

**HAND WRITTEN TEXT RECOGNITION**

(USING DEEP LEARNING)

**SUBMITTED TO :**

Dr. Ravi Kumar Jatoth.

Professor , Department of Electronics & communication Engineering

**SUBMITTED BY:**

Sarvepalli. Mahathi – 21ECB0B53

Vattam. Sai Sharanya – 21ECB0B63

1. INTRODUCTION

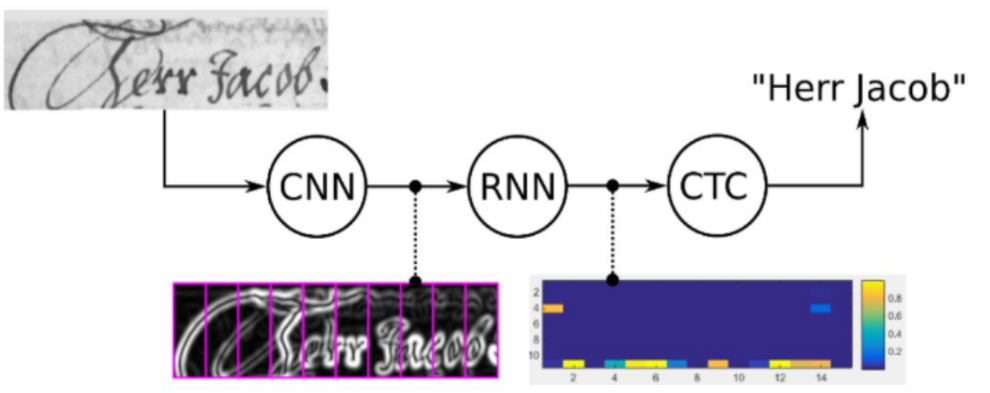
Currently majority of the scripts are handwritten due to the ease of using pen-tip in place of keyboard. As a result, the manual processing of the handwritten reports/letters/scripts takes a lot of time and sometimes even leads to errors. Hence text extraction from handwritten content is essential for faster and efficient evaluation/processing of information.

With the world gravitating towards absolute digitization, there is a high demand for a Handwritten Text Recognition system. However, the challenge of implementing this system lies in the fact that handwritten words vary in characteristics such as slant and rounded letters, diacritic dots, crossbars, and humped letters. A good handwriting recognition system must accurately identify the distorted characters and hence find the most plausible words.

Our project ‘Handwritten text recognition using deep learning’ aims at converting text written with hand into digital text by using Convolution Neural Network (CNN), Long Short-Term Memory (LSTM) which uses the architecture of Recurrent Neural Network (RNN) and Connectionist Temporal Classification (CTC). The workflow of this system would be to extract the words from the image of a text using OpenCV and feed the words into a neural network model for the recognition of the words.

In this project, the IAM Dataset (Source) containing more than 100,000 images of unconstrained handwritten text is used for training, validating, and testing the system to achieve better efficiency. The Neural Network is trained using this dataset to eventually recognize handwriting. The recurrent neural network (RNN) can process larger input though it has lesser computational power. On the other hand, the Convolutional neural network (CNN) needs larger data for training. In this handwritten text recognition system, an adaptive method is proposed for offline Handwritten text recognition by integrating both. Here the dataset is trained consecutively with CNN and RNN. Connectionist temporal classification (CTC) network is fitted along with RNN through training to model the probability of a label.

The objectives of the project are to analyse the complexities involved in recognizing handwritten text by using both CNN and RNN and to calculate the loss using the Connectionist temporal classification (CTC) network and to test the performance evaluation of the CNN-RNN method.



Outline of the neural network used

An improvement in the man-to machine interactions in many applications occurs by utilizing the advanced automation processes involved in the handwritten text recognition systems.

1. METHODOLOGY

The process of text recognition is achieved using a neural network consisting of CNN layers, LSTM layers and CTC layer. The input of the neural network is a pre processed image of handwritten word and the output is the predicted text.

The implementation of this project is done in two steps namely model development and model training. The trained model works as the text recognition system which takes the pictures of discrete words and gives the predicted text as the output.

Each of these steps is discussed below in detail.

1. Building the neural network model.
2. Training the model

**BUILDING THE NEURAL NETWORK MODEL**

The neural network layer is a combination of CNN and RNN, consisting of two convolution layers, one dense layer and two bidirectional (RNN) layers.

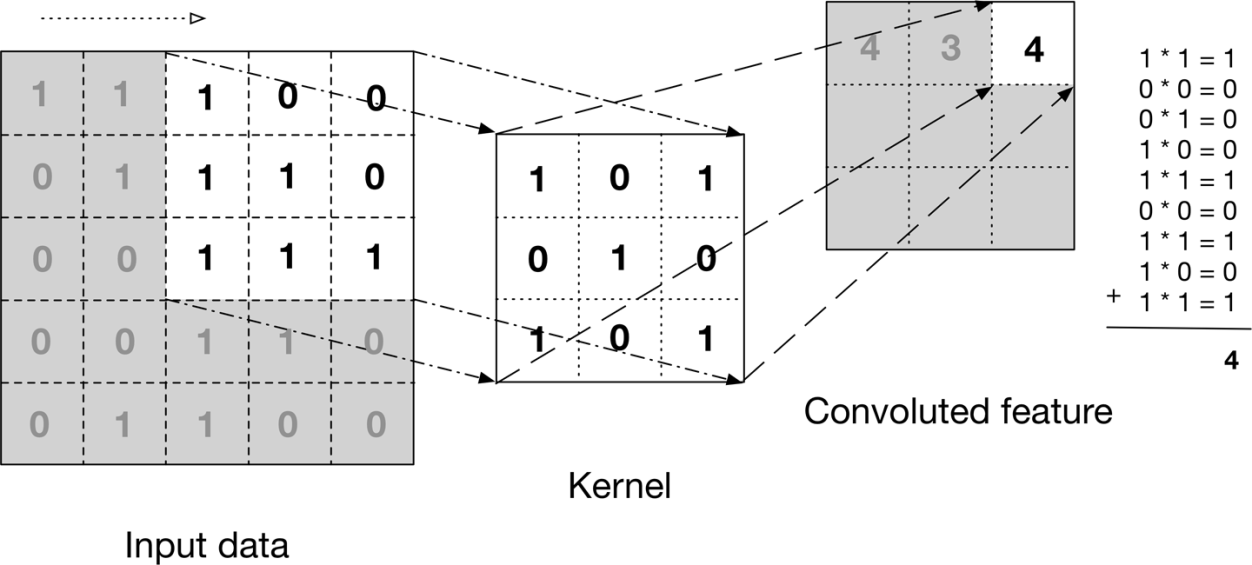
The library ‘TensorFlow’ and the ‘Keras’ API have been used for performing mathematical and computational operations in the neural network used.

**Convolutional neural network:**

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

The basic function of CNN is that to reduce the images into a form that is easier to process, without losing features that are critical for getting a good prediction***.***

**Convolution Layer :**



The convolution operation is performed over the image to extract features.The above image shows what a convolution is. We take a filter/kernel(3×3 matrix) and apply it to the input image to get the convolved feature. This convolved feature is passed on to the next layer.

**Pooling Layer :**

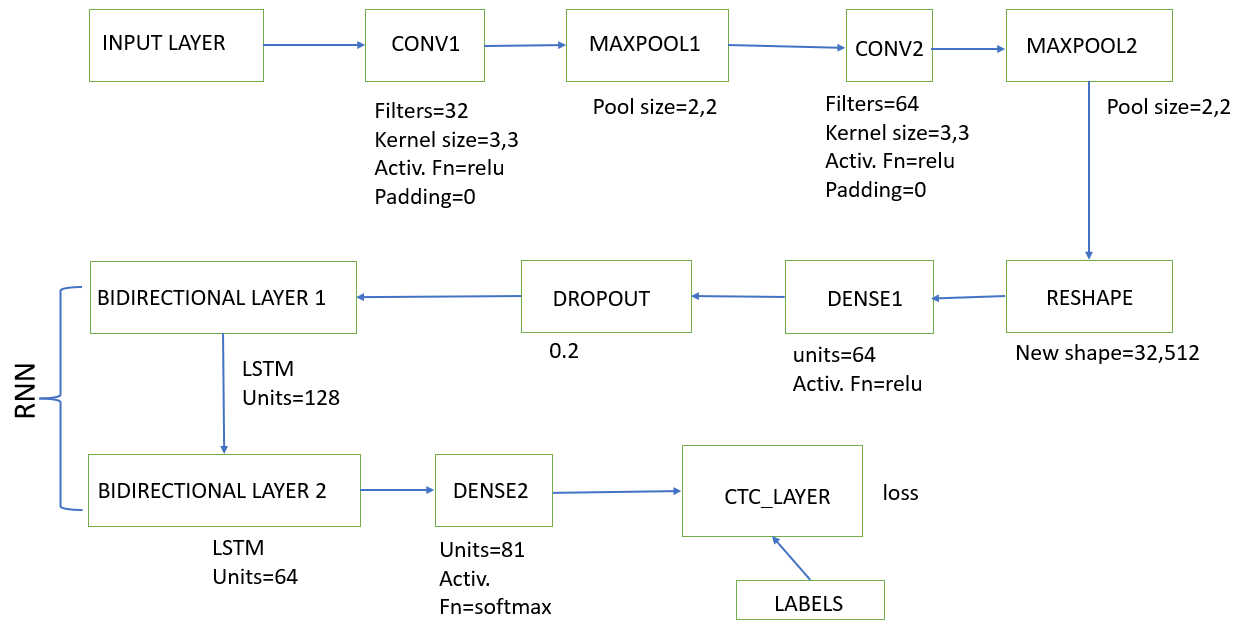
Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data by reducing the dimensions. There are two types of pooling average pooling and max pooling. Max pooling is generally used.

So what we do in Max Pooling is we find the maximum value of a pixel from a portion of the image covered by the kernel. Max Pooling also performs as a**“**Noise Suppressant”. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction.

On the other hand, “Average Pooling**”**returns the average of all the values from the portion of the image covered by the Kernel. Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that Max Pooling performs a lot better than Average Pooling.



The following flow-chart, depicts in brief the layers present in the model.



Each image in grey scale is sized to 128x32 and given to the input layer of the neural network. The image inputs are then convolved with 32 kernels of size 3x3 and 32 features of size 128x32 are extracted, the weights of the kernels are initialised using the he\_normal function which draws samples from a truncated normal distribution centred on 0 with standard deviation (after truncation) given by

stddev = sqrt(2 / fan\_in)

where fan\_in is the number of input units in the weight tensor.

This is followed by a maxpooling layer of size 2x2, this reduces the size of each feature to 64x16.

The features then pass through a convolution layer of size 3x3 with 64 filters and maxpooling layer of size 2x2. The features (of size 32x8) are then reshaped to size 32x512 and given to the dense layer. A dense layer is a deeply connected neural network, which takes input from all the nodes of the previous network. It uses the formula given below to compute the output:

output = activation (dot(input, kernel) + bias)

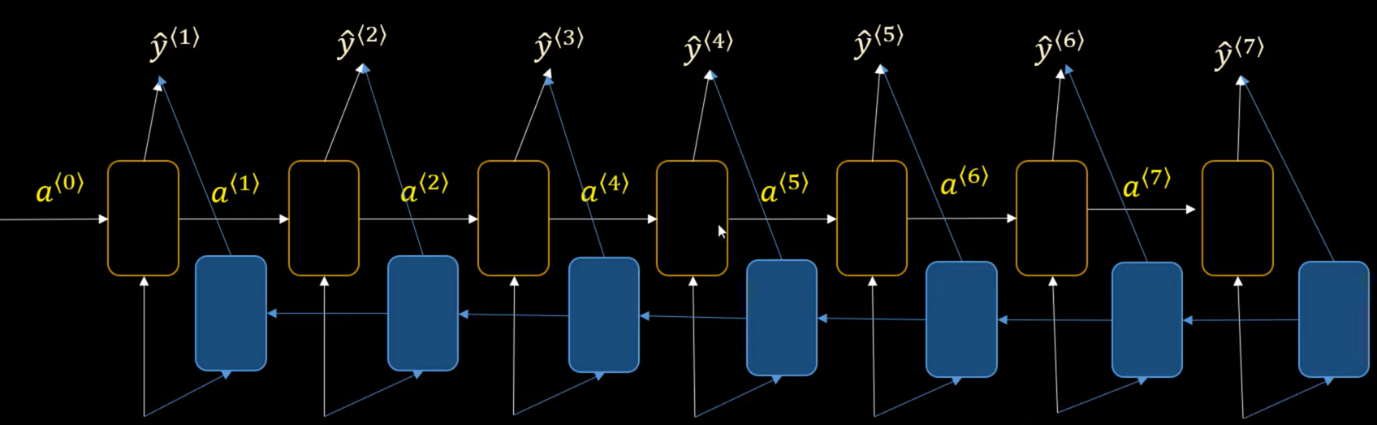
Where kernel is the weight matrix created within and by the layer.

The dense layer used here has 512 input units and 64 output units and the activation function used is ‘Relu’. Finally, a dropout layer is applied to avoid over-fitting of the model.

The network used up till now is the convolutional neural network (CNN), for further processing Recurrent Neural Network (RNN) is required.

* A **recurrent neural network** (**RNN**) is a class of [artificial neural networks](https://en.wikipedia.org/wiki/Artificial_neural_network) where connections between nodes can create a cycle, allowing output from some nodes to affect subsequent input to the same nodes.
* This allows it to exhibit temporal dynamic behavior.
* Derived from [feedforward neural networks](https://en.wikipedia.org/wiki/Feedforward_neural_networks), RNNs can use their internal state (memory) to process variable length sequences of inputs.

In RNN, the bidirectional layers have been used in the code. In a bidirectional RNN layer, the sequence is encoded in both backward and forward directions and the result is the concatenation of both the forward and the backward LSTMs in each step. And here the two hidden layers, in opposite directions, are connected to the same output.



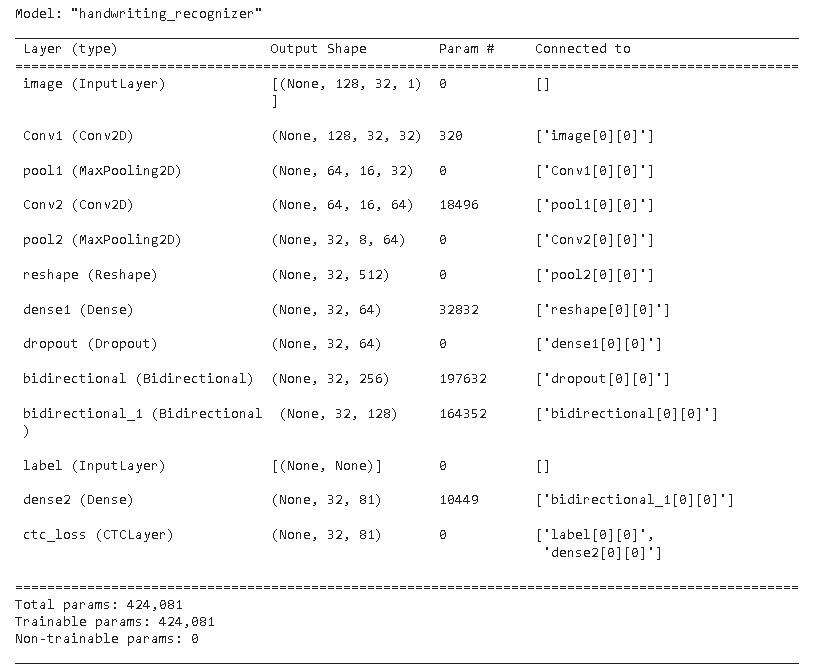
Bidirectional LSTM

In the project, two bidirectional layers have been used, one with 256 output units 128 neural layers and the other with 128 output units and 64 neural layers, also a dropout of 0.25 is given to each layer to prevent over fitting.

The LSTM (long short term memory) is the type of recurrent neural network used in the code. LSTMs are used for processing not just single data points but also entire sequences of data.

Finally, the output of the second bidirectional layer is given to a dense layer with 81 outputs. Each of the output of the dense layer corresponds to one character in literature, and with softmax as the activation function, the dense layer gives a high output at the predicted character and a low output at all other characters.

The summary of the neural network model is given below:



Summary of the model

**TRAINING THE MODEL**

The training of the model involves the following algorithmic steps:

* Preparing ground truth tables.
* Building character vocabulary.
* Pre processing images.
* Pre processing image labels.
* Creating data input pipeline for loading data into the model.
* Building the model.
* Defining functions for calculating metrics.
* Training the model and saving it.
* Predicting output using the trained model.

Each of the step is discussed in detail below.

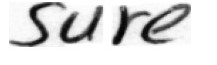
1. **Preparing ground truth tables and Building character vocabulary**

This step extracts the ‘set’ of characters from the labels of images. And Defines function for assigning a number to each character and function for converting a number back into its corresponding character. The neural network model can only process numbers and hence this step is essential.

1. **Pre-processing images**

All the images in the dataset are pre-processed to take a fixed size of 128 x 32 pixels. The function distortion\_free\_resize() has been used for this purpose. Also, padding is added to some of images for obtaining uniform size, the amount of padding added is based on the original size of the image. The resized image is then transposed as this gives more accuracy.

Finally, the pixel values are type casted into the data type of float32, and each pixel value is divided by 255.0 so as to normalize the images.

Original image Image after pre-processing

1. **Pre-processing image labels**

As a part of data pre-processing, each of the image labels is converted into a vector of fixed size using the vectorize\_labels() function. Here, each of the characters in a label is converted to a specific number and additional blank spaces (if needed) are added to each vector to ensure that all vectors have the same length. The length of each vector is equal to 21 which is the length of the largest label.

1. **Creating data input pipeline**

A data input pipeline is used to help automate machine learning workflows. They operate by enabling a sequence of data to be transformed and correlated together in a model that can be tested and evaluated to achieve an outcome. In the project we us the pipeline to correlate the images with their labels for easy training of the model.

Here the training, testing and validation data are converted into a tensorflow dataset. Wherein they are converted into objects of the data type tf.data.Dataset. Only in the training dataset the images are mapped with their respective labels, the train and validation datasets contain just the batches of images. The datasets are batched into batches of 64.

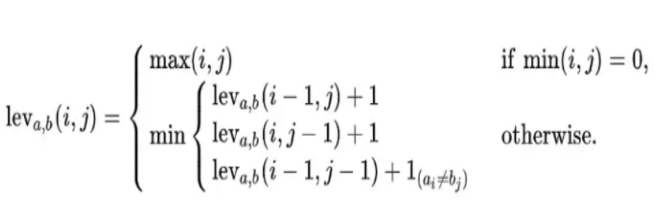
1. **Building the model (this has been discussed in (i) in II).**
2. **Calculating metrices**

The loss function is used to optimize your model. This is the function that will get minimized by the optimizer. A metric is used to judge the performance of your model. This is only for you to look at and has nothing to do with the optimization process.

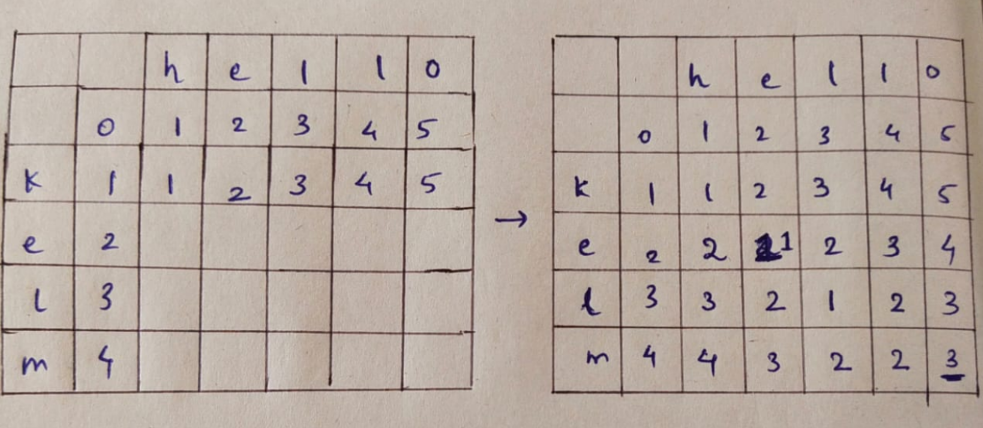
The metric we are using is mean\_edit\_distance. It is the most widely used metric for hand written text recognition, it is a string metric. This metric is found by calculating minimum number of operations required to transform one string to other. The edit\_distance api of tensorflow will calculate the ‘lavenshtein’ distance between the two strings.

The output of the metrics edit distance is Levenshtein distance between predicted values and output labels. Levenshtein distance operations are the removal, insertion, or substitution of a character in the string. Being the most common metric, the term Levenshtein distance is often used interchangeably with edit distance.

**Levenshtein Distance**:



Where, a is string 1 , b is string 2 and i,j are positions in string 1,2 respectively.

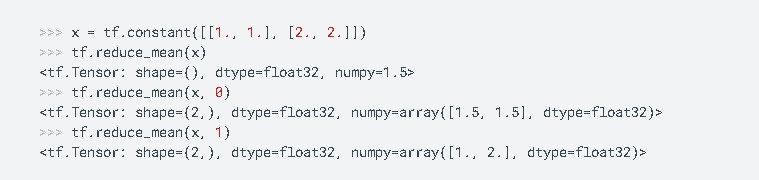


In the example above, by using Levenshtein distance formula it has been found that three transformations (deletion/modification/insertion) must be made to convert ‘kelm’ to ‘hello’ and vice-versa.

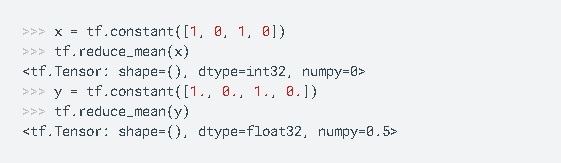
The function ‘tf.reduce\_mean’ reduces input\_tensor along the dimensions given in axis by computing the mean of elements across the dimensions in axis. Unless ‘keepdims’ is true, the rank of the tensor is reduced by 1 for each of the entries in axis, which must be unique. If ‘keepdims’ is true, the reduced dimensions are retained with length 1.

If ‘axis’ is None, all dimensions are reduced, and a tensor with a single element is returned.

For example:



The metric mean\_edit\_distance is equivalent to np.mean. But with the difference that np.mean has a dtype parameter that could be used to specify the output type. By default this is dtype=float64. On the other hand, tf.reduce\_mean has an aggressive type inference from input\_tensor, for example:



**CTC layer:**

Last layer of our model is CTC layer.Connectionist Temporal Classification (CTC) is a type of Neural Network output helpful in tackling sequence problems like handwriting and speech recognition where the timing varies. Using CTC ensures that one does not need an aligned dataset, which makes the training process more straightforward.

CRNN output a character-score for each time-step, which is represented by a matrix.

The matrix is used to do the following tasks:

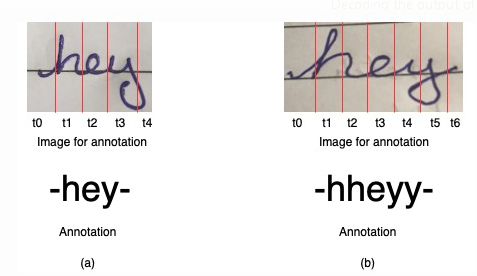
Training the Neural Network, i.e., calculating the loss

Decoding the output of the Neural Network

CTC operation helps in achieving both tasks.

**Problems solved by CTC:**

The problem arises when character takes up more than one time-step (as shown in the figure below) It will result in duplication of the characters.



In figure (a) single character takes single time step but in figure (b) few characters take more than one time step.

**CTC comes to the rescue:**

CTC is formulated in such a way, that it only requires the text that occurs in the image. We can ignore both the width and position of the characters in an image.

There is no need for post-processing the output of the CTC operation. Using decoding techniques, result is obtained from the network.

**CTC works on the following three major concepts:**

Encoding the text

Loss calculation

Decoding

**Encoding the text :**The issue arise with methods not using CTC when the character takes more than one time-step in the image. Non-CTC methods would fail here and give duplicate characters.

To solve this issue, CTC merges all the repeating characters into a single character. For example, if the word in the image is ‘hey’ where ‘h’ takes three time-steps, ‘e’ and ‘y’ take one time-step each. Then the output from the network using CTC will be ‘hhhey’, which as per our encoding scheme, gets collapsed to ‘hey’.

For handling those cases where the words contain repeating characters CTC introduces a pseudo-character called blank denoted as “-“ in the following .While encoding the text, if a character repeats, then a blank is placed between the characters in the output text. Let’s consider the word ‘meet’, possible encodings for it will be, ‘mm-ee-ee-t’, ‘mmm-e-e-ttt’, wrong encoding will be ‘mm-eee-tt’, as it’ll result in ‘met’ when decoded. The CRNN is trained to output the encoded text.

**Loss calculation:**

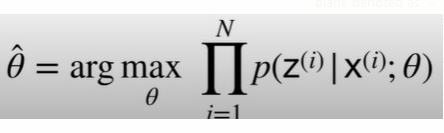
At each time step the network actually outputs a probability distribution over all the possible labels + the blank symbol. If we randomly choose a label independently from each of these distributions, negative logarithm of the probability that we will get an output sequence that collapses into the ground truth (correct output) sequence is CTC loss.

**Formal description of CTC loss**:

Let L be the set of labels and L’ be the set of labels with the “blank” label.

For a sequence of length T , we denote the set of Possible paths L’T = A.

Given a sequence of inputs x and labelling z , where |z| <= |x| , try to maximize the probability of the labelling given the sequence (maximum likelihood estimate).

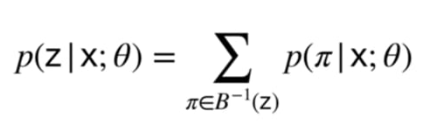


Define the many-to-one map B : L’T -> L<\_T that maps length T label sequence of alphabet L’ to their labelling equivalent in L, while removing all blanks and repeated labels. In effect, B performs the “collapsing” operation discussed previously.

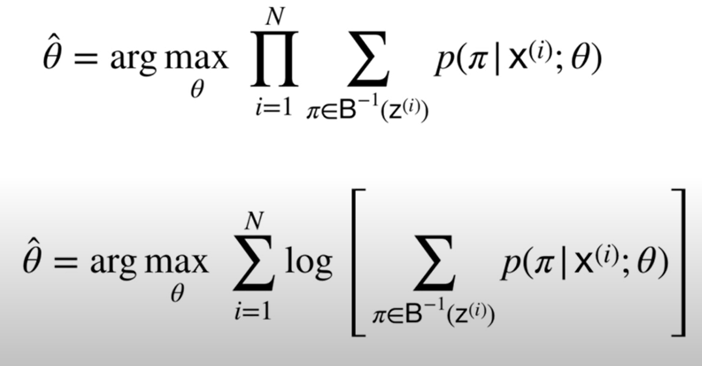


Define B-1 to map a label sequence z to the set of all possible label sequences (paths in A) that collapse to z. So {B(x) | x belongs to B-1(y) } = y.

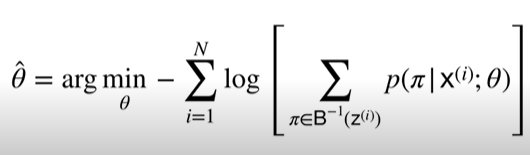
Therefore, we can consider the likelihood of a given labelling z as the sum of the probabilities of all the paths that can collapse to z.



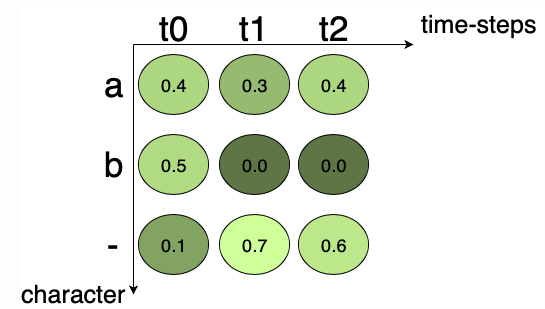
Our objective is now more clear and we can plug in our formulation of the likelihood of a labelling .The objective of the arg min is the CTC loss.



Changing the problem from maximization to minimization



A simple example to the above stated method for the loss calculation given below



Here, if the ground truth is “a”, all the possible paths for “a” are “aaa”, “a–”, “a-“, “aa-“, “-aa”, “–a”. Summing up the score of the individual path we get, 0.048 + 0.168 + 0.018 + 0.072 + 0.012 + 0.028 = 0.346. 0.346 is the probability of the ground truth occurring, it is not the loss. The loss is the negative logarithm of probability, it can be calculated easily. This loss can be back-propagated and the network can be trained.

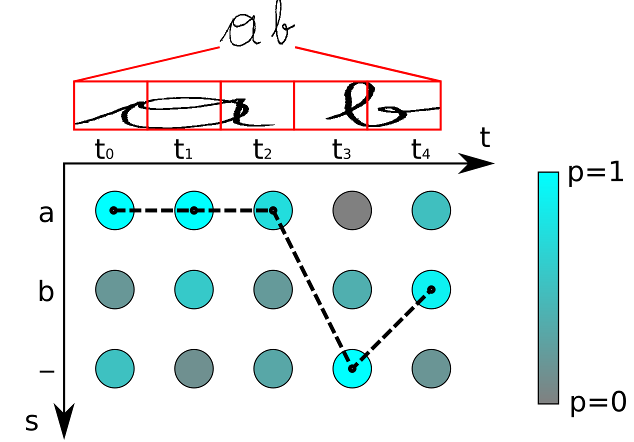
**Decoding:**

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

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

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

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

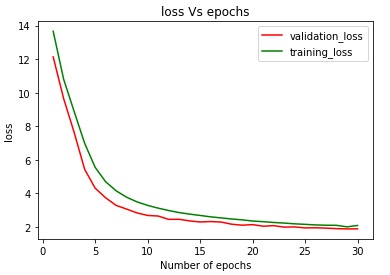


There are more advanced decoders such as beam-search decoding, prefix-search decoding or token passing, which also use information about language structure to improve the results.

1. RESULTS

**LOSS V/S EPOCHS**

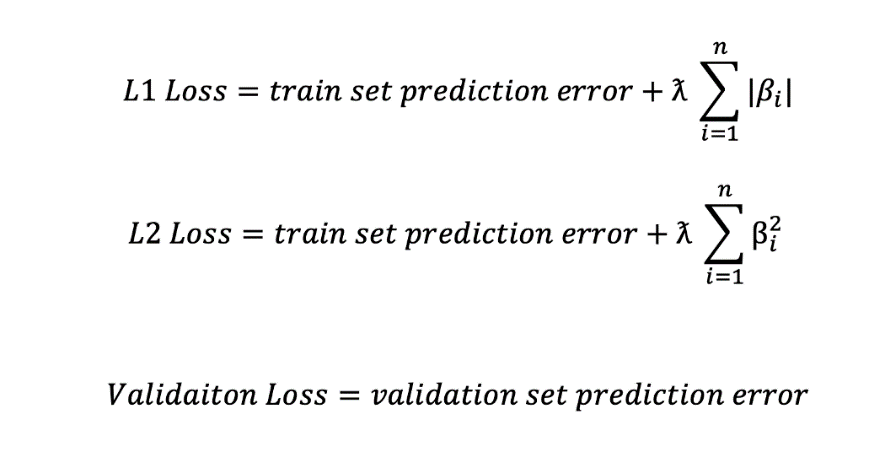
The plot of training loss and validation loss as a function of the number of epochs is as below:



Here, it is observed that the training loss is greater than the validation loss. The reasons for this are 2, which are as follows:

**Reason 1: L1 or L2 Regularization**

Whether L1 or L2 regularization is used, the error function is effectively getting inflated when the model weights are added to it:



The regularization terms are only applied while training the model on the training set, inflating the training loss. During validation and testing, your loss function only comprises prediction error, resulting in a generally lower loss than the training set.

The gap between validation and train loss shrinks after each epoch. This is because as the network learns the data, it also shrinks the regularization loss (model weights), leading to a minor difference between validation and train loss.

**Reason 2: Dropout**

Dropout penalizes model variance by randomly freezing neurons in a layer during model training. Like L1 and L2 regularization, dropout is only applicable during the training process and affects training loss, leading to cases where validation loss is lower than training loss.

**ACCURACY**

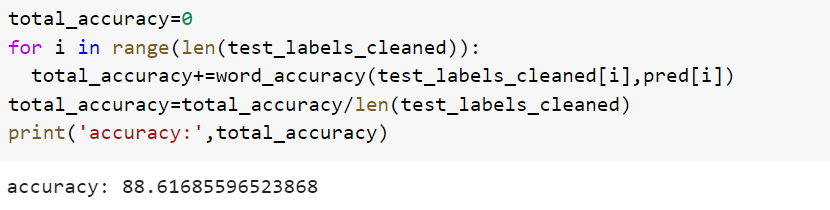
Another parameter which evaluates the performance of this model is the average accuracy with which each of the words has been identified.

A function has been used for this purpose which computes the percentage of the word which has been correctly identified, by checking the letters at each of the position in the predicted text.

For instance, let the original text be ‘Sketches’, if the predicted text is ‘Skekchea’ i.e., 6 out of eight letters have been rightly identified and are present in the same position as in the original text, hence the accuracy of the predicted word is equal to 75 percent. Similarly, if the original text is ‘late’ and the predicted text is ‘bte’, then the accuracy of prediction is 50 percent as 2 out of 4 words are rightly identified and are present at the right position with respect to the original text.

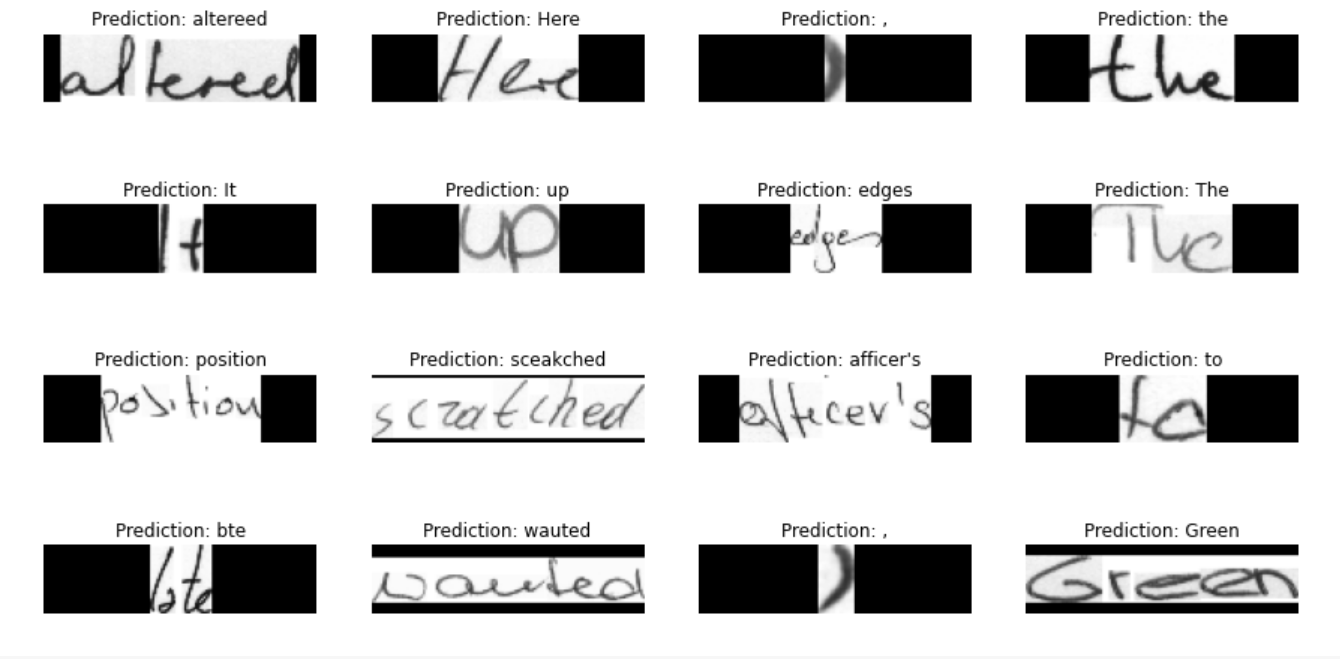
Hence, the total accuracy of the model is equal to the average accuracy of predictions of all the handwritten words.

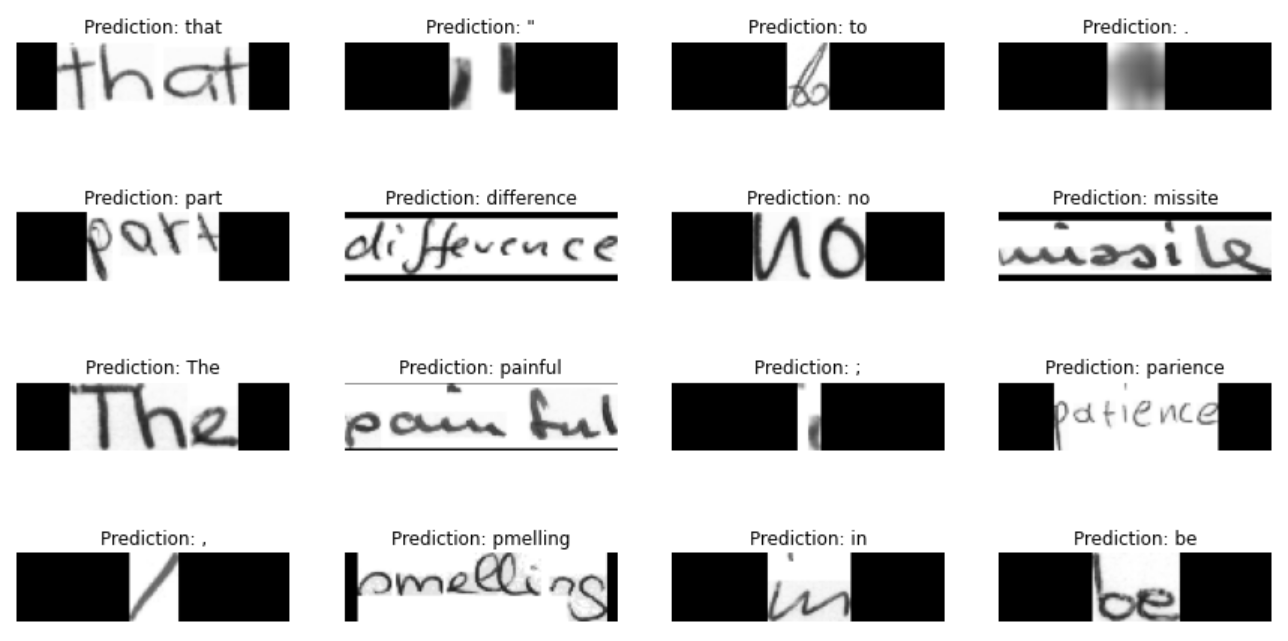
By considering all the handwritten word images in the test dataset (i.e., 4823 samples) the **accuracy of the model is** **88.62%.**



Accuracy of the neural network

The below are some of the samples whose text has been extracted by the model.



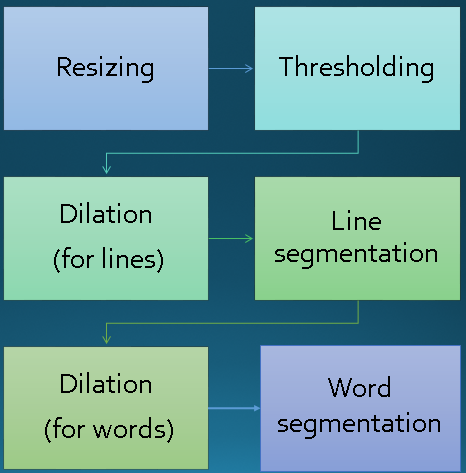


1. FUTURE SCOPE

The current project performs the task of taking the image of a handwritten word as input and extracts the text present in it.

For this project to be practically feasible, it must have the capacity to take sentences or paragraphs of handwritten text and extract the content present. We aim to achieve this by using the python library ‘OpenCV’. It is a library of programming functions mainly aimed at real-time computer vision.

Line and word segmentation using ‘OpenCV’ to extract words from an image containing hand written text.



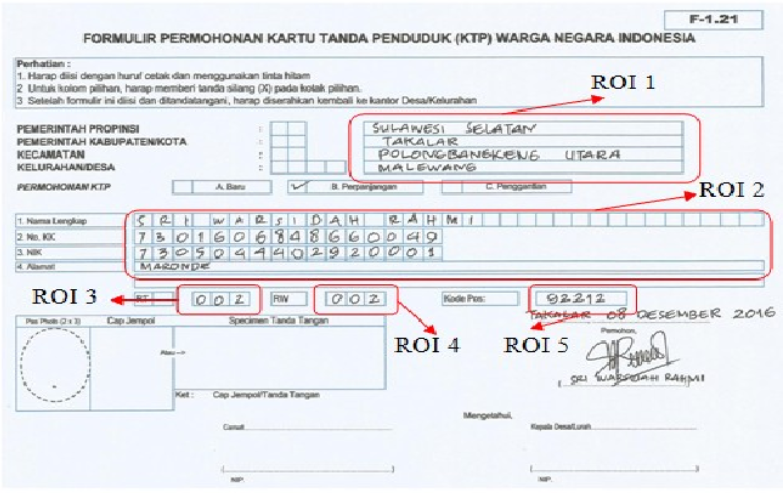
1. APPLICATIONS

Handwriting recognition helps to transform the writings in the papers to a text document format which can also be said as readable electronic format. Handwritten Text Recognition (HTR) technology is now a mature machine learning tool, becoming integrated in the processes of digitisation. It helps in reducing manual effort and increases the efficiency of performing mundane and cumbersome task of manually analysing, processing and evaluating handwritten text documents.

One of the major applications of this project is that, it can be added as a feature to apps where the input is text file or image of text file. If this feature is added, then the app can take input of hand written document and extract text from it, this saves the effort the user needs to put in to retype the entire text.

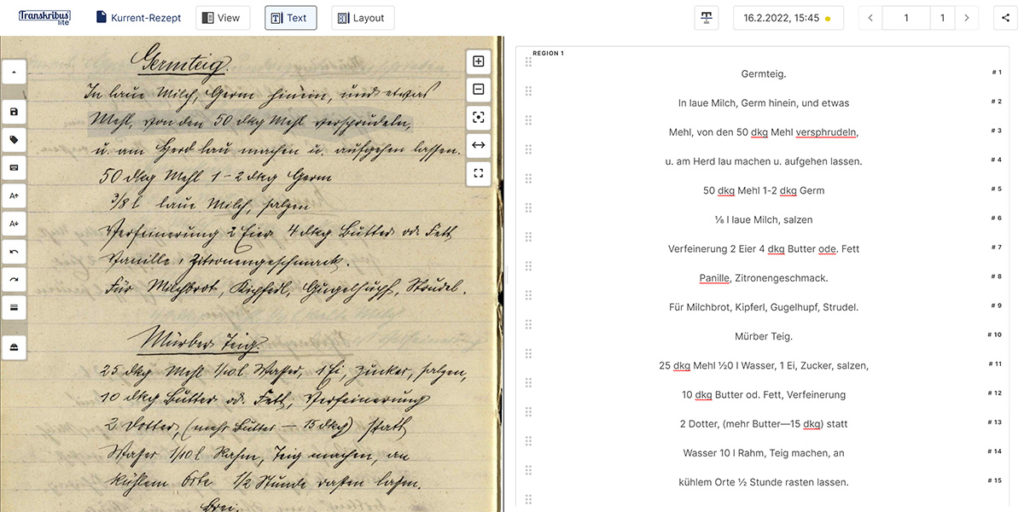
Eg : This feature can be added to the ‘Doubtnut App’ (an EdTech app which aims at providing solutions to the questions asked by the students) so that the students can directly upload a picture of their question into the app rather than typing again the whole text.

Another application being that handwritten text recognition helps in classifying files (like resume) according to the required criteria by extracting the information automatically without the supervision of humans. The text in the files or scripts or applications can be extracted, converted into digital text and can be directly processed by the computer to obtain the necessary information.



It can also be used to read the hand written information on posts in post office and arrange the letters in the desired order.

 Furthermore, OCR plays an important role for digital libraries, allowing the entry of image textual information into computers by digitization, image restoration, and recognition methods.



**VI .CONCLUSION:**

In this project we proposed a method for the identification of hand written text. Our model consists of CNN and RNN layers consecutively and end point layer is CTC layer. The images in the dataset are preprocessed and fed to the CNN layer the outputs of CNN are given to RNN and to decode the RNN layer outputs CTC layer is used. The results obtained shows that the hybrid model of CNN and RNN works well for text recognition.

**VII . REFERENCES:**

1.G.R. Hemanth, M. Jayasree, S. Keerthi Venii, P. Akshaya, and R. Saranya “CNN-RNN BASED HANDWRITTEN TEXT RECOGNITION “-<http://ictactjournals.in/ArticleDetails.aspx?id=6603>

2. Rohan Vaidya, Darshan Trivedi , Sagar Satra ,Prof . Mrunalini Pimpale -“Handwritten Character Recognition Using Deep-Learning” - [https://ieeexplore.ieee.org/document/8473291/authors#authors](AIML_report15.docx)

3. Alex Graves, Santiago Fernandez, Faustino Gomez, Jurgen Schmidhuber - <https://www.cs.toronto.edu/~graves/icml_2006.pdf>

**Data Set :**

We have used the IAM\_Words dataset which comprises of 1,15,320 handwritten words and their corresponding labels**.**

IAM Dataset - <https://www.kaggle.com/datasets/nibinv23/iam-handwriting-word-database>