IARJSET

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Vol. 8, Issue 6, June 2021

DOI: 10.17148/IARJSET.2021.86148

Handwritten Text Recognition using Deep Learning (CNN & RNN)

Rohini G. Khalkar[1], Adarsh Singh Dikhit[2], Anirudh Goel[3], Manisha Gupta[4]

Assistant professor, Department of Computer Engineering, Bharati Vidyapeeth (Deemed to be University)

College of Engineering, Pune, India-411043[1]

Department of Computer Engineering, Bharati Vidyapeeth (Deemed to be University)

College of Engineering, Pune, India-411043[2,3,4]

Abstract: Handwriting Detection is a technique or ability of a computer to receive and interpret intelligible handwritten input from source such as paper documents, touch screen, photographs etc. Handwritten Text recognition is one of area pattern recognition. The purpose of pattern recognition is to categorize or classification data or object of one of the classes or categories. Traditional systems of handwriting recognition have relied on handcrafted features and a large amount of prior knowledge. Training an Optical character recognition (OCR) system based on these prerequisites is a challenging task. Research in the handwriting recognition field is focused on deep learning techniques and has achieved breakthrough performance in the last few years. Still, the rapid growth in the amount of handwritten data and the availability of massive processing power demands improvement in recognition accuracy and deserves further investigation. Convolutional neural networks (CNNs) are very effective in perceiving the structure of handwritten characters/words in ways that help in automatic extraction of distinct features and make CNN the most suitable approach for solving handwriting recognition problems. This system will be applied to detect the writings of different format. The development of handwriting is more sophisticated, which is found various kinds of handwritten character such as digit, numeral, cursive script, symbols, and scripts including English and other languages.

Keywords: CNN, RNN, CTC, Tensor Flow, OCR, SoftMax

INTRODUCTION

An OCR system depends mainly on the extraction of features and discrimination/classification of these features (based on patterns). Handwritten OCR have received increasing attention as a subfield of OCR.

Handwritten Text Recognition is a technology that is much needed in this world as of today. Before proper implementation of this technology, we have relied on writing texts with our own hands which can result in errors. It is difficult to store and access physical data with efficiency. Manual labor is required to maintain proper organization of the data. Throughout history, there has been severe loss of data because of the traditional method of storing data. Modern day technology is letting people store the data over machines, where the storage, organization and accessing of data is relatively easier.

Adopting the use of Handwritten Text Recognition software, it is easier to store and access data that was traditionally stored. Furthermore, it provides more security to the data. One such example of Handwritten text Recognition software is the Google Lens. The aim of our project is to make an application that can recognize the handwriting using concepts of deep learning. We are thinking by approaching our problem using CNN as they provide better accuracy over such tasks.

OBJECTIVE

Handwritten Text recognition is one of the areas of pattern recognition. The purpose of pattern recognition is to categorize or classify data or object of one of the classes or categories. Handwriting recognition is defined as the task of transforming a language represented in its spatial form of graphical marks into its symbolic representation. Each script has a set of icons, which are known as characters or letters, which have certain basic shapes.

The goal of handwriting is to identify input characters or image correctly then analyzed to many automated process systems. This system will be applied to detect the writings of different format. The development of handwriting is more sophisticated, which is found various kinds of handwritten character such as digit, numeral, cursive script, symbols, and

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
Vol. 8, Issue 6, June 2021
DOI: 10.17148/IARJSET.2021.86148

scripts including English and other languages. The automatic recognition of handwritten text can be extremely useful in many applications where it is necessary to process large volumes of handwritten data, such as recognition of addresses and postcodes on envelopes, interpretation of amounts on bank checks, document analysis, and verification of signatures. Therefore, computer is needed to be able to read document or data for ease of document processing. Artificial Intelligence is a branch of computer science that endeavors to replicate or simulate human intelligence in a machine, so machines can perform tasks that typically require human intelligence. Some programmable functions of AI systems include planning, learning, reasoning, problem solving, and decision making. Artificial intelligence systems are powered by algorithms, using techniques such as machine learning, deep learning and rules. Machine learning algorithms feed computer data to AI systems, using statistical techniques to enable AI systems to learn. Through machine learning, AI systems get progressively better at tasks, without having to be specifically programmed to do so.

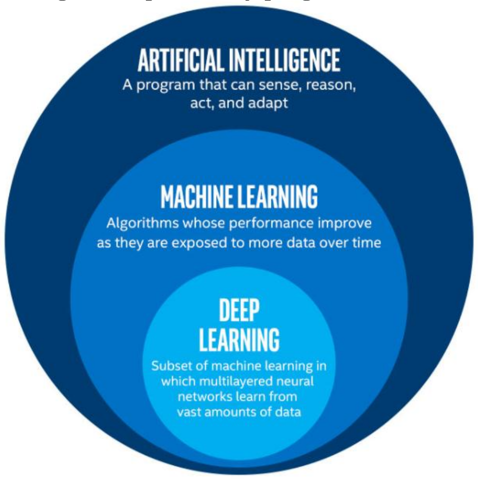

ARTIFICIAL INTELLIGENCE
A program that can sense, reason, act, and adapt
MACHINE LEARNING
Algorithms whose performance improve as they are exposed to more data over time
DEEP LEARNING
Subset of machine learning in which multilayered neural networks learn from vast amounts of data

Figure 1.1: Venn Diagram for AI, ML, NLP & DL.

Deep Learning is a subset of Machine Learning, which on the other hand is a subset of Artificial Intelligence. Deep learning algorithms attempt to draw similar conclusions as humans would by continually analyzing data with a given logical structure. To achieve this, deep learning uses a multi-layered structure of algorithms called neural networks.

The design of the neural network is based on the structure of the human brain. Just as we use our brains to identify patterns and classify different types of information, neural networks can be taught to perform the same tasks on data.

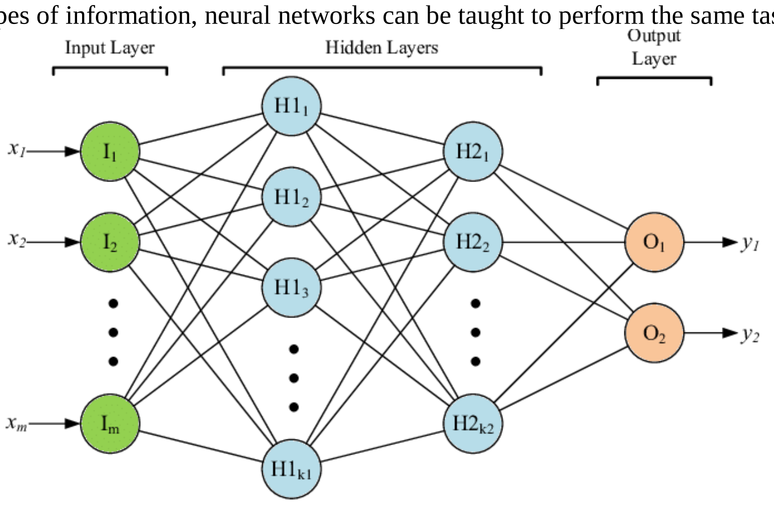


Figure 1.2: DL – Neural Network Architecture.

Natural language processing (NLP) refers to the branch of computer science and more specifically, the branch of artificial intelligence or AI concerned with giving computers the ability to understand text and spoken words in much the same way human beings can. NLP combines computational linguistics rule-based modelling of human language with statistical, machine learning, and deep learning models. Together, these technologies enable computers to process human language in the form of text or voice data and to 'understand' its full meaning, complete with the speaker or writer's intent and sentiment. NLP drives computer programs that translate text from one language to another, respond to spoken commands, and summarize large volumes of text rapidly—even in real time. There's a good chance we've interacted with NLP in the

IARJSET

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Vol. 8, Issue 6, June 2021

DOI: 10.17148/IARJSET.2021.86148

form of voice-operated GPS systems, digital assistants, speech-to-text dictation software, customer service chatbots, and other consumer conveniences. But NLP also plays a growing role in enterprise solutions that help streamline business operations, increase employee productivity, and simplify mission-critical business processes.

CNN image classifications take an input image, process it and classify it under certain categories (E.g., Dog, Cat, Tiger, Lion). Computers sees an input image as array of pixels. Based on the image resolution, it will see h x w x d (h = Height, w = Width, d = Dimension). E.g. An image of 6 x 6 x 3 array of matrix of RGB (3 refers to RGB values) and an image of 4 x 4 x 1 array of matrix of grayscale image where 3 and 1 are the number of color values required to represent an image pixel. The followings are the type of layers which as commonly found in the CNNs:
1. Convolution – The main building block of CNN is the convolutional layer which is a mathematical operation to merge two sets of information. The convolution is applied on the input data using a convolution filter to produce a feature map. The first layer to extract features from an input image is convolution.
2. Striding – The number of shifts over the input image is called stride. Example- When stride is 1 then we move the filters to 1 pixel at a time.
3. Non-Linearity – The Activation function generally used is ReLU stands for Rectified Linear Unit for a non-linear operation. The output is:

$$\{F(x) = \max(O, x)\}$$

4. Pooling Layer – After a convolution operation we usually perform pooling to reduce the dimensionality. This enables us to reduce the number of parameters, which in turn shortens the training time and reduces chances of overfitting. Pooling layers down sample each feature map independently, reducing the height and width, keeping the depth intact. Contrary to the convolution operation, pooling has no parameters. It slides a window over its input, and simply takes the maximum value in the window. The largest element from the feature map is selected by using this operation.

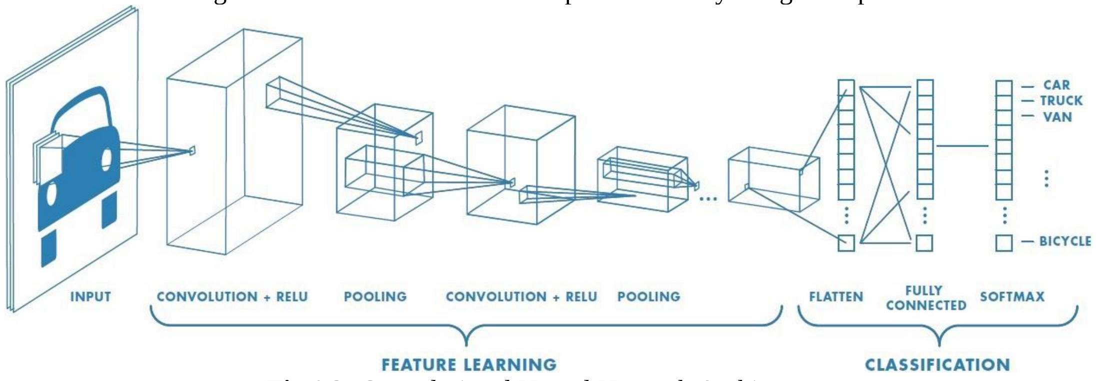


Fig 1.3: Convolutional Neural Network Architecture

LITERATURE REVIEW

We have decided to use the dataset of MNIST [1], there have been several accomplishments that has been achieved using this dataset. Even before using Deep learning, Handwritten text recognition has been made possible, however their accuracies were really low, or they had a relatively small dataset as said by Line Eikvil [2]. In this paper, usage of OCR has been discussed such as in Speech Recognition, Radio Frequency, Vision systems, Magnetic Stripes, Bar Code and Optical Mark Reading. A popular machine learning task is classifying the MNIST dataset, which is dataset of numbers. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis by Simard, Steinkraus and Platt is a valuable paper for understanding usage of convoluted neural networks (CNNs). For word recognition, a Paper by Pham et al., used a 2-layer CNN which fed into a bidirectional recurrent neural network (RNN) with Long Short-Term Memory (LSTM) cells [3].

The best model implemented, according to us, is by Graves and Schmiduber with a multidimensional RNN structure. [4] Another paper on 'Handwritten Text Recognition' by M.J. Castro-Bleda dealt with dataset with slanted words as well and corrected them during pre-processing. [5] Development of English Handwritten Recognition Using Deep Neural Network by Teddy Surya and Ahmad Fakhrur uses a Deep Neural Network model having two Encoding layer and one SoftMax layer on the EMNIST dataset. Their accuracy using DNN was way better than the earlier proposed patterned and feedforward net ANN (Artificial Neural Networks). [6] Handwritten text recognition can also be achieved on basis of Relaxation Convolutional Neural Networks(R-CNN) and alternatively trained relaxation convolutional neural networks (ATRCNN) as done by ChunPeng Wu and Wei Fan. [7] Our model achieved accuracy over 87 percent using Convolutional Neural Networks from Keras[8] library.

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

Vol. 8, Issue 6, June 2021

DOI: 10.17148/IARJSET.2021.86148

In the past few years, the CNN model has been extensively employed for handwritten digit recognition from the MNIST benchmark database. Some researchers have reported accuracy as good as 98% or 99% for handwritten digit recognition [8]. An ensemble model has been designed using a combination of multiple CNN models. The recognition experiment was carried out for MNIST digits, and an accuracy of 99.73% was reported [9]. Later, this "7-net committee" was extended to the "35-net committee" experiment, and the improved recognition accuracy was reported as 99.77% for the same MNIST dataset. An extraordinary recognition accuracy of 99.81% was reported by Niu and Suen by integrating the SVM (support vector machine) capability of minimizing the structural risk and the capability of a CNN model for extracting the deep features for the MNIST digit recognition experiment [1]. The bend directional feature maps were investigated using CNN for in-air handwritten Chinese character recognition [9]. Recently, the work of Alvear-Sandoval et al. achieved a 0.19% error rate for MNIST by building diverse ensembles of deep neural networks (DNN). However, on careful investigation, it has been observed that the high recognition accuracy of MNIST dataset images is achieved through ensemble methods only. Ensemble methods help in improving the classification accuracy but at the cost of high testing complexity and increased computational cost for real-world application.
The IAM dataset is a collection of handwritten alphanumeric derived from the NIST Special Database 19. Each image is converted to a 28x28 format and dataset structure that directly matches the dataset is used. The training set has 697932 images and test set has 116323 of uppercase and lowercase alphabets and numerals from 0-9 which are mapped to their corresponding classes. The test set and training set is available in the form of list within list. Each item of outer list represents an image and inner list represents the intensity values of 784 pixels (because size of image is 28 x 28 pixels) ranging from 0-255. The test images as well as train images have a white foreground and black background. Both the test images as well as train images are horizontally flipped and rotated 90 degrees clockwise. Y train and Y test both are arrays which contain number ranging from 0 to 61 as there are 10 numerals from 0-9 and 26 uppercase and lowercase alphabets each which adds up to 62. [1]

SYSTEM DESIGN

Handwritten Text Recognition (HTR) systems consist of handwritten text in the form of scanned images as shown in figure 1. we are going to build a Neural Network (NN) which is trained on word-images from the IAM dataset. because the input layer (and therefore also all the opposite layers) are often kept small for word-images, NN-training is possible on the CPU (of course, a GPU would be better). For the implementation of HTR, the minimum requirement is TF.

Secretary
Secretary

Fig 4.1: Image of word taken from IAM Dataset

The block diagram of the proposed E handwritten character classification is shown in Figure 2. The proposed recognition technique relies on a convolutional neural network model (CNN) with a feature mapped output layer. Our proposed model will classify the given input out of 10 classes using CNN while classifying the Urdu numeral. Similarly, while classifying the Urdu character, the same model will classify the given character out of 12 classes (see Figure 2). The detail about the different phases of our proposed model will come in the following subsections.

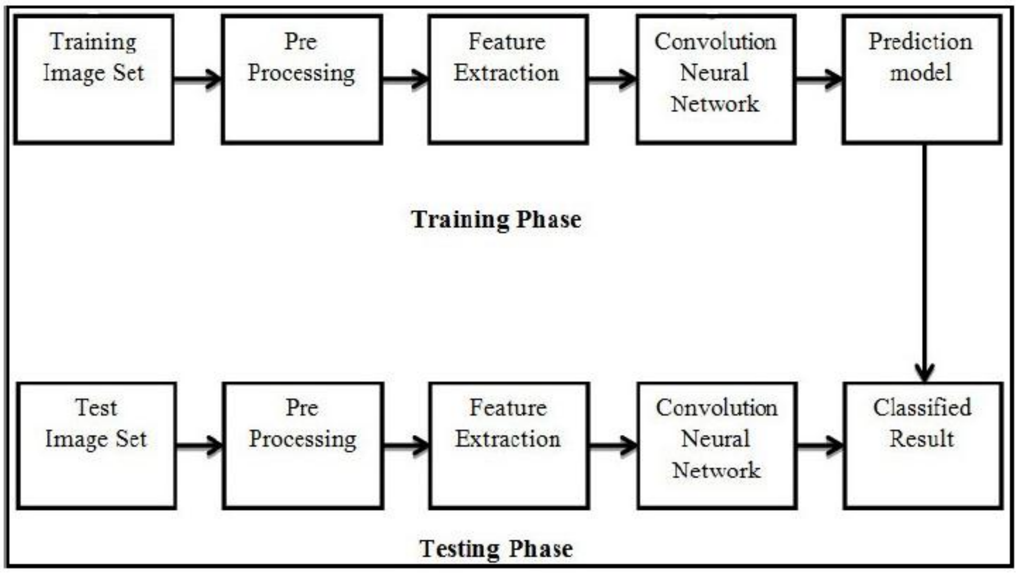


Fig 4.2: Block diagram of proposed handwritten character classification system

Process Model are processes of the same nature that are classified together into a model. Thus, a process model is a description of a process at the type of level. Since the process model is at the type of level, a process is an instantiation of it. The same process model is used repeatedly for the development of many applications and thus, has many instantiations. One possible use of a process model is to prescribe how things must/should/could is done in contrast to the process itself which is really what happens. A process model is roughly anticipation of what the process will look like. What the process shall be determined during actual system development.

The goal of a process model is to be:
1. Descriptive
1. Track what happens during a process.
2. Take the point of view of an external observer who looks at the way a process has been performed and determines the improvements that must be made to make it perform more effectively or efficiently.
2. Prescriptive
1. Define the desired processes and how they should/could/might be performed.
2. Establish rules, guidelines, and behavior patterns which, if followed, would lead to the desired process performance. They can range from strict enforcement to flexible guidance.
3. Explanatory
1. Provide explanations about the rationale of processes.
2. Explore and evaluate the several possible courses of action based on rational arguments.
3. Establish an explicit link between processes and the requirements that the model needs to fulfil.
4. Pre-defines points at which data can be extracted for reporting purposes.

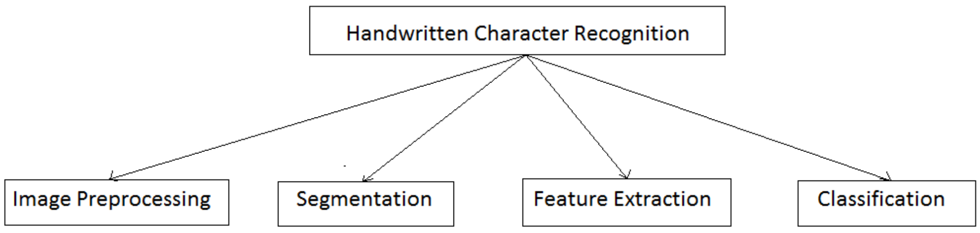


Fig 4.3: Breakdown model

1. **Image Preprocessing**
The image is preprocessed using different image processing algorithms like Inverting image, Gray Scale Conversion, and image thinning.

2. Segmentation
After preprocessing of the image segmentation is done. This is done with the help of following steps:

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

Vol. 8, Issue 6, June 2021

DOI: 10.17148/IARJSET.2021.86148

1. Remove the borders
2.Divide the text into rows
3.Divide the rows (lines) into words
4.Divide the word into letters

3. Feature Extraction

Once the character is segmented, we generate the binary glyphs and calculate the summation of each rows and columns values as an features.

4. Classification

In this phase, we are going to train and test the Neural Network, to finally identify the answer.

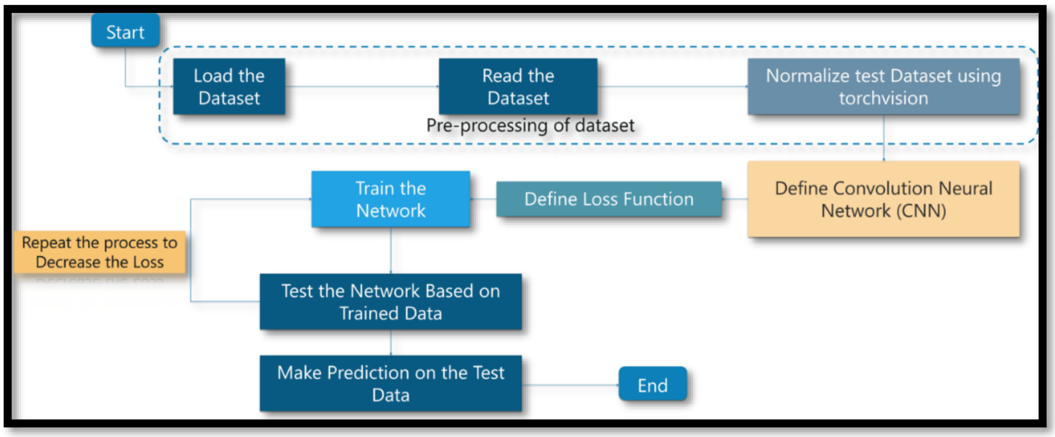

Start
Load the Dataset
Read the Dataset
Normalize test Dataset using torchvision
Pre-processing of dataset
Train the Network
Define Loss Function
Define Convolution Neural Network (CNN)
Repeat the process to Decrease the Loss
Test the Network Based on Trained Data
Make Prediction on the Test Data
End

Fig 4.4: Process Flow Diagram for Model Creation

The Model Architecture proposed involves pre-processing the data and then feeding it into the CNN, after which it will be feeded into the RNN network which will then finalise and give us our output.

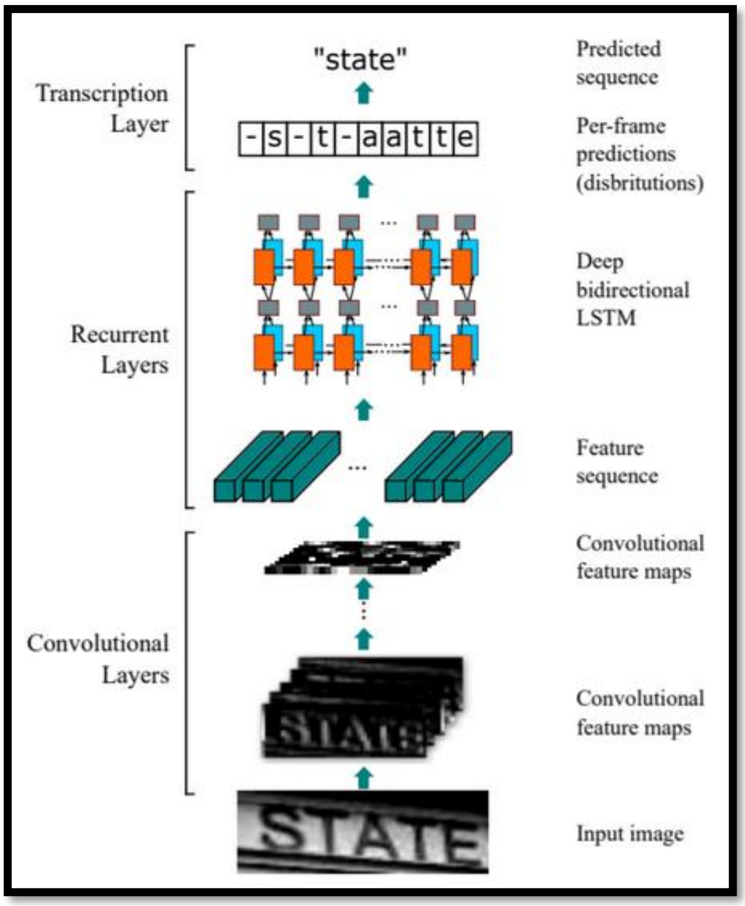


Fig 4.5: Proposed Model Architecture

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
Vol. 8, Issue 6, June 2021
DOI: 10.17148/IARJSET.2021.86148

CONTROL FLOW DIAGRAM

The large class of applications having following characteristics requires control flow modelling:
• The applications that are driven by events rather than data.
• The applications that produce control flow information rather than reports or displays.
• The application that process information in specific time.

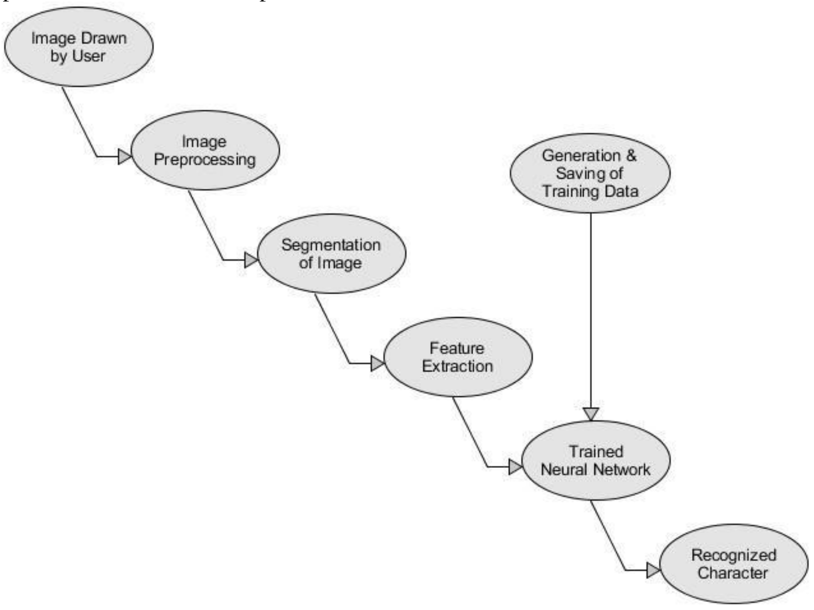


Fig 4.6: Control Flow Diagram for HCR System

DEPLOYMENT DIAGRAM

A deployment diagram shows the allocation of processes to processors in the physical design of a system. A deployment diagram may represent all or part of the process architecture of a system.

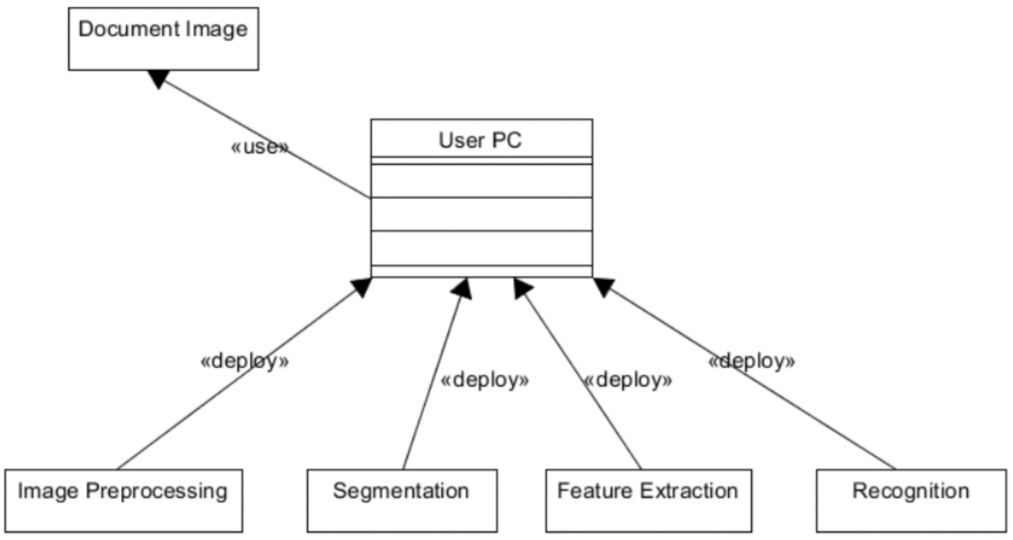


Fig 4.7: Deployment Diagram

Training the Model:
• The relevant dataset chosen by us, MNIST dataset will be loaded and a subset of it will be used, after dividing it into training and testing set.
• Firstly, we are looking to develop a model architecture with a mixture of CNN, RNN and other desired layers, using DL libraries of Python.

IARJSET

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

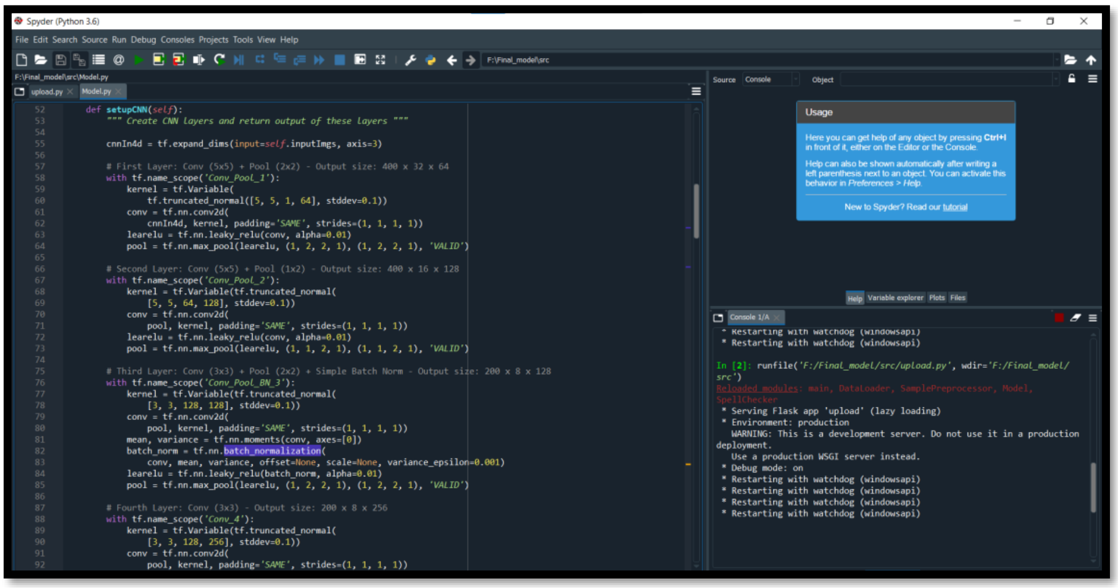
International Advanced Research Journal in Science, Engineering and Technology

Vol. 8, Issue 6, June 2021

DOI: 10.17148/IARJSET.2021.86148

• Then we are looking to first perform the pre-processing operations required to be done on the dataset, to feed the data then to the Training Model.
• After the Model will be put under training, and we will train multiple models by changing it's parameters just so slightly for us to have multiple models to test upon.

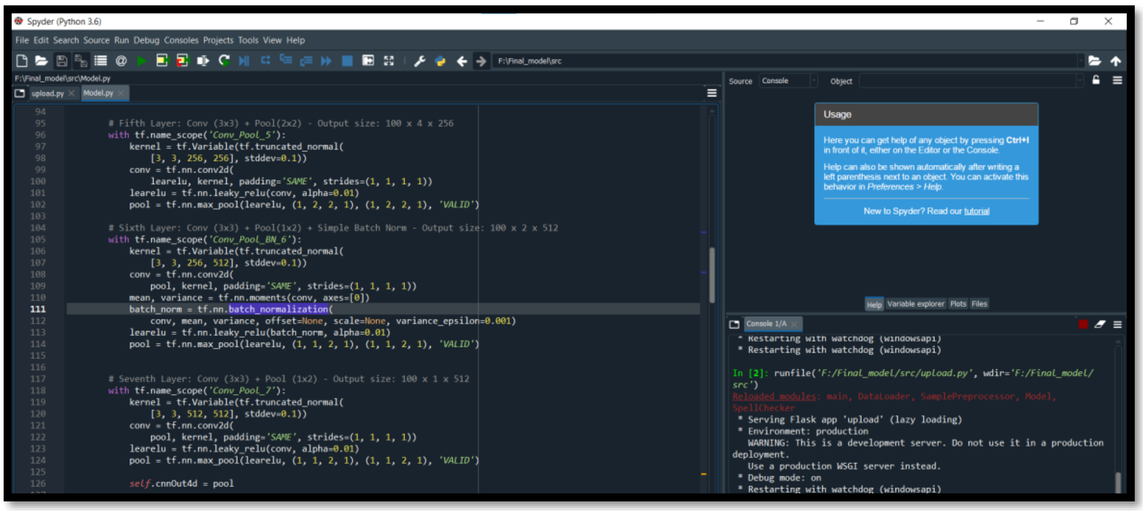
1. Setup CNN



```python
    def setupCNN(self):
        """ Create CNN layers and return output of these layers """

        cnnIn4d = tf.expand_dims(input=self.inputImgs, axis=3)

        # First Layer: Conv (5x5) + Pool (2x2) - Output size: 400 x 32 x 64
        with tf.name_scope('Conv_Pool_1'):
            kernel = tf.Variable(
                tf.truncated_normal([5, 5, 1, 64], stddev=0.1))
            conv = tf.nn.conv2d(
                cnnIn4d, kernel, padding='SAME', strides=(1, 1, 1, 1))
            learelu = tf.nn.leaky_relu(conv, alpha=0.01)
            pool = tf.nn.max_pool(learelu, (1, 2, 2, 1), (1, 2, 2, 1), 'VALID')

        # Second Layer: Conv (5x5) + Pool (1x2) - Output size: 400 x 16 x 128
        with tf.name_scope('Conv_Pool_2'):
            kernel = tf.Variable(tf.truncated_normal(
                [5, 5, 64, 128], stddev=0.1))
            conv = tf.nn.conv2d(
                pool, kernel, padding='SAME', strides=(1, 1, 1, 1))
            learelu = tf.nn.leaky_relu(conv, alpha=0.01)
            pool = tf.nn.max_pool(learelu, (1, 1, 2, 1), (1, 1, 2, 1), 'VALID')

        # Third Layer: Conv (3x3) + Pool (2x2) + Simple Batch Norm - Output size: 200 x 8 x 128
        with tf.name_scope('Conv_Pool_BN_3'):
            kernel = tf.Variable(tf.truncated_normal(
                [3, 3, 128, 128], stddev=0.1))
            conv = tf.nn.conv2d(
                pool, kernel, padding='SAME', strides=(1, 1, 1, 1))
            mean, variance = tf.nn.moments(conv, axes=[0])
            batch_norm = tf.nn.batch_normalization(
                conv, mean, variance, offset=None, scale=None, variance_epsilon=0.001)
            learelu = tf.nn.leaky_relu(batch_norm, alpha=0.01)
            pool = tf.nn.max_pool(learelu, (1, 2, 2, 1), (1, 2, 2, 1), 'VALID')

        # Fourth Layer: Conv (3x3) - Output size: 200 x 8 x 256
        with tf.name_scope('Conv_4'):
            kernel = tf.Variable(tf.truncated_normal(
                [3, 3, 128, 256], stddev=0.1))
            conv = tf.nn.conv2d(
                pool, kernel, padding='SAME', strides=(1, 1, 1, 1))
```

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.
New to Spyder? Read our tutorial

```
 * Restarting with watchdog (windowsapi)
 * Restarting with watchdog (windowsapi)

In [2]: runfile('F:/Final_model/src/upload.py', wdir='F:/Final_model/
src')
Reloaded modules: main, DataLoader, SamplePreprocessor, Model,
SpellChecker
 * Serving Flask app 'upload' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production
deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with watchdog (windowsapi)
 * Restarting with watchdog (windowsapi)
 * Restarting with watchdog (windowsapi)
 * Restarting with watchdog (windowsapi)
```

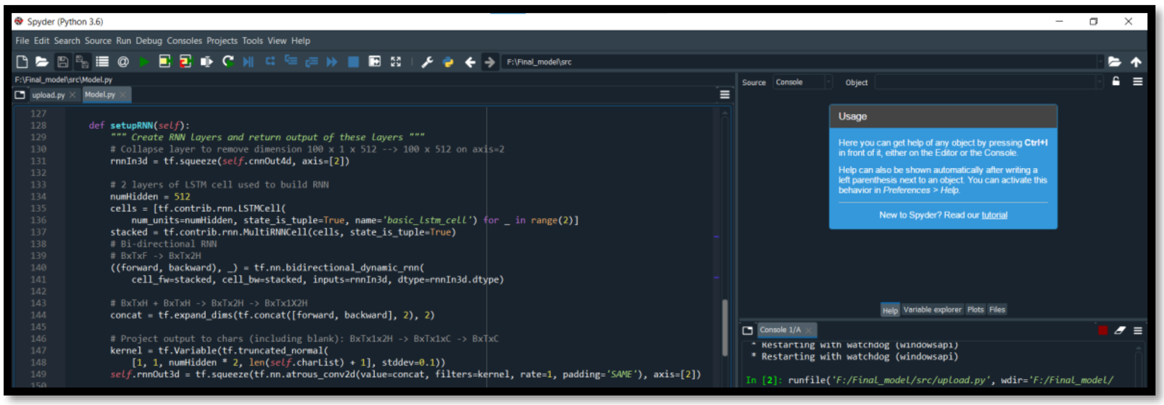


```python

        # Fifth Layer: Conv (3x3) + Pool(2x2) - Output size: 100 x 4 x 256
        with tf.name_scope('Conv_Pool_5'):
            kernel = tf.Variable(tf.truncated_normal(
                [3, 3, 256, 256], stddev=0.1))
            conv = tf.nn.conv2d(
                learelu, kernel, padding='SAME', strides=(1, 1, 1, 1))
            learelu = tf.nn.leaky_relu(conv, alpha=0.01)
            pool = tf.nn.max_pool(learelu, (1, 2, 2, 1), (1, 2, 2, 1), 'VALID')

        # Sixth Layer: Conv (3x3) + Pool(1x2) + Simple Batch Norm - Output size: 100 x 2 x 512
        with tf.name_scope('Conv_Pool_BN_6'):
            kernel = tf.Variable(tf.truncated_normal(
                [3, 3, 256, 512], stddev=0.1))
            conv = tf.nn.conv2d(
                pool, kernel, padding='SAME', strides=(1, 1, 1, 1))
            mean, variance = tf.nn.moments(conv, axes=[0])
            batch_norm = tf.nn.batch_normalization(
                conv, mean, variance, offset=None, scale=None, variance_epsilon=0.001)
            learelu = tf.nn.leaky_relu(batch_norm, alpha=0.01)
            pool = tf.nn.max_pool(learelu, (1, 1, 2, 1), (1, 1, 2, 1), 'VALID')


        # Seventh Layer: Conv (3x3) + Pool (1x2) - Output size: 100 x 1 x 512
        with tf.name_scope('Conv_Pool_7'):
            kernel = tf.Variable(tf.truncated_normal(
                [3, 3, 512, 512], stddev=0.1))
            conv = tf.nn.conv2d(
                pool, kernel, padding='SAME', strides=(1, 1, 1, 1))
            learelu = tf.nn.leaky_relu(conv, alpha=0.01)
            pool = tf.nn.max_pool(learelu, (1, 1, 2, 1), (1, 1, 2, 1), 'VALID')

            self.cnnOut4d = pool
```

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.
New to Spyder? Read our tutorial

```
 * Restarting with watchdog (windowsapi)
 * Restarting with watchdog (windowsapi)

In [2]: runfile('F:/Final_model/src/upload.py', wdir='F:/Final_model/
src')
Reloaded modules: main, DataLoader, SamplePreprocessor, Model,
SpellChecker
 * Serving Flask app 'upload' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production
deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with watchdog (windowsapi)
```

2. Setup RNN



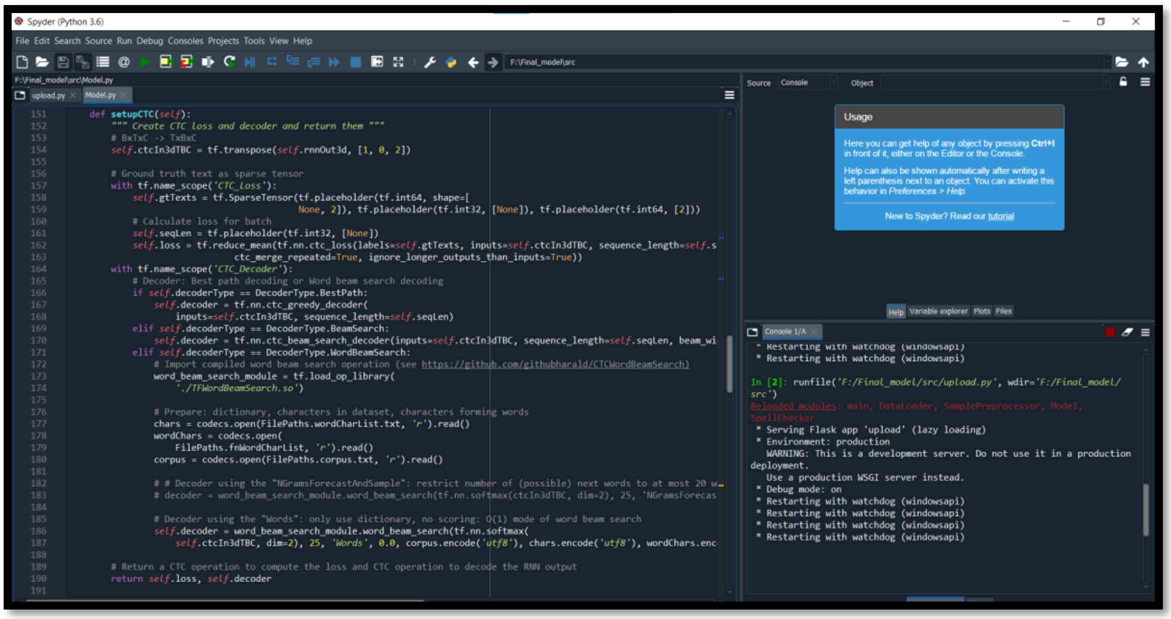
```python

    def setupRNN(self):
        """ Create RNN layers and return output of these layers """
        # Collapse layer to remove dimension 100 x 1 x 512 --> 100 x 512 on axis=2
        rnnIn3d = tf.squeeze(self.cnnOut4d, axis=[2])

        # 2 layers of LSTM cell used to build RNN
        numHidden = 512
        cells = [tf.contrib.rnn.LSTMCell(
            num_units=numHidden, state_is_tuple=True, name='basic_lstm_cell') for _ in range(2)]
        stacked = tf.contrib.rnn.MultiRNNCell(cells, state_is_tuple=True)
        # Bi-directional RNN
        # BxTxF -> BxTx2H
        ((forward, backward), _) = tf.nn.bidirectional_dynamic_rnn(
            cell_fw=stacked, cell_bw=stacked, inputs=rnnIn3d, dtype=rnnIn3d.dtype)

        # BxTxH + BxTxH -> BxTx2H -> BxTx1X2H
        concat = tf.expand_dims(tf.concat([forward, backward], 2), 2)

        # Project output to chars (including blank): BxTx1x2H -> BxTx1xC -> BxTxC
        kernel = tf.Variable(tf.truncated_normal(
            [1, 1, numHidden * 2, len(self.charList) + 1], stddev=0.1))
        self.rnnOut3d = tf.squeeze(tf.nn.atrous_conv2d(value=concat, filters=kernel, rate=1, padding='SAME'), axis=[2])
```

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.
New to Spyder? Read our tutorial

```
 * Restarting with watchdog (windowsapi)
 * Restarting with watchdog (windowsapi)

In [2]: runfile('F:/Final_model/src/upload.py', wdir='F:/Final_model/
```

DOI: 10.17148/IARJSET.2021.86148

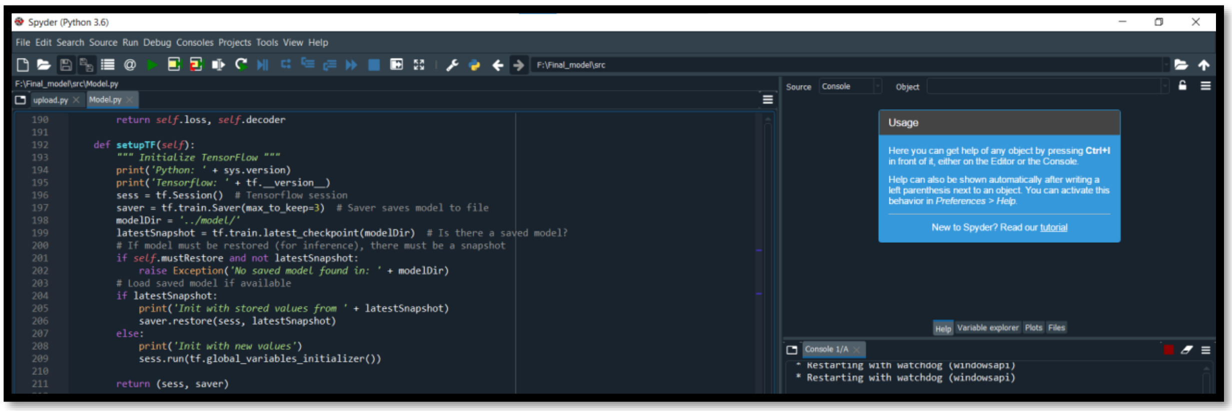
3. Setup CTC


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
F:\Final_model\src
F:\Final_model\src\Model.py
upload.py
Model.py
Source Console Object
Usage
Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Help.
New to Spyder? Read our tutorial
Help Variable explorer Plots Files
Console 1/A

```python
    def setupCTC(self):
        """ Create CTC loss and decoder and return them """
        # BxTxC -> TxBxC
        self.ctcIn3dTBC = tf.transpose(self.rnnOut3d, [1, 0, 2])

        # Ground truth text as sparse tensor
        with tf.name_scope('CTC_Loss'):
            self.gtTexts = tf.SparseTensor(tf.placeholder(tf.int64, shape=[
                                           None, 2]), tf.placeholder(tf.int32, [None]), tf.placeholder(tf.int64, [2]))
            # Calculate loss for batch
            self.seqLen = tf.placeholder(tf.int32, [None])
            self.loss = tf.reduce_mean(tf.nn.ctc_loss(labels=self.gtTexts, inputs=self.ctcIn3dTBC, sequence_length=self.s
                                 ctc_merge_repeated=True, ignore_longer_outputs_than_inputs=True))
        with tf.name_scope('CTC_Decoder'):
            # Decoder: Best path decoding or Word beam search decoding
            if self.decoderType == DecoderType.BestPath:
                self.decoder = tf.nn.ctc_greedy_decoder(
                    inputs=self.ctcIn3dTBC, sequence_length=self.seqLen)
            elif self.decoderType == DecoderType.BeamSearch:
                self.decoder = tf.nn.ctc_beam_search_decoder(inputs=self.ctcIn3dTBC, sequence_length=self.seqLen, beam_wi
            elif self.decoderType == DecoderType.WordBeamSearch:
                # Import compiled word beam search operation (see https://github.com/githubharald/CTCWordBeamSearch)
                word_beam_search_module = tf.load_op_library(
                    './TFWordBeamSearch.so')

                # Prepare: dictionary, characters in dataset, characters forming words
                chars = codecs.open(FilePaths.wordCharList.txt, 'r').read()
                wordChars = codecs.open(
                    FilePaths.fnWordCharList, 'r').read()
                corpus = codecs.open(FilePaths.corpus.txt, 'r').read()

                # # Decoder using the "NGramsForecastAndSample": restrict number of (possible) next words to at most 20 w
                # decoder = word_beam_search_module.word_beam_search(tf.nn.softmax(ctcIn3dTBC, dim=2), 25, 'NGramsForecas

                # Decoder using the "Words": only use dictionary, no scoring: O(1) mode of word beam search
                self.decoder = word_beam_search_module.word_beam_search(tf.nn.softmax(
                    self.ctcIn3dTBC, dim=2), 25, 'Words', 0.0, corpus.encode('utf8'), chars.encode('utf8'), wordChars.enc

        # Return a CTC operation to compute the loss and CTC operation to decode the RNN output
        return self.loss, self.decoder

```

```
 * Restarting with watchdog (windowsapi)
 * Restarting with watchdog (windowsapi)

In [2]: runfile('F:/Final_model/src/upload.py', wdir='F:/Final_model/
src')
Reloaded modules: main, DataLoader, SamplePreprocessor, Model,
SpellChecker
 * Serving Flask app 'upload' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production
deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with watchdog (windowsapi)
 * Restarting with watchdog (windowsapi)
 * Restarting with watchdog (windowsapi)
 * Restarting with watchdog (windowsapi)
```

4. Setup TensorFlow


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
F:\Final_model\src
F:\Final_model\src\Model.py
upload.py
Model.py
Source Console Object
Usage
Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Help.
New to Spyder? Read our tutorial
Help Variable explorer Plots Files
Console 1/A

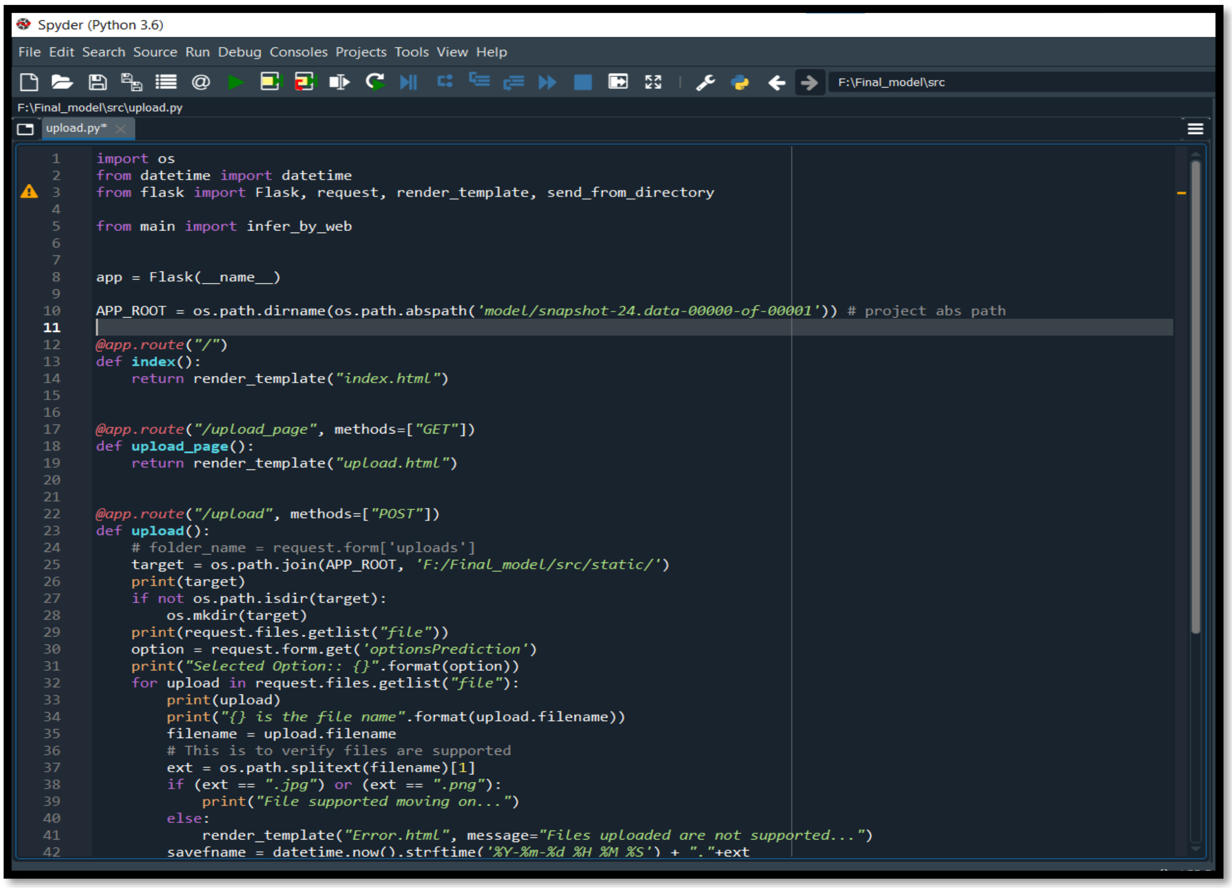
```python
        return self.loss, self.decoder

    def setupTF(self):
        """ Initialize TensorFlow """
        print('Python: ' + sys.version)
        print('Tensorflow: ' + tf.__version__)
        sess = tf.Session()  # Tensorflow session
        saver = tf.train.Saver(max_to_keep=3)  # Saver saves model to file
        modelDir = '../model/'
        latestSnapshot = tf.train.latest_checkpoint(modelDir)  # Is there a saved model?
        # If model must be restored (for inference), there must be a snapshot
        if self.mustRestore and not latestSnapshot:
            raise Exception('No saved model found in: ' + modelDir)
        # Load saved model if available
        if latestSnapshot:
            print('Init with stored values from ' + latestSnapshot)
            saver.restore(sess, latestSnapshot)
        else:
            print('Init with new values')
            sess.run(tf.global_variables_initializer())

        return (sess, saver)
```

```
 * Restarting with watchdog (windowsapi)
 * Restarting with watchdog (windowsapi)
```

Testing the Model:

- The models trained in the earlier steps will not be tested on the data that it had not seen before.
- For, different models we will be getting different accuracies and then we will be able to have the best accurate model.
- We can now use this model, to finally build and deploy our HTR service for the users.

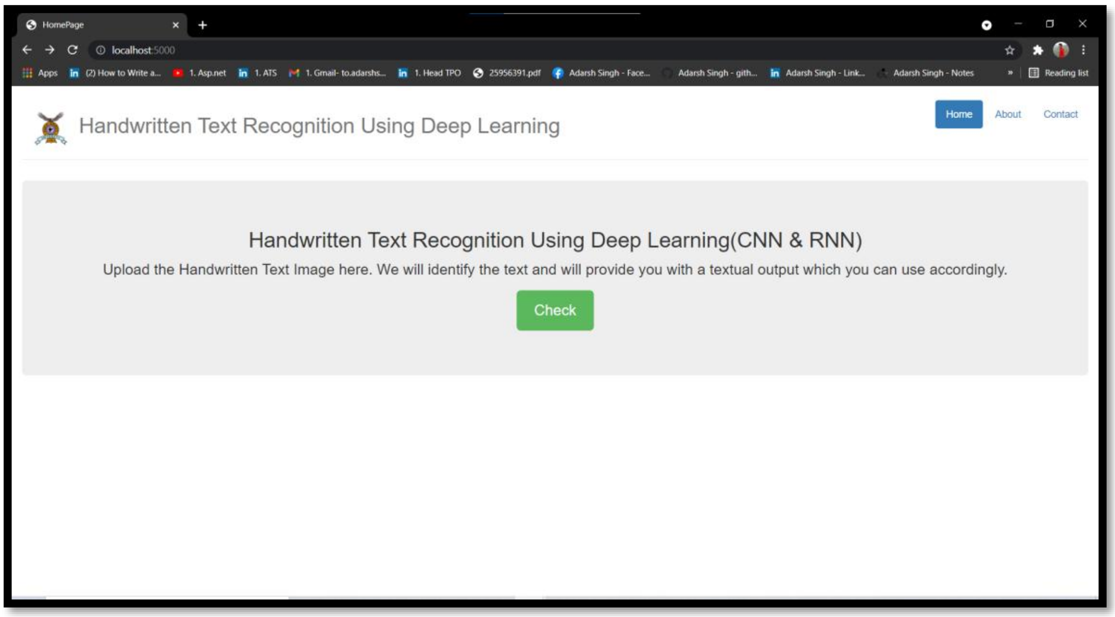
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
Vol. 8, Issue 6, June 2021
DOI: 10.17148/IARJSET.2021.86148


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
F:\Final_model\src
F:\Final_model\src\upload.py
upload.py*

```
import os
from datetime import datetime
from flask import Flask, request, render_template, send_from_directory

from main import infer_by_web


app = Flask(__name__)

APP_ROOT = os.path.dirname(os.path.abspath('model/snapshot-24.data-00000-of-00001')) # project abs path

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


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


@app.route("/upload", methods=["POST"])
def upload():
    # folder_name = request.form['uploads']
    target = os.path.join(APP_ROOT, 'F:/Final_model/src/static/')
    print(target)
    if not os.path.isdir(target):
        os.mkdir(target)
    print(request.files.getlist("file"))
    option = request.form.get('optionsPrediction')
    print("Selected Option:: {}".format(option))
    for upload in request.files.getlist("file"):
        print(upload)
        print("{} is the file name".format(upload.filename))
        filename = upload.filename
        # This is to verify files are supported
        ext = os.path.splitext(filename)[1]
        if (ext == ".jpg") or (ext == ".png"):
            print("File supported moving on...")
        else:
            render_template("Error.html", message="Files uploaded are not supported...")
        savefname = datetime.now().strftime('%Y-%m-%d %H %M %S') + "."+ext
```

5.2 Proposed System

In the proposed system, after training and saving the model, we are looking to deploy the model in the form of an API, or by using any kind of deploying service into the web. Then we will create an interface with the model where the user will be able to access it and then use it accordingly. We are looking to develop a website kind of an interface at first whereas per the user's requirement the model will be called to perform and identify the characters.



Handwritten Text Recognition Using Deep Learning

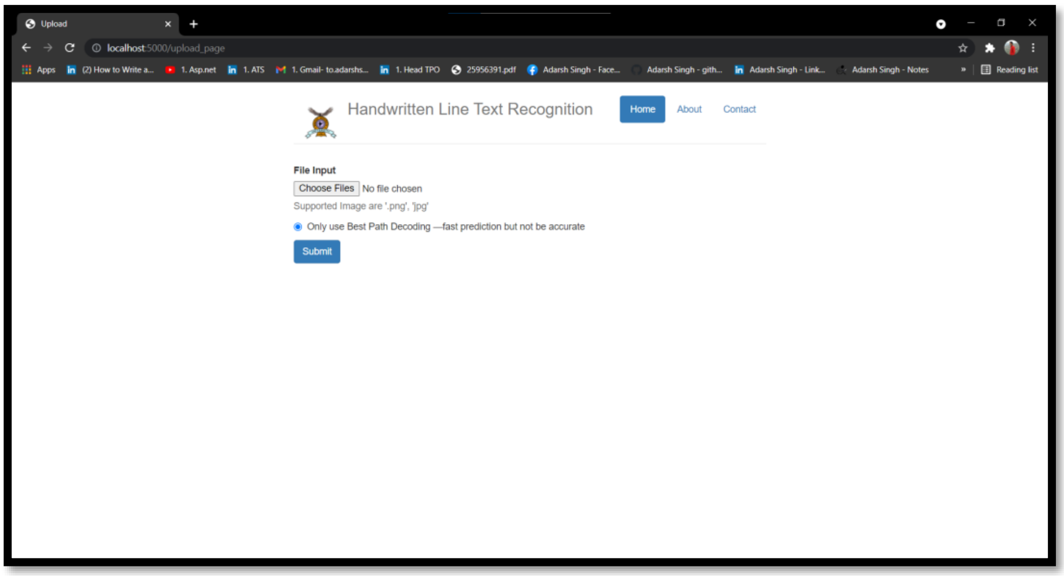
Handwritten Text Recognition Using Deep Learning(CNN & RNN)

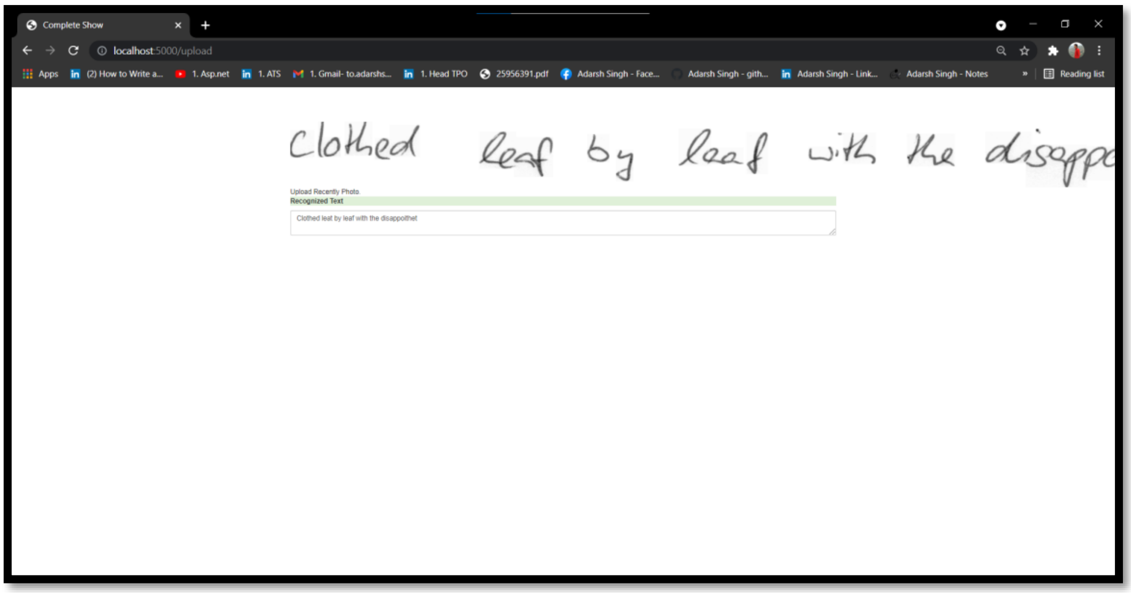
Upload the Handwritten Text Image here. We will identify the text and will provide you with a textual output which you can use accordingly.

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

Vol. 8, Issue 6, June 2021

DOI: 10.17148/IARJSET.2021.86148


Upload
localhost:5000/upload_page
Handwritten Line Text Recognition
Home
About
Contact
File Input
Choose Files
No file chosen
Supported Image are '.png', 'jpg'
Only use Best Path Decoding —fast prediction but not be accurate
Submit



Clothed leat by leaf with the disappoithet

CONCLUSION

Using modern day techniques like neural networks to implement deep learning to solve basic tasks which are done with a blink of an eye by any human like text recognition is just scratching the surface of the potential behind machine learning. There are infinite possibilities and application of this technology. Traditional OCR used to work like biometric device. Photo sensor technology was used to gather the match points of physical attributes and then convert it into database of known types. But with the help of modern-day techniques like convolution neural networks we are able to scan and understand words with an accuracy never seen before in history.

We aim to achieve highest level of accuracy that is possible for a lightweight easy to train model, with the help of freely available GPU resources and Compute time. We will be using and finalizing the best performing CRNN architecture for our model.

REFERENCES

1. G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "Emnist: an extension of mnist to handwritten letters," arXiv preprint arXiv:1702.05373, 2017.
2. L. Eikvil, "Optical character recognition," citeseer. ist. psu. edu/142042. html, 1993.
3. A. Graves and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks," in Advances in neural information processing systems, 2009, pp. 545–552.

IARJSET

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Vol. 8, Issue 6, June 2021

DOI: 10.17148/IARJSET.2021.86148

4. V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, “Dropout improves recurrent neural networks for handwriting recognition,” in Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on. IEEE, 2014, pp. 285–290.
5. S. Espana-Boquera, M. J. Castro-Bleda, J. Gorbe-Moya, and F. Zamora-Martinez, “Improving offline handwritten text recognition with hybrid hmm/ann models,” IEEE transactions on pattern analysis and machine intelligence, vol. 33, no. 4, pp. 767–779, 2011.
6. T. S. Gunawan, A. F. R. M. Noor, and M. Kartiwi, “Development of english handwritten recognition using deep neural network,” 2018.
7. C. Wu, W. Fan, Y. He, J. Sun, and S. Naoi, “Handwritten character recognition by alternately trained relaxation convolutional neural network,” 2014 14th International Conference on Frontiers in Handwriting Recognition, pp. 291–296, 2014.
8. F. Chollet et al., “Keras,” https://github.com/fchollet/keras, 2015.
9. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Man´e, R. Monga, S. Moore, D. Murray,
C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Vi´egas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015