:**

The CNN layers produce a 32-character sequence as their output. There are 256 features in each entry. Of course, the Recurrent Neural Network layers will treat these features further, but few of them already have a strong relationship with the input image's high-level attributes: for example, here are features that have a high similarity with characters (e.g. “e”), duplicate characters (e.g. “tt”), or character-properties such as loops (as found in handwritten “l”s or “e”s).

**4.3 Recurrent Neural Network:**

The RNN propagates important information along the feature pattern, which has about 256 features per each time-step. The popular Long Short-Term Memory (LSTM) RNN implementation is utilised since it can relay information over greater distances and has more strong training features than a vanilla RNN. The RNN output sequence is transferred to a 3280-by-3280 matrix. The IAM dataset contains exactly 79 different characters, plus one extra character for the CTC procedure (the CTC blank label), for a total of 80 entries for each of the 32 time-steps.

Figure 4.2 is the code snippet for RNN.



**Figure 4.2: Code snippet of RNN**

Create two RNN layers and then stack two RNN layers with 256 units each.

Then, from it, build a bi-directional RNN that traverses the input sequence from front to back and the other way around. As a result, we get two 32×256 output sequences, fw and bw, which we subsequently concatenate along the feature-axis to generate a 32×512 sequence. Finally, it is mapped to a 32×80 output sequence (or matrix) that is sent into the CTC layer.

**RNN output:**

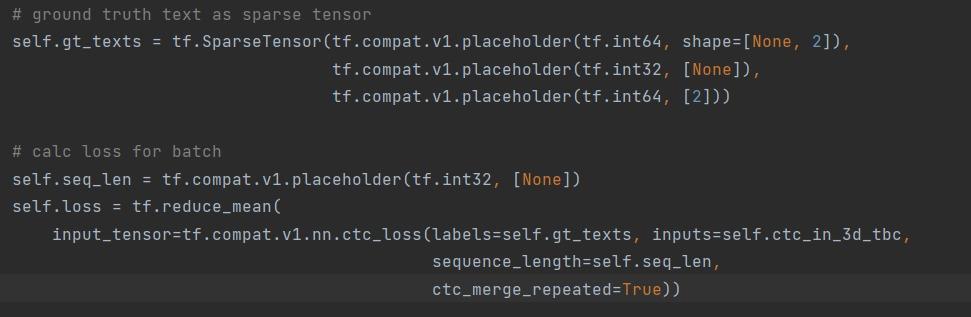
RNN output matrix bears the characters scores, with the CTC blank label as the final (80th) element. From top to bottom, the other matrix elements match to the characters listed below:

“!”./0123456789:;?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz”. It can be observed that the characters are almost always predicted precisely where they appear in the picture. The CTC procedure does not need segmentation and is unconcerned about absolute locations.

**4.4 Connectionist Temporal Classification (CTC):**

During the training of the NN, the CTC is given the RNN output matrix and the ground truth text, and it computes the loss value. The matrix to infer is simply handed to the CTC, which decodes it into the final text. The maximum length of both the ground truth and recognised messages is 32 characters.

Figure 4.3 has the code for CTC.



**Figure 4.3: Code snippet of CTC**

We input the operation both the ground truth text and the matrix for loss computation. A sparse tensor is used to encode the ground truth text. Both CTC processes need the length of the input sequences.

We now have all of the input data needed to perform the loss and decoding operations.

**4.5 Training:**



**Figure 4.4: Code snippet for training**

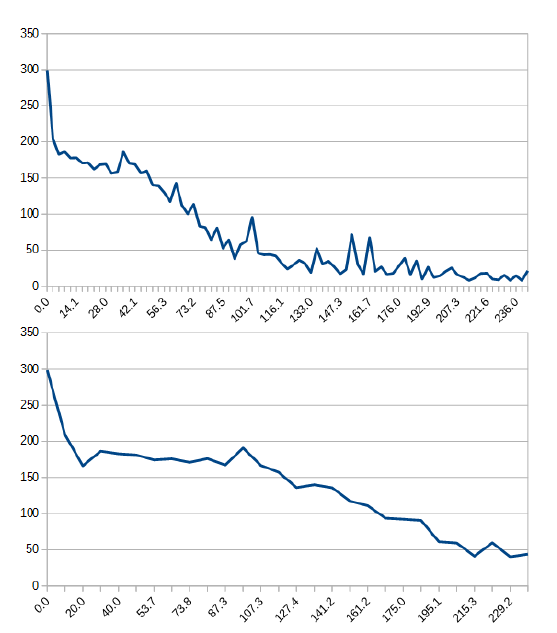
The mean of the batch element loss values is used to train the NN, which is then put into an optimizer like Adam Optimizer. This is shown in the above Figure 4.4.

**4.6 Model Evaluation:**

Model evaluation is an important step in the creation of a model. It aids in the selection of the best

model to represent our data and the prediction of how well the chosen model will perform in the future. Both techniques employ a test set (not visible to the model) to evaluate model performance in order to avoid overfitting.

The goal of model assessment is to determine a model's generalisation accuracy on future (unseen/out-of-sample) data. The graphs of Loss and Accuracy against epochs are plotted here.

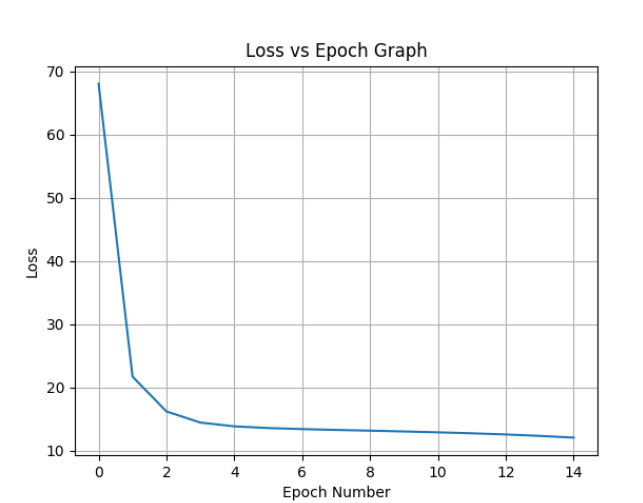
****

**Figure 4.5: Plot of loss function**

On the IAM dataset, a plot of the loss function for the first 4 minutes of training is shown in Figure 4.5. The time in seconds is represented on the horizontal axis, while the CTC loss is represented on the vertical axis.

Top image batch size = 5

Bottom image batch size = 25



**Figure 4.6: Loss vs Epoch Graph**

Loss vs Epoch graph is plotted in Figure 4.6.

The Character Error Rate (CER) is 10.72%, while the Word Error Rate (WER) is 26.45%, resulting in a Word Accuracy of 73.55%.

* 1. **Output images obtained on testing**

**Table 5.2: The test images along with output digitalized text**

| S.no. | Test Image |
| --- | --- |
| 1 | Recognized: "put down a resouution on the subject"  Probability: 0.3410680592060089 |
| 2 | Recognized: "A MOVE to stop Mr.Gaitskell from"  Probability: 0.4538659006357193 |
| 3 | Recognized: "nomiuating any more Labour life Peers"  Probability: 0.21466925740242 |
| 4 | Recognized: "is to be made at a meeting of Labour"  Probability: 0.66699731349945 |
| 5 | Recognized: "MPs omorrow"  Probability: 0.17748476564884186 |
| 6 | Recognized: "Mr.Mchael Foot has"  Probability: 0.4178291469812393 |
| 7 | Recognized: "and he is to be backed by Mr.Will"  Probability: 0.600307181477547 |
| 8 | Recognized: "Griffiths , MPfor Manchester Exchange ."  Probability: 0.25567521806806326 |

**4.8 Deployment:**

This model can be deployed in educational institutions, IT companies, Healthcare centres and hospitals, business industries to digitalize the handwritten information and directly store it in the database saving time and hard work.

Also, the project can be extended by adding algorithms to identify the strokes, pressure, slant and handwriting style. This model can be linked to any type of database like criminal database and police records to identify the trespasser, thus identifying and avoiding forgery and any such crimes.

Further, this project can be improved to identify the style of writing, i.e., the vocabulary used, the grammar used, repetition of words and phrases etc., to identify the author for unknown ancient scripts.

**CHAPTER 5**

**CONCLUSION AND FUTURE SCOPE**

**5.1 CONCLUSION:**

In this thesis, we have discussed about the Neural network which is able to recognise text in images.

The document also contains the explanation of all the methods used in the model with the help of flowchart. The NN generates a character-probability matrix and has 5 CNN and 2 RNN layers. This matrix is either used to calculate CTC loss or to decode CTC.

Some important parts of the codes are presented. Outputs of the handwritten images are tabulated.

**5.2 FUTURE SCOPE:**

To feed a complete page, instead of line images, we can increase the input size of the Neural Network. Also, to improve the recognition accuracy, some of the following methods can be employed:

* Data augmentation: We can increase the dataset size by randomly applying transformations
* to the already existing input images.
* Deslanting : The cursive writing style in the input photos can be removed.
* Adding a greater number of CNN layers.
* Replacing LSTM by 2D-LSTM.
* Decoder: To limit the output to dictionary terms, utilise token passing or word beam search decoding.
* Text correction: If the identified word isn't found in a dictionary, look for the closest match.

This model can be deployed in educational institutions, IT companies, Healthcare centers and hospitals, business industries to digitalize the handwritten information and directly store it in the database saving time and hard work.

Also, the project can be extended by adding algorithms to identify the strokes, pressure, slant and handwriting style. This model can be linked to any type of database like criminal database and police records to identify the trespasser, thus identifying and avoiding forgery and any such crimes.

Further, this project can be improved to identify the style of writing, i.e., the vocabulary used, the grammar used, repetition of words and phrases etc., to identify the author for unknown ancient scripts.

**References:**

| [1] | K. Gaurav and Bhatia P. K., “Analytical Review of Pre-processing Techniques for Offline Handwritten Character Recognition”, 2nd International Conference on Emerging Trends in Engineering & Management, ICETEM, 2013. |
| --- | --- |
| [2] | Salvador España-Boquera, Maria J. C. B., Jorge G. M. and Francisco Z. M., “Improving Offline Handwritten Text Recognition with Hybrid HMM/ANN Models”, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 33, No. 4, April 2011. |
| [3] | U. Pal, T. Wakabayashi and F. Kimura, “Handwritten numeral recognition of six popular scripts,” Ninth International conference on Document Analysis and Recognition ICDAR 07, Vol.2, pp.749-753, 2007 |
| [4] | J. Pradeep, E. Srinivasan and S. Himavathi, “Diagonal Based Feature Extraction For Handwritten Alphabets Recognition System Using Neural Network”, International Journal of Computer Science & Information Technology (IJCSIT), Vol 3, No 1, Feb 2011. |
| [5] | A. Brakensiek, J. Rottland, A. Kosmala and J. Rigoll, “Offline Handwriting Recognition using various Hybrid Modeling Techniques & Character N-Grams |
| [6] | Sandhya Arora, “Combining Multiple Feature Extraction Techniques for Handwritten Devnagari Character Recognition”, IEEE Region 10 Colloquium and the Third ICIIS, Kharagpur, INDIA, December 2008. |
| [7] | Nafiz Arica, and Fatos T. Yarman-Vural, ―Optical Character Recognition for Cursive Handwriting,ǁ IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.24, no.6, pp. 801-113, June 2002 |
| [8] | M. Hanmandlu, O.V. Ramana Murthy, “Fuzzy model-based recognition of handwritten numerals”, pattern recognition, vol.40, pp.1840-1854, 2007. |
| [9] | T.Som, Sumit Saha, "Handwritten Character Recognition Using Fuzzy Membership Function", International Journal of Emerging Technologies in Sciences and Engineering, Vol.5, No.2, pp. 11-15, Dec 2011. |
| [10] | Reena Bajaj, Lipika Dey, and S. Chaudhury, “Devnagari numeral recognition by combining decision of multiple connectionist classifiers”, Sadhana, Vol.27, part. 1, pp.-59-72, 2002. |
| [11] | Mohammed Z. Khedher, Gheith A. Abandah, and Ahmed M. AlKhawaldeh, “Optimizing Feature Selection for Recognizing Handwritten Arabic Characters”, proceedings of World Academy of Science Engineering and Technology, vol. 4, February 2005 ISSN 1307-6884. |
| [12] | Sushree Sangita Patnaik and Anup Kumar Panda, “Particle Swarm Optimization and Bacterial Foraging Optimization Techniques for Optimal Current Harmonic Mitigation by Employing Active Power Filter Applied Computational Intelligence and Soft Computing”, Volume 2012, Article ID 897127. |
| [13] | K. H. Aparna,, Vidhya Subramanian, M. Kasirajan, G. Vijay Prakash, V. S. Chakravarthy, Sriganesh Madhvanath, "Online Handwriting Recognition for Tamil", IWFHR, 2004, Proceedings. Ninth International Workshop on Frontiers in Handwriting Recognition, Proceedings. Ninth International Workshop on Frontiers in Handwriting Recognition 2004, pp. 438-443, doi:10.1109/IWFHR.2004. |
| [14] | G. Pirlo and D. Impedovo, “Fuzzy Zoning Based Classification for Handwritten Characters”, IEEE Transaction on pattern Recognition and Machine Intelligence, vol.19, no. 04, pp.780-785, August 2011. |
| [15] | Yoshimasa Kimura, "Feature Selection for Character Recognition Using Genetic Algorithm", Fourth International Conference on Innovative Computing, Information and Control 978-0-7695-3873-0/09© 2009 IEEE, 2009. |

**ANNEXURE:**

**Model.py**: file that generates the above mentioned model, loads and saves models, manages TF sessions, and offers a training and inference interface

import os  
import sys  
from typing import List, Tuple  
  
import numpy as np  
import tensorflow as tf  
  
from dataloader\_iam import Batch  
  
tf.compat.v1.disable\_eager\_execution()  
  
  
class DecoderType:  
 BestPath = 0  
 BeamSearch = 1  
 WordBeamSearch = 2  
  
  
class Model:  
 def \_\_init\_\_(self,  
 char\_list: List[str],  
 decoder\_type: str = DecoderType.BestPath,  
 must\_restore: bool = False,  
 dump: bool = False) -> None:  
 self.dump = dump  
 self.char\_list = char\_list  
 self.decoder\_type = decoder\_type  
 self.must\_restore = must\_restore  
 self.snap\_ID = 0  
  
 self.is\_train = tf.compat.v1.placeholder(tf.bool, name=**'is\_train'**)  
  
self.input\_imgs = tf.compat.v1.placeholder(tf.float32, shape=(None, None, None))  
  
self.setup\_cnn()  
 self.setup\_rnn()  
 self.setup\_ctc()  
self.batches\_trained = 0  
 self.update\_ops = tf.compat.v1.get\_collection(tf.compat.v1.GraphKeys.UPDATE\_OPS)  
 with tf.control\_dependencies(self.update\_ops):  
 self.optimizer = tf.compat.v1.train.AdamOptimizer().minimize(self.loss)  
self.sess, self.saver = self.setup\_tf()  
  
 def setup\_cnn(self) -> None:  
cnn\_in4d = tf.expand\_dims(input=self.input\_imgs, axis=3)  
  
kernel\_vals = [5, 5, 3, 3, 3]  
 feature\_vals = [1, 32, 64, 128, 128, 256]  
 stride\_vals = pool\_vals = [(2, 2), (2, 2), (1, 2), (1, 2), (1, 2)]  
 num\_layers = len(stride\_vals) *#5* pool = cnn\_in4d

for i in range(num\_layers):  
 kernel = tf.Variable(  
 tf.random.truncated\_normal([kernel\_vals[i], kernel\_vals[i], feature\_vals[i], feature\_vals[i + 1]],  
 stddev=0.1))  
 conv = tf.nn.conv2d(input=pool, filters=kernel, padding=**'SAME'**, strides=(1, 1, 1, 1))  
 conv\_norm = tf.compat.v1.layers.batch\_normalization(conv, training=self.is\_train)  
 relu = tf.nn.relu(conv\_norm)  
 pool = tf.nn.max\_pool2d(input=relu, ksize=(1, pool\_vals[i][0], pool\_vals[i][1], 1),  
 strides=(1, stride\_vals[i][0], stride\_vals[i][1], 1), padding=**'VALID'**)  
  
 self.cnn\_out\_4d = pool  
  
 def setup\_rnn(self) -> None:  
 rnn\_in3d = tf.squeeze(self.cnn\_out\_4d, axis=[2])  
  
num\_hidden = 256  
 cells = [tf.compat.v1.nn.rnn\_cell.LSTMCell(num\_units=num\_hidden, state\_is\_tuple=True) for \_ in  
 range(2)] *# 2 layers* stacked = tf.compat.v1.nn.rnn\_cell.MultiRNNCell(cells, state\_is\_tuple=True)  
  
 (fw, bw), \_ = tf.compat.v1.nn.bidirectional\_dynamic\_rnn(cell\_fw=stacked, cell\_bw=stacked, inputs=rnn\_in3d,  
 dtype=rnn\_in3d.dtype)  
  
  
 concat = tf.expand\_dims(tf.concat([fw, bw], 2), 2)  
  
  
 kernel = tf.Variable(tf.random.truncated\_normal([1, 1, num\_hidden \* 2, len(self.char\_list) + 1], stddev=0.1))  
 self.rnn\_out\_3d = tf.squeeze(tf.nn.atrous\_conv2d(value=concat, filters=kernel, rate=1, padding=**'SAME'**),  
 axis=[2])  
  
 def setup\_ctc(self) -> None:  
self.ctc\_in\_3d\_tbc = tf.transpose(a=self.rnn\_out\_3d, perm=[1, 0, 2])  
self.gt\_texts = tf.SparseTensor(tf.compat.v1.placeholder(tf.int64, shape=[None, 2]),  
 tf.compat.v1.placeholder(tf.int32, [None]),  
 tf.compat.v1.placeholder(tf.int64, [2]))  
  
self.seq\_len = tf.compat.v1.placeholder(tf.int32, [None])  
 self.loss = tf.reduce\_mean(  
 input\_tensor=tf.compat.v1.nn.ctc\_loss(labels=self.gt\_texts, inputs=self.ctc\_in\_3d\_tbc,  
 sequence\_length=self.seq\_len,  
 ctc\_merge\_repeated=True))  
  
self.saved\_ctc\_input = tf.compat.v1.placeholder(tf.float32,  
 shape=[None, None, len(self.char\_list) + 1])  
 self.loss\_per\_element = tf.compat.v1.nn.ctc\_loss(labels=self.gt\_texts, inputs=self.saved\_ctc\_input,  
 sequence\_length=self.seq\_len, ctc\_merge\_repeated=True)  
  
if self.decoder\_type == DecoderType.BestPath:  
 self.decoder = tf.nn.ctc\_greedy\_decoder(inputs=self.ctc\_in\_3d\_tbc, sequence\_length=self.seq\_len)  
 elif self.decoder\_type == DecoderType.BeamSearch:  
 self.decoder = tf.nn.ctc\_beam\_search\_decoder(inputs=self.ctc\_in\_3d\_tbc, sequence\_length=self.seq\_len,beam\_width=50)  
 elif self.decoder\_type == DecoderType.WordBeamSearch:  
 chars = **''**.join(self.char\_list)  
 word\_chars = open(**'../model/wordCharList.txt'**).read().splitlines()[0]  
 corpus = open(**'../data/corpus.txt'**).read()  
  
 from word\_beam\_search import WordBeamSearch  
 self.decoder = WordBeamSearch(50, **'Words'**, 0.0, corpus.encode(**'utf8'**), chars.encode(**'utf8'**),  
 word\_chars.encode(**'utf8'**))  
  
 self.wbs\_input = tf.nn.softmax(self.ctc\_in\_3d\_tbc, axis=2)  
  
 def setup\_tf(self) -> Tuple[tf.compat.v1.Session, tf.compat.v1.train.Saver]:  
 print(**'Python: '** + sys.version)  
 print(**'Tensorflow: '** + tf.\_\_version\_\_)  
  
 sess = tf.compat.v1.Session() *# TF session* saver = tf.compat.v1.train.Saver(max\_to\_keep=1)

model\_dir = **'../model/'** latest\_snapshot = tf.train.latest\_checkpoint(model\_dir)if self.must\_restore and not latest\_snapshot:  
 raise Exception(**'No saved model found in: '** + model\_dir)  
  
if latest\_snapshot:  
 print(**'Init with stored values from '** + latest\_snapshot)  
 saver.restore(sess, latest\_snapshot)  
 else:  
 print(**'Init with new values'**)  
 sess.run(tf.compat.v1.global\_variables\_initializer())  
  
 return sess, saver  
  
 def to\_sparse(self, texts: List[str]) -> Tuple[List[List[int]], List[int], List[int]]:  
indices = []  
 values = []  
 shape = [len(texts), 0] *# last entry must be max(labelList[i])* for batchElement, text in enumerate(texts):  
 *# convert to string of label (i.e. class-ids)* label\_str = [self.char\_list.index(c) for c in text]  
 *# sparse tensor must have size of max. label-string* if len(label\_str) > shape[1]:  
 shape[1] = len(label\_str)  
 *# put each label into sparse tensor* for i, label in enumerate(label\_str):  
 indices.append([batchElement, i])  
 values.append(label)  
  
 return indices, values, shape  
  
 def decoder\_output\_to\_text(self, ctc\_output: tuple, batch\_size: int) -> List[str]:  
 *"""Extract texts from output of CTC decoder."""  
  
 # word beam search: already contains label strings* if self.decoder\_type == DecoderType.WordBeamSearch:  
 label\_strs = ctc\_output  
  
 *# TF decoders: label strings are contained in sparse tensor* else:  
 *# ctc returns tuple, first element is SparseTensor* decoded = ctc\_output[0][0]  
  
 *# contains string of labels for each batch element* label\_strs = [[] for \_ in range(batch\_size)]  
  
 *# go over all indices and save mapping: batch -> values* for (idx, idx2d) in enumerate(decoded.indices):  
 label = decoded.values[idx]  
 batch\_element = idx2d[0] *# index according to [b,t]* label\_strs[batch\_element].append(label)  
  
 *# map labels to chars for all batch elements* return [**''**.join([self.char\_list[c] for c in labelStr]) for labelStr in label\_strs]  
  
 def train\_batch(self, batch: Batch) -> float:  
 *"""Feed a batch into the NN to train it."""* num\_batch\_elements = len(batch.imgs)  
 max\_text\_len = batch.imgs[0].shape[0] // 4  
 sparse = self.to\_sparse(batch.gt\_texts)  
 eval\_list = [self.optimizer, self.loss]  
 feed\_dict = {self.input\_imgs: batch.imgs, self.gt\_texts: sparse,  
 self.seq\_len: [max\_text\_len] \* num\_batch\_elements, self.is\_train: True}  
 \_, loss\_val = self.sess.run(eval\_list, feed\_dict)  
 self.batches\_trained += 1  
 return loss\_val  
  
 @staticmethod  
 def dump\_nn\_output(rnn\_output: np.ndarray) -> None:  
 *"""Dump the output of the NN to CSV file(s)."""* dump\_dir = **'../dump/'** if not os.path.isdir(dump\_dir):  
 os.mkdir(dump\_dir)  
  
 *# iterate over all batch elements and create a CSV file for each one* max\_t, max\_b, max\_c = rnn\_output.shape  
 for b in range(max\_b):  
 csv = **''** for t in range(max\_t):  
 for c in range(max\_c):  
 csv += str(rnn\_output[t, b, c]) + **';'** csv += **'**\n**'** fn = dump\_dir + **'rnnOutput\_'** + str(b) + **'.csv'** print(**'Write dump of NN to file: '** + fn)  
 with open(fn, **'w'**) as f:  
 f.write(csv)  
  
 def infer\_batch(self, batch: Batch, calc\_probability: bool = False, probability\_of\_gt: bool = False):  
 *"""Feed a batch into the NN to recognize the texts."""  
  
 # decode, optionally save RNN output* num\_batch\_elements = len(batch.imgs)  
  
 *# put tensors to be evaluated into list* eval\_list = []  
  
 if self.decoder\_type == DecoderType.WordBeamSearch:  
 eval\_list.append(self.wbs\_input)  
 else:  
 eval\_list.append(self.decoder)  
  
 if self.dump or calc\_probability:  
 eval\_list.append(self.ctc\_in\_3d\_tbc)  
  
 *# sequence length depends on input image size (model downsizes width by 4)* max\_text\_len = batch.imgs[0].shape[0] // 4  
  
 *# dict containing all tensor fed into the model* feed\_dict = {self.input\_imgs: batch.imgs, self.seq\_len: [max\_text\_len] \* num\_batch\_elements,  
 self.is\_train: False}  
  
 *# evaluate model* eval\_res = self.sess.run(eval\_list, feed\_dict)  
  
 *# TF decoders: decoding already done in TF graph* if self.decoder\_type != DecoderType.WordBeamSearch:  
 decoded = eval\_res[0]  
 *# word beam search decoder: decoding is done in C++ function compute()* else:  
 decoded = self.decoder.compute(eval\_res[0])  
  
 *# map labels (numbers) to character string* texts = self.decoder\_output\_to\_text(decoded, num\_batch\_elements)  
  
 *# feed RNN output and recognized text into CTC loss to compute labeling probability* probs = None  
 if calc\_probability:  
 sparse = self.to\_sparse(batch.gt\_texts) if probability\_of\_gt else self.to\_sparse(texts)  
 ctc\_input = eval\_res[1]  
 eval\_list = self.loss\_per\_element  
 feed\_dict = {self.saved\_ctc\_input: ctc\_input, self.gt\_texts: sparse,  
 self.seq\_len: [max\_text\_len] \* num\_batch\_elements, self.is\_train: False}  
 loss\_vals = self.sess.run(eval\_list, feed\_dict)  
 probs = np.exp(-loss\_vals)  
  
 *# dump the output of the NN to CSV file(s)* if self.dump:  
 self.dump\_nn\_output(eval\_res[1])  
  
 return texts, probs  
  
 def save(self) -> None:  
 *"""Save model to file."""* self.snap\_ID += 1  
 self.saver.save(self.sess, **'../model/snapshot'**, global\_step=self.snap\_ID)

**Preprocessor.py:** prepares the images from the IAM dataset for the NN

import random  
from typing import Tuple  
  
import cv2  
import numpy as np  
  
from dataloader\_iam import Batch  
  
  
class Preprocessor:  
 def \_\_init\_\_(self,  
 img\_size: Tuple[int, int],  
 padding: int = 0,  
 dynamic\_width: bool = False,  
 data\_augmentation: bool = False,  
 line\_mode: bool = False) -> None:  
 *# dynamic width only supported when no data augmentation happens* assert not (dynamic\_width and data\_augmentation)  
 *# when padding is on, we need dynamic width enabled* assert not (padding > 0 and not dynamic\_width)  
  
 self.img\_size = img\_size  
 self.padding = padding  
 self.dynamic\_width = dynamic\_width  
 self.data\_augmentation = data\_augmentation  
 self.line\_mode = line\_mode  
  
 @staticmethod  
 def \_truncate\_label(text: str, max\_text\_len: int) -> str:

cost = 0  
 for i in range(len(text)):  
 if i != 0 and text[i] == text[i - 1]:  
 cost += 2  
 else:  
 cost += 1  
 if cost > max\_text\_len:  
 return text[:i]  
 return text  
  
 def \_simulate\_text\_line(self, batch: Batch) -> Batch:  
default\_word\_sep = 30  
 default\_num\_words = 5  
  
 *# go over all batch elements* res\_imgs = []  
 res\_gt\_texts = []  
 for i in range(batch.batch\_size):  
 *# number of words to put into current line* num\_words = random.randint(1, 8) if self.data\_augmentation else default\_num\_words  
  
 *# concat ground truth texts* curr\_gt = **' '**.join([batch.gt\_texts[(i + j) % batch.batch\_size] for j in range(num\_words)])  
 res\_gt\_texts.append(curr\_gt)  
  
 *# put selected word images into list, compute target image size* sel\_imgs = []  
 word\_seps = [0]  
 h = 0  
 w = 0  
 for j in range(num\_words):  
 curr\_sel\_img = batch.imgs[(i + j) % batch.batch\_size]  
 curr\_word\_sep = random.randint(20, 50) if self.data\_augmentation else default\_word\_sep  
 h = max(h, curr\_sel\_img.shape[0])  
 w += curr\_sel\_img.shape[1]  
 sel\_imgs.append(curr\_sel\_img)  
 if j + 1 < num\_words:  
 w += curr\_word\_sep  
 word\_seps.append(curr\_word\_sep)  
  
 *# put all selected word images into target image* target = np.ones([h, w], np.uint8) \* 255  
 x = 0  
 for curr\_sel\_img, curr\_word\_sep in zip(sel\_imgs, word\_seps):  
 x += curr\_word\_sep  
 y = (h - curr\_sel\_img.shape[0]) // 2  
 target[y:y + curr\_sel\_img.shape[0]:, x:x + curr\_sel\_img.shape[1]] = curr\_sel\_img  
 x += curr\_sel\_img.shape[1]  
  
 *# put image of line into result* res\_imgs.append(target)  
  
 return Batch(res\_imgs, res\_gt\_texts, batch.batch\_size)  
  
 def process\_img(self, img: np.ndarray) -> np.ndarray:  
 *"""Resize to target size, apply data augmentation. 128\*32"""  
  
 # there are damaged files in IAM dataset - just use black image instead* if img is None:  
 img = np.zeros(self.img\_size[::-1])  
  
 *# data augmentation* img = img.astype(np.float)  
 if self.data\_augmentation:  
 *# photometric data augmentation* if random.random() < 0.25:  
 def rand\_odd():  
 return random.randint(1, 3) \* 2 + 1  
 img = cv2.GaussianBlur(img, (rand\_odd(), rand\_odd()), 0)  
 if random.random() < 0.25:  
 img = cv2.dilate(img, np.ones((3, 3)))  
 if random.random() < 0.25:  
 img = cv2.erode(img, np.ones((3, 3)))  
  
 *# geometric data augmentation* wt, ht = self.img\_size  
 h, w = img.shape  
 f = min(wt / w, ht / h)  
 fx = f \* np.random.uniform(0.75, 1.05)  
 fy = f \* np.random.uniform(0.75, 1.05)  
  
 *# random position around center* txc = (wt - w \* fx) / 2  
 tyc = (ht - h \* fy) / 2  
 freedom\_x = max((wt - fx \* w) / 2, 0)  
 freedom\_y = max((ht - fy \* h) / 2, 0)  
 tx = txc + np.random.uniform(-freedom\_x, freedom\_x)  
 ty = tyc + np.random.uniform(-freedom\_y, freedom\_y)  
  
 *# map image into target image* M = np.float32([[fx, 0, tx], [0, fy, ty]])  
 target = np.ones(self.img\_size[::-1]) \* 255  
 img = cv2.warpAffine(img, M, dsize=self.img\_size, dst=target, borderMode=cv2.BORDER\_TRANSPARENT)  
  
 *# photometric data augmentation* if random.random() < 0.5:  
 img = img \* (0.25 + random.random() \* 0.75)  
 if random.random() < 0.25:  
 img = np.clip(img + (np.random.random(img.shape) - 0.5) \* random.randint(1, 25), 0, 255)  
 if random.random() < 0.1:  
 img = 255 - img  
  
 *# no data augmentation* else:  
 if self.dynamic\_width:  
 ht = self.img\_size[1]  
 h, w = img.shape  
 f = ht / h  
 wt = int(f \* w + self.padding)  
 wt = wt + (4 - wt) % 4  
 tx = (wt - w \* f) / 2  
 ty = 0  
 else:  
 wt, ht = self.img\_size  
 h, w = img.shape  
 f = min(wt / w, ht / h)  
 tx = (wt - w \* f) / 2  
 ty = (ht - h \* f) / 2  
  
 *# map image into target image* M = np.float32([[f, 0, tx], [0, f, ty]])  
 target = np.ones([ht, wt]) \* 255  
 img = cv2.warpAffine(img, M, dsize=(wt, ht), dst=target, borderMode=cv2.BORDER\_TRANSPARENT)  
  
 *# transpose for TF* img = cv2.transpose(img)  
  
 *# convert to range [-1, 1]* img = img / 255 - 0.5  
 return img  
  
 def process\_batch(self, batch: Batch) -> Batch:  
 if self.line\_mode:  
 batch = self.\_simulate\_text\_line(batch)  
  
 res\_imgs = [self.process\_img(img) for img in batch.imgs]  
 max\_text\_len = res\_imgs[0].shape[0] // 4  
 res\_gt\_texts = [self.\_truncate\_label(gt\_text, max\_text\_len) for gt\_text in batch.gt\_texts]  
 return Batch(res\_imgs, res\_gt\_texts, batch.batch\_size)  
  
  
def main():  
 import matplotlib.pyplot as plt  
  
 img = cv2.imread(**'../data/test06.png'**, cv2.IMREAD\_GRAYSCALE)  
 img\_aug = Preprocessor((256, 32), data\_augmentation=True).process\_img(img)  
 plt.subplot(121)  
 plt.imshow(img, cmap=**'gray'**)  
 plt.subplot(122)  
 plt.imshow(cv2.transpose(img\_aug) + 0.5, cmap=**'gray'**, vmin=0, vmax=1)  
 plt.show()  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()

**DataLoader.py:** reads samples, puts them into batches and provides an iterator-interface to go through the data

import pickle  
import random  
from collections import namedtuple  
from typing import Tuple  
  
import cv2  
import lmdb  
import numpy as np  
from path import Path  
  
Sample = namedtuple(**'Sample'**, **'gt\_text, file\_path'**)  
Batch = namedtuple(**'Batch'**, **'imgs, gt\_texts, batch\_size'**)  
  
  
class DataLoaderIAM:  
 def \_\_init\_\_(self,  
 data\_dir: Path,  
 batch\_size: int,  
 data\_split: float = 0.95,  
 fast: bool = True) -> None:  
 *"""Loader for dataset."""* assert data\_dir.exists()  
  
 self.fast = fast  
 if fast:  
 self.env = lmdb.open(str(data\_dir / **'lmdb'**), readonly=True)  
  
 self.data\_augmentation = False  
 self.curr\_idx = 0  
 self.batch\_size = batch\_size  
 self.samples = []  
  
 f = open(data\_dir / **'gt/words.txt'**)  
 chars = set()  
 bad\_samples\_reference = [**'a01-117-05-02'**, **'r06-022-03-05'**] *# known broken images in IAM dataset* for line in f:  
 *# ignore comment line* if not line or line[0] == **'#'**:  
 continue  
  
 line\_split = line.strip().split(**' '**)  
 assert len(line\_split) >= 9  
  
 *# filename: part1-part2-part3 --> part1/part1-part2/part1-part2-part3.png* file\_name\_split = line\_split[0].split(**'-'**)  
 file\_name\_subdir1 = file\_name\_split[0]  
 file\_name\_subdir2 = **f'**{file\_name\_split[0]}**-**{file\_name\_split[1]}**'** file\_base\_name = line\_split[0] + **'.png'** file\_name = data\_dir / **'img'** / file\_name\_subdir1 / file\_name\_subdir2 / file\_base\_name  
  
 if line\_split[0] in bad\_samples\_reference:  
 print(**'Ignoring known broken image:'**, file\_name)  
 continue  
  
 *# GT text are columns starting at 9* gt\_text = **' '**.join(line\_split[8:])  
 chars = chars.union(set(list(gt\_text)))  
  
 *# put sample into list* self.samples.append(Sample(gt\_text, file\_name))  
  
 *# split into training and validation set: 95% - 5%* split\_idx = int(data\_split \* len(self.samples))  
 self.train\_samples = self.samples[:split\_idx]  
 self.validation\_samples = self.samples[split\_idx:]  
  
 *# put words into lists* self.train\_words = [x.gt\_text for x in self.train\_samples]  
 self.validation\_words = [x.gt\_text for x in self.validation\_samples]  
  
 *# start with train set* self.train\_set()  
  
 *# list of all chars in dataset* self.char\_list = sorted(list(chars))  
  
 def train\_set(self) -> None:  
 *"""Switch to randomly chosen subset of training set."""* self.data\_augmentation = True  
 self.curr\_idx = 0  
 random.shuffle(self.train\_samples)  
 self.samples = self.train\_samples  
 self.curr\_set = **'train'** def validation\_set(self) -> None:  
 *"""Switch to validation set."""* self.data\_augmentation = False  
 self.curr\_idx = 0  
 self.samples = self.validation\_samples  
 self.curr\_set = **'val'** def get\_iterator\_info(self) -> Tuple[int, int]:  
 *"""Current batch index and overall number of batches."""* if self.curr\_set == **'train'**:  
 num\_batches = int(np.floor(len(self.samples) / self.batch\_size)) *# train set: only full-sized batches* else:  
 num\_batches = int(np.ceil(len(self.samples) / self.batch\_size)) *# val set: allow last batch to be smaller* curr\_batch = self.curr\_idx // self.batch\_size + 1  
 return curr\_batch, num\_batches  
  
 def has\_next(self) -> bool:  
 *"""Is there a next element?"""* if self.curr\_set == **'train'**:  
 return self.curr\_idx + self.batch\_size <= len(self.samples) *# train set: only full-sized batches* else:  
 return self.curr\_idx < len(self.samples) *# val set: allow last batch to be smaller* def \_get\_img(self, i: int) -> np.ndarray:  
 if self.fast:  
 with self.env.begin() as txn:  
 basename = Path(self.samples[i].file\_path).basename()  
 data = txn.get(basename.encode(**"ascii"**))  
 img = pickle.loads(data)  
 else:  
 img = cv2.imread(self.samples[i].file\_path, cv2.IMREAD\_GRAYSCALE)  
  
 return img  
  
 def get\_next(self) -> Batch:  
 *"""Get next element."""* batch\_range = range(self.curr\_idx, min(self.curr\_idx + self.batch\_size, len(self.samples)))  
  
 imgs = [self.\_get\_img(i) for i in batch\_range]  
 gt\_texts = [self.samples[i].gt\_text for i in batch\_range]  
  
 self.curr\_idx += self.batch\_size  
 return Batch(imgs, gt\_texts, len(imgs))

**main.py:** puts all previously mentioned modules together

import argparse  
import json  
from typing import Tuple, List  
  
import cv2  
import editdistance  
from path import Path  
  
from dataloader\_iam import DataLoaderIAM, Batch  
from model import Model, DecoderType  
from preprocessor import Preprocessor  
  
  
class FilePaths:  
 *"""Filenames and paths to data."""* fn\_char\_list = **'../model/charList.txt'** fn\_summary = **'../model/summary.json'** fn\_corpus = **'../data/corpus.txt'**def get\_img\_height() -> int:  
 *"""Fixed height for NN."""* return 32  
  
  
def get\_img\_size(line\_mode: bool = False) -> Tuple[int, int]:  
 *"""Height is fixed for NN, width is set according to training mode (single words or text lines)."""* if line\_mode:  
 return 256, get\_img\_height()  
 return 128, get\_img\_height()  
  
  
def write\_summary(char\_error\_rates: List[float], word\_accuracies: List[float]) -> None:  
 *"""Writes training summary file for NN."""* with open(FilePaths.fn\_summary, **'w'**) as f:  
 json.dump({**'charErrorRates'**: char\_error\_rates, **'wordAccuracies'**: word\_accuracies}, f)  
  
  
def train(model: Model,  
 loader: DataLoaderIAM,  
 line\_mode: bool,  
 early\_stopping: int = 25) -> None:  
 *"""Trains NN."""* epoch = 0 *# number of training epochs since start* summary\_char\_error\_rates = []  
 summary\_word\_accuracies = []  
 preprocessor = Preprocessor(get\_img\_size(line\_mode), data\_augmentation=True, line\_mode=line\_mode)  
 best\_char\_error\_rate = float(**'inf'**) *# best valdiation character error rate* no\_improvement\_since = 0 *# number of epochs no improvement of character error rate occurred  
 # stop training after this number of epochs without improvement* while True:  
 epoch += 1  
 print(**'Epoch:'**, epoch)  
  
 *# train* print(**'Train NN'**)  
 loader.train\_set()  
 while loader.has\_next():  
 iter\_info = loader.get\_iterator\_info()  
 batch = loader.get\_next()  
 batch = preprocessor.process\_batch(batch)  
 loss = model.train\_batch(batch)  
 print(**f'Epoch:** {epoch} **Batch:** {iter\_info[0]}**/**{iter\_info[1]} **Loss:** {loss}**'**)  
  
 *# validate* char\_error\_rate, word\_accuracy = validate(model, loader, line\_mode)  
  
 *# write summary* summary\_char\_error\_rates.append(char\_error\_rate)  
 summary\_word\_accuracies.append(word\_accuracy)  
 write\_summary(summary\_char\_error\_rates, summary\_word\_accuracies)  
  
 *# if best validation accuracy so far, save model parameters* if char\_error\_rate < best\_char\_error\_rate:  
 print(**'Character error rate improved, save model'**)  
 best\_char\_error\_rate = char\_error\_rate  
 no\_improvement\_since = 0  
 model.save()  
 else:  
 print(**f'Character error rate not improved, best so far:** {char\_error\_rate \* 100.0}**%'**)  
 no\_improvement\_since += 1  
  
 *# stop training if no more improvement in the last x epochs* if no\_improvement\_since >= early\_stopping:  
 print(**f'No more improvement since** {early\_stopping} **epochs. Training stopped.'**)  
 break  
  
  
def validate(model: Model, loader: DataLoaderIAM, line\_mode: bool) -> Tuple[float, float]:  
 *"""Validates NN."""* print(**'Validate NN'**)  
 loader.validation\_set()  
 preprocessor = Preprocessor(get\_img\_size(line\_mode), line\_mode=line\_mode)  
 num\_char\_err = 0  
 num\_char\_total = 0  
 num\_word\_ok = 0  
 num\_word\_total = 0  
 while loader.has\_next():  
 iter\_info = loader.get\_iterator\_info()  
 print(**f'Batch:** {iter\_info[0]} **/** {iter\_info[1]}**'**)  
 batch = loader.get\_next()  
 batch = preprocessor.process\_batch(batch)  
 recognized, \_ = model.infer\_batch(batch)  
  
 print(**'Ground truth -> Recognized'**)  
 for i in range(len(recognized)):  
 num\_word\_ok += 1 if batch.gt\_texts[i] == recognized[i] else 0  
 num\_word\_total += 1  
 dist = editdistance.eval(recognized[i], batch.gt\_texts[i])  
 num\_char\_err += dist  
 num\_char\_total += len(batch.gt\_texts[i])  
 print(**'[OK]'** if dist == 0 else **'[ERR:%d]'** % dist, **'"'** + batch.gt\_texts[i] + **'"'**, **'->'**,  
 **'"'** + recognized[i] + **'"'**)  
  
 *# print validation result* char\_error\_rate = num\_char\_err / num\_char\_total  
 word\_accuracy = num\_word\_ok / num\_word\_total  
 print(**f'Character error rate:** {char\_error\_rate \* 100.0}**%. Word accuracy:** {word\_accuracy \* 100.0}**%.'**)  
 return char\_error\_rate, word\_accuracy  
  
  
def infer(model: Model, fn\_img: Path) -> None:  
 *"""Recognizes text in image provided by file path."""* img = cv2.imread(fn\_img, cv2.IMREAD\_GRAYSCALE)  
 assert img is not None  
  
 preprocessor = Preprocessor(get\_img\_size(), dynamic\_width=True, padding=16)  
 img = preprocessor.process\_img(img)  
  
 batch = Batch([img], None, 1)  
 recognized, probability = model.infer\_batch(batch, True)  
 print(**f'Recognized: "**{recognized[0]}**"'**)  
 print(**f'Probability:** {probability[0]}**'**)  
  
  
def main():  
 *"""Main function."""* parser = argparse.ArgumentParser()  
  
 parser.add\_argument(**'--mode'**, choices=[**'train'**, **'validate'**, **'infer'**], default=**'infer'**)  
 parser.add\_argument(**'--decoder'**, choices=[**'bestpath'**, **'beamsearch'**, **'wordbeamsearch'**], default=**'bestpath'**)  
 parser.add\_argument(**'--batch\_size'**, help=**'Batch size.'**, type=int, default=100)  
 parser.add\_argument(**'--data\_dir'**, help=**'Directory containing IAM dataset.'**, type=Path, required=False)  
 parser.add\_argument(**'--fast'**, help=**'Load samples from LMDB.'**, action=**'store\_true'**)  
 parser.add\_argument(**'--line\_mode'**, help=**'Train to read text lines instead of single words.'**, action=**'store\_true'**)  
 parser.add\_argument(**'--img\_file'**, help=**'Image used for inference.'**, type=Path, default=**'../data/test08.png'**)  
 parser.add\_argument(**'--early\_stopping'**, help=**'Early stopping epochs.'**, type=int, default=25)  
 parser.add\_argument(**'--dump'**, help=**'Dump output of NN to CSV file(s).'**, action=**'store\_true'**)  
 args = parser.parse\_args()  
  
 *# set chosen CTC decoder* decoder\_mapping = {**'bestpath'**: DecoderType.BestPath,  
 **'beamsearch'**: DecoderType.BeamSearch,  
 **'wordbeamsearch'**: DecoderType.WordBeamSearch}  
 decoder\_type = decoder\_mapping[args.decoder]  
  
 *# train or validate on IAM dataset* if args.mode in [**'train'**, **'validate'**]:  
 *# load training data, create TF model* loader = DataLoaderIAM(args.data\_dir, args.batch\_size, fast=args.fast)  
 char\_list = loader.char\_list  
  
 *# when in line mode, take care to have a whitespace in the char list* if args.line\_mode and **' '** not in char\_list:  
 char\_list = [**' '**] + char\_list  
  
 *# save characters of model for inference mode* open(FilePaths.fn\_char\_list, **'w'**).write(**''**.join(char\_list))  
  
 *# save words contained in dataset into file* open(FilePaths.fn\_corpus, **'w'**).write(**' '**.join(loader.train\_words + loader.validation\_words))  
  
 *# execute training or validation* if args.mode == **'train'**:  
 model = Model(char\_list, decoder\_type)  
 train(model, loader, line\_mode=args.line\_mode, early\_stopping=args.early\_stopping)  
 elif args.mode == **'validate'**:  
 model = Model(char\_list, decoder\_type, must\_restore=True)  
 validate(model, loader, args.line\_mode)  
  
 *# infer text on test image* elif args.mode == **'infer'**:  
 model = Model(list(open(FilePaths.fn\_char\_list).read()), decoder\_type, must\_restore=True, dump=args.dump)  
 infer(model, args.img\_file)  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()