# Introduction

# Overview

Despite the abundance of technological writing tools, many people still choose to take their notes traditionally with pen and paper. Also, we come across various handwritten texts as in bank cheques, application forms, etc.

In our project, we will be converting this handwritten text into digital text by developing an efficient handwritten text recognition system for English characters based on ANN. We are developing a CRNN model.

This document will focus on the low-level design of the work we have done so far.

* 1. **Purpose**

Nowthat weare working on implementing our idea through code, this low-level document is going to focus on the various design aspects in more detail. Using the work, we have done so far, we will be able to discuss the new ideas, changes, and improvements brought upon by further research, practical constraints, etc. We will also be able to show code implementations for such ideas.

Other than the above mentioned, we will also be looking at the logical units of our project, such as modules, on a more magnified scale.

# IAM Handwriting Dataset

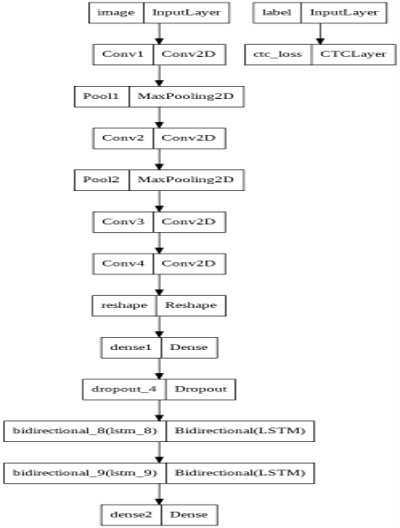
In the realm of artificial intelligence, advancements in handwriting recognition have opened doors to seamless interactions between humans and machines. The IAM Handwriting Dataset serves as a cornerstone for this progress, providing a rich source of unconstrained handwritten English text forms. With over 13,000 images penned by 657 individuals, the dataset encompasses 1,539 handwritten pages, meticulously segmented into 115,320 words. This remarkable collection offers a diverse range of writing styles, reflecting the nuances of individual handwriting.

The IAM Handwriting Dataset's unconstrained nature mimics real-world scenarios, where handwritten text often appears in various formats and conditions. This characteristic makes the dataset invaluable for training and evaluating handwriting recognition models, ensuring their robustness and generalizability. Researchers and developers worldwide have employed the IAM Handwriting Dataset to advance the state-of-the-art in handwriting recognition, achieving impressive results.

Furthermore, the IAM Handwriting Dataset has fostered significant contributions to writer identification and verification research. By analyzing the unique patterns and characteristics of individual handwriting, researchers have developed algorithms that can accurately identify and authenticate writers. This capability holds immense potential for applications such as forensic document analysis, signature verification, and personalized handwriting recognition systems.

In conclusion, the IAM Handwriting Dataset has emerged as an indispensable resource for handwriting recognition and writer identification research. Its comprehensive nature, diversity of writing styles, and unconstrained format have fueled advancements in these fields, paving the way for more intuitive and efficient human-computer interactions. As research in these areas continues to flourish, the IAM Handwriting Dataset will undoubtedly remain at the forefront, enabling further breakthroughs and shaping the future of handwriting analysis.

# Model architecture:



**Model Used: CRNN**

Description:

The proposed model is CRNN. A combination of CNN and RNN. The loss function used is CTC.

It comprises of 4 convolutional layers, 2 max pooling layers and 2 bi-directional layers. The activation function used with CNN is ReLu. At the dense layer, the activation function used is SoftMax. The optimizer used is Adam.

The proposed model outperforms the standard CNN.

Convolutional Neural Networks (CNN) have the ability or benefit to develop an internal representation of a two-dimensional image. This allows the network to detect the position and scale of letters in the input data images. As for Recurrent Neural Network (RNN), it works with sequence prediction problems.

The hybrid model has combining benefits of both CNN and RNN networks to be used to tackle the problem of extracting letters.

**3.1 .1 Algorithm and Pseudocode**

* + **Method 1**
* **Purpose:** To get paths and labels of sample data
* **Input:** samples
* **Output:** paths, corrected\_samples
* **Pseudo-code:**

FUNCTION get\_image\_paths\_and\_labels():

for every sample

splitting is done according to '-'

Image path is extracted and is appended to the paths list

Labels are extracted and is appended to corrected\_samples list

return paths, corrected\_samples

* **Method 2**
* **Purpose:** To clean labels for validation and test sets
* **Input:** labels
* **Output:** cleaned\_labels
* **Pseudo-code:**

FUNCTION clean\_labels(labels):

for every label

splitting according to space

the resulting label is appended to cleaned\_labels list

return cleaned\_labels

* **Method 3**
* **Purpose:** To resize the image without distortion
* **Input:** image, image\_size
* **Output:** image without distortion
* **Pseudo-code:**

FUNCTION distortion\_free\_resize(image, img\_size):

Resize the image according to img\_size

Check if pad\_height is odd

height\_top and height\_bottom are unequal

Else

height\_top and height\_bottom are equal

Check if pad\_width is odd

width\_left and width\_right are unequal

Else

width\_left and width\_right are equal

Resize the image with the resultant size

Transpose the image

Flip the image

return image

* **Method 4**
* **Purpose:** To pre-process image
* **Input:** image\_path, img\_size
* **Output:** image
* **Pseudo-code:**

FUNCTION preprocess\_image()

image is read from the path

image is decoded

call for distortion\_free\_resize function and stored in variable image

cast the image

return the resulting image

* **Method 5**
* **Purpose:** To vectorize an image
* **Input:** label
* **Output:** label
* **Pseudo-code:**

FUNCTION vectorize\_label()

label is converted to num

pad\_amount is calculated

label is padded with the corresponding pad\_amount

return label

* **Method 6**
* **Purpose:** Preparing dataset
* **Input:** labels, image\_paths
* **Output:** None
* **Pseudo-code:**

//prefetch one batch of data and make sure that there is always one ready

//saves some operations from being executed during each epoch

tensor\_slices operation is performed

* **Method 7**
* **Purpose:** CTC loss function
* **Input:** y\_pred, y\_true
* **Output:** y\_pred
* **Pseudo-code:**

FUNCTION call()

Initializes batch\_len, input\_length and label\_length

Defines a loss function

Return y\_pred

* **Method 8**
* **Purpose:** Building a model
* **Input:** None
* **Output:** returns CRNN model
* **Pseudo-code:**

FUNCTION build\_ model()

Initializes the model with Input layer

First convolution layer with ReLu activation function

First Maxpooling layer

Second convolution layer with ReLu activation function

Second Maxpooling layer

Third convolution layer with ReLu activation function

Fourth convolution layer with ReLu activation function

Dense layer with ReLu activation function

Dropout 0.2% of neurons

First Bidirectional layer with 0.25 dropout

Second Bidirectional layer with 0.25 dropout

Dense layer with softmax activation function

Adam optimizer

Compiling the model with Adam optimizer

Return model

* **Method 9**
* **Purpose:** To calculate edit distance
* **Input:** labels, predictions
* **Output:** edit\_distances
* **Pseudo-code:**

FUNCTION calculate\_edit\_distance()

Convert the labels to sparse tensors

Make predictions and convert them to sparse tensors

Compute individual edit distances and average them

Returns edit\_distances

* **Method 10**
* **Purpose:** Plotting the results
* **Input:** pred
* **Output:** plots for predicted and actual text for test images
* **Pseudo-code:**

FUNCTION decode\_batch\_predictions()

Input length is found

Iterating over results to get the text and the result is appended to output\_text list

For every batch

Predictions are made

Call for function prediction\_model.predict and is stored in variable preds

Call for decode\_batch\_predictions and is stored in variable pred\_textes

For an iterator in range 16

Image is flipped, transposed and plotted

* **Method 11**
* **Purpose:** To process image labels
* **Input:** image\_path, label
* **Output:** image, label
* **Pseudo-code:**

FUNCTION process\_images\_labels()

call for function presprocess\_image and store in variable image

call for function vectorize\_label and store in variable label

return image and label

* **Method 12**
* **Purpose:** call for edit distance
* **Input:** epoch
* **Output:** None
* **Pseudo-code:**

FUNCTION on\_epoch\_end()

For every validation\_image, prediction is made and edit distance is calculated

Call for calculate\_edit\_distance function

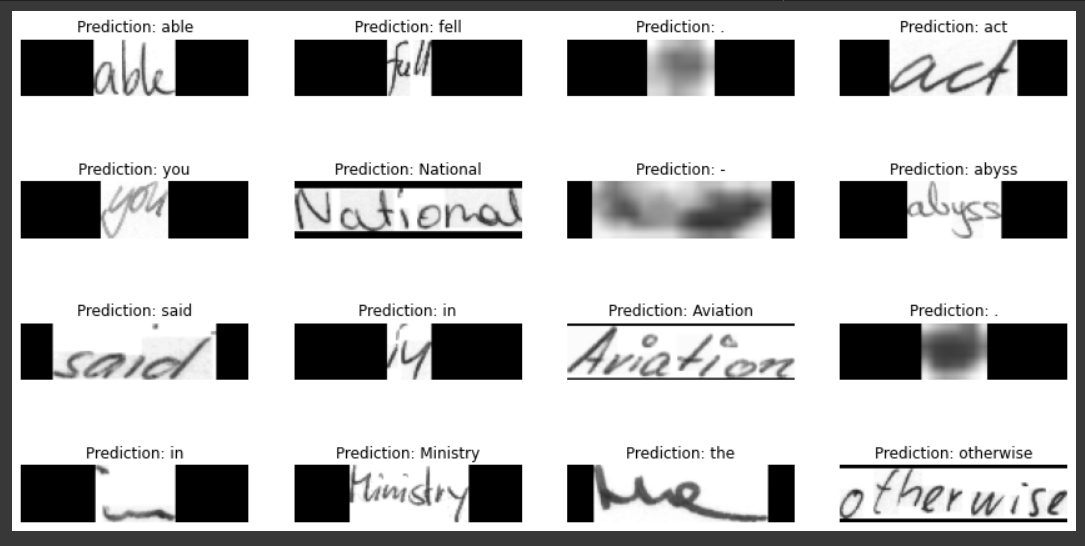
Print(edit distance in every iteration of training

**3.2 Implementation and Results**

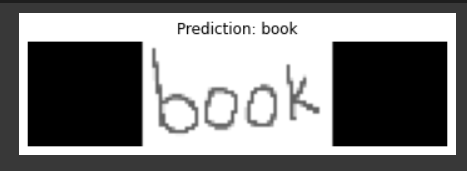
The model designed is CRNN. It is trained for 120 epochs. It is capable of predicting words that are not cursive. It can also predict non cursive sentences. But spaces are being neglected while predicting the sentence. So, we have to work on the same.

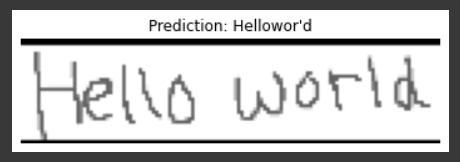
**Test set predictions:**

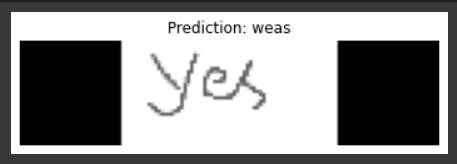
The plot comprises of the image and the prediction made by the model



**Real-time predictions:**







# Appendix A: Definitions, Acronyms and Abbreviations

1. CRNN- Convolutional recurrent neural network is a model is a combination of CNN and RNN that feeds every window frame by frame into a recurrent layer and use the outputs and hidden states of the recurrent units in each frame for extracting features from the window.
2. CNN- Convolutional neural network is a class of neural networks that specializes in processing data that has grid like topology. Here, it is used to develop an internal representation of a two-dimensional image.
3. RNN- Recurrent neural network is used for saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer. Here, it works with sequence prediction problems.
4. CTC loss- This loss function is designed for tasks where there is a need for alignment between sequences and where the alignment is difficult.
5. Edit distance- Minimum number of character insertions, deletions, and substitutions required to change one string to another.

# Appendix B: References

[1] R. Achkar, K. Ghayad, R. Haidar, S. Saleh and R. Al Hajj, "Medical Handwritten Prescription Recognition Using CRNN," 2019 International Conference on Computer, Information and Telecommunication Systems (CITS), 2019, pp. 1-5, doi: 10.1109/CITS.2019.8862004

[2] S. Butala, A. Lad, H. Chheda, M. Bhat and A. Nimkar, "Natural Language Parser for Physician’s Handwritten Prescription," 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), 2020, pp. 1-7, doi: 10.1109/ic-ETITE47903.2020.325. ​

[3] E. Hassan, H. Tarek, M. Hazem, S. Bahnacy, L. Shaheen and W. H. Elashmwai, "Medical Prescription Recognition using Machine Learning," 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC), 2021, pp. 0973-0979, doi: 10.1109/CCWC51732.2021.9376141. ​