Handwritten Text Recognition and Generation using Neural Networks

Sai Kumar R, Naveen M, Shaneel Reddy L
Students, School of Electronics Engineering (SENSE), VIT Chennai

Abstract

With the development in the machine learning and computer vision field handwritten text recognition, that is recognizing intelligible handwritten text from an image, and text generation has been made possible. Written words can now be identified and converted to text format. Even though deep learning methods provide quite a boost in performance there are limitations in the accuracy as minimal changes or preprocessing bring major deviations in results. Images contain noise and hence need to be pre-processed. More improvement in accuracy can be brought by using deep learning RNN (Recurrent Neural Network) models such as LSTM (Long Short-Term Memory) models which can be trained using pre-existing datasets to recognize and generate text. The letters can be segmented and extracted from the image before being classified by the neural network. Different preprocessing techniques have varying effects on the outcome of the model. Techniques such as low-intensity pixel removal can be used to further reduce noise. The generated text is formatted and saved in editable formats such as Word and PDF document files. Therefore, our approach results in highly improved precision when recognizing and generating text from handwritten images, thereby promising a much more reliable solution for document digitization.

Keywords: CNN, RNN, LSTM, Image Preprocessing, Deep Learning, IAM dataset

Introduction

Handwritten text recognition enables machines to recognize, extract, and convert handwritten text from images to machine-readable formats. As mobile phones have been widely spread and used, there is a need to replace traditional OCR (Optical Character Recognition) scanners with software that converts handwritten text from mobile phone images to machine-readable text files. With the help of the deep learning method, this is made possible albeit with lesser accuracy. The accuracy of the outcome can be improved using various techniques.

Preprocessing of images is a crucial step in eliminating noise from mobile phone images and thus helps in improving accuracy significantly. There are many steps in doing so. Noise reduction is important before extracting text from an image. This is done using techniques such as binarization and thresholding to get a clearer image before extracting text from it. Rescaling of the image is done to standardize any inconsistency in image size. Standardizing the size of an image helps in improving the accuracy of the outcome. These methods of preprocessing help in improving the accuracy of the outcome even before the text is extracted.

During the extraction of text, the image can be segmented into individual characters. Each letter or character is detected and separated. Then these segments are converted into a format that can be fed into the neural network. IAM handwritten forms dataset is used to train the deep learning model. This dataset is a valuable resource that can be used in training models on handwritten text. The data that is fed to the neural network will give its output in the form of a sequence of words or letters.

Now that the text is recognized the next step is to format it. This is done with the help of string manipulation functions in Python. This is done to ensure proper punctuation, spacing, etc. This text can be saved into a file using the 'python-docx' library for saving it into a Word document which is editable. Libraries like 'report lab' can be used to save this into a pdf file.

People have moved on from using traditional OCR for handwritten text recognition. Traditional OCR had many limitations due to the variability in the handwriting. Nowadays deep learning methods like CNNs and RNNs are more prevalent and are replacing older approaches such as HMMs (Hidden Markov Models) and SVMs (Support Vector Machines) (Doe et al., 2015). CNNs are capable of learning hierarchical features from images (LeCun et al., 1998) while the LSTM network can effectively process and address the sequential nature of handwritten text (Graves et al., 2009). Recognition of text can be improved by using hybrid models that use both RNNs and CNNs (Bluche et al., 2017), use transfer learning

with pre-trained models like VGG and ResNet (Zhang et al., 2020).

The remainder of this paper is organized as follows: Section II provides a detailed review of related work, discussing previous research in the field of handwritten text recognition and generation using neural networks. Section III outlines the proposed methodology, including the problem definition, model architecture, algorithm description, and data preprocessing steps. Section IV presents the experimental setup, results, and evaluation of the model, comparing its performance with baseline models and discussing the outcomes. Finally, Section V concludes the paper, summarizing the key findings and contributions, and suggests potential directions for future work.

Related Works:

OCR is a text recognition technique that can read individual words, characters, or the full document. The IAM dataset, which consists of several photos with handwritten text on them, is used for this.

Convolutional Neural Networks for Handwritten Text Recognition:

CNNs are now a basic tool for image processing applications, such as the recognition of handwritten text. Their ability to recognize patterns in handwriting is greatly enhanced by their capacity to automatically learn and extract hierarchical features from photos.

The development of LeNet-5 architecture by Yann LeCun in 1998 was considered a stepping stone in this domain, as it demonstrated the powers of CNNs in recognizing handwritten digits, particularly using the MNIST dataset. A new type of CNNs, called multi-scale CNNs, has been introduced to further advancements in this field. To capture features at various resolutions, these architectures were designed. This is particularly useful for detecting handwritten text. Handling the variety of handwriting across different people has shown to benefit from the capacity to recognize features at several scales.

In this field, transfer learning has also been quite important. Researchers have been able to fine-tune pre-trained CNN models such as VGG and ResNet on particular handwriting datasets, which led to improvement in performance even when large labeled datasets were not available. This method has been widely adopted since it greatly lowers the quantity of training data needed and increases the models' generalizability.

Recurrent Neural Networks for Handwritten Text Recognition:

RNNs known as Long Short Term Memory (LSTM) networks are used for handwritten text recognition for sequence modeling. RNN is one of the most widely used algorithms for sequentially processing data and is mainly applied in speech and text recognition. RNNs are used for capturing temporal dependencies and for processing and detecting a sequence of handwritten characters or words. Traditional methods like Hidden Markov Models(HMMs) and Support Vector Machines(SVMs) were widely used in this but had problems with the sequential nature of handwritten texts. The Limitations of Traditional RNNs were later addressed by the introduction of LSTM and Gated Recurrent Unit (GRU) mainly concerning the issue of vanishing gradients.

Graves et al. (2009) presented the first application of LSTM networks to the offline recognition of handwritten texts. Their model outperformed the baseline by a huge amount when compared to the one using HMMs; this is because memory cells in LSTMs could model long-range dependencies in sequences of handwriting.

Pham et al. (2014) went further on this line of work, working CTC into RNNs themselves to directly map input sequences onto transcriptions without requiring any sort of pre-segmented data.

Various Handwritten text recognition architectures have been explored and The bidirectional LSTM is one of them which reads text in both forward and backward directions, so it specifically makes good use of the contextual information.

Bluche et al.(2017) presented a hybrid model where BiLSTM is combined with convolutional neural networks to intensify the feature extraction before feeding into the RNN.

A handwritten text recognizer model encounters many challenges. These include the differences in penmanship styles among individuals, wrinkling, poor quality photos, and unclear characters like the capital and lowercase letters "I" and "l," as well as the number "1". We can overcome the issues by training with an increasing variety of datasets and teaching it to comprehend context.

Proposed Methodology

Problem Definition :
The problem involves extracting text from images. The extracted text is then used to generate text into a text file.
Text extraction: OCR(Optical Character Recognition) extracts text from images.
Text generation: Text is generated based on the extracted images or 'seeds'.
The final target is to make an automated model that takes images of written text as input, extracts them, and generates meaningful and related continuations.

Model Architecture:
The architecture is divided into two main parts, the text extraction module and the text generation module.
The text is extracted using Pytesseract. Pytesseract is a Python wrapper for Google's Tesseract-OCR engine.
In this module, a predefined directory of images is fed. The images are processed using various methods and OCR is applied to them to recognize text. The images are first converted to a grayscale to simplify the data for OCR. This does so by reducing color-related noise. Then the images are resized to a uniform dimension. This helps maintain consistency and reduces computational costs. To standardize pixel values and hence enhance model performance, the mean and standard deviation values of the images are normalized to 0.50. To make it compatible with PyTorch-based operations the images are converted to a tensor. These preprocessed images are the ones sent to OCR for text recognition. OCR detects and extracts the text. This is used as the 'seed' for the text generation module.

The text generation is an LSTM-based RNN model. It is made to handle sequential data. Using the seed text it generates text character-by-character. To capture relationships between characters they are converted to dense vectors. Sequential contexts are stored in hidden and cell states, to capture long-term dependencies across characters. For character prediction, LSTM output is projected to vocabulary size. Probability distributions are generated to predict the next character. New characters are generated iteratively based on the seed text.

Algorithm Description:
The text is extracted from images using Pytesseract and is then input as a text seed.
Encoding: the seed text is encoded as embeddings which initialize the model's hidden state and cell state.

Sequential Character Prediction: At time-step t, the model predicts the next character conditioned on the previously generated characters and the hidden state h_{t-1} which updates hidden and cell states for coherence.
Temperature-controlled Sampling: The softmax layer outputs probabilities and has a temperature parameter controlling the diversity. Low temperatures give more conservative outputs, while high values introduce lots of creativity.
Recurrence: The model iterates over time steps until it achieves the desired sequence length. It gives text that makes sense and, therefore, fits the context from the seed.

Mathematical Formulation:
For Text extraction:

$P(Text|Image)=OCR(Image)$ where OCR is the Optical Character Recognition function provided by Pytesseract.

For text generation, the task can be described as a sequence modeling problem:

$$P(x_t|\boldsymbol{x}_1,x_2,\ldots,x_{t-1})=Model(x_{-t-1},h_{t-1})$$

where x_t is the predicted character at time t, conditioned on the previous sequence $x_1,\ldots,x_{t-1}$, and h_{t-1} is the hidden state.

Data:
The dataset containing images with textual content is downloaded from Kaggle and is stored in a local directory. This data is then sourced from this local directory. The Path library is used to load the image files. The dataset contains 1539 image files (5.5 GB) of textual content.
The images are converted to grayscale to avoid color-based noise. Then they are resized to a uniform dimension of 128x128 pixels and the pixels are normalized to a mean and standard deviation of 0.5. Images are converted to tensors to be compatible with PyTorch-based operations like To Tensor().

Results and Discussion

Experimental Setup :

Hardware:

1. Intel 11th Gen i5-1135G7
2. 20 GB DDR4 RAM
3. Intel Iris Xe Graphics

Software:
Libraries and Frameworks

PyTorch:
This library is used to train and implement the CNN-LSTM model architecture; PyTorch enables the construction of deep learning models with well-documented modules and is efficient in tensor computations while permitting GPU acceleration and thus a faster training process for image data processing and handling.

Pytesseract:
It is a Python wrapper for the Tesseract-OCR engine of Google, identifying text based on images. This tool performs the extraction of initial text data from pre-processed images from which the identified text is used as a "seed" input further processed and generated by the LSTM model.

python-docx and ReportLab:
- python-docx: This library can save the output text to an editable Word document format, making it easy to edit and share.
- ReportLab: The library saves the final output text in PDF file formats, which can then be stored as static, shareable documents or archived for distribution.

Data Processing Tools
Image Preprocessing

Grayscale Conversion: Convert every image to grayscale, and this alleviates data complexity because the important factor will be intensity rather than color for recognition of texts. Simultaneously, it also filters away the noise due to color.

Resizing: Resize Images to a uniform dimension that is 128x128 pixels, which makes input consistency uniform across the dataset. This can help in learning better by the model.

Normalization: The pixel values of the images get normalized to a mean and standard deviation of 0.5. This scaling stabilizes training and results in faster model convergence.

Tensor Conversion: Images are now converted into tensors. This is a format ready for PyTorch, which will enhance the fast processing of data while training the models.

Text Generation

-Character Embedding: There will be identifiable characters represented as a dense vector embedding which presents each character in the form of a numerical vector so that the LSTM model can capture sequential dependencies.

-Temperature-Controlled Sampling: The model softmax layer controls temperature that modulates sample diversity of text-generated outputs. Lows act as a conservative predictor, while highs encourage variability and creativity. This setup streams through image preprocessing to text recognition generation by saving the output in user-friendly formats, editable, and shareable formats.

Baseline Comparison
NotesAI was compared against traditional OCRs and baselines for handwritten text recognition on the architecture of a Recurrent Neural Network (RNN). Classical OCRs such as Tesseract fundamentally rely on either rule-based or statistical techniques, like Hidden Markov Models (HMMs), which do better on printed text but significantly fail on handwritten text due to variations in style and complexity.

The RNN-based architecture in the NotesAI system gives the system advantages over these baselines:

1. Sequential Learning with RNN: The RNN can functionally model sequential conditioning of characters, so dependencies between characters will be considered. Therefore, the notion of many connected and continued characters can be handled properly when captured within the RNN.

2. Context-Aware Predictions: Unlike usual baseline OCR systems which are designed to do their operation on single, disjointed characters, RNNs take previous input into consideration. Thus, the letters are better interpreted because of contextualization.

3. Convergence at stability: The train and validation logs in the appendix confirm that NotesAI achieved stable low training and validation losses: it obtained a high accuracy compared with baseline OCR models and trained efficiently.

Interpretation
The RNN structure of the NotesAI model allows for strong performance because it can easily manage sequential and contextual information in the

handwritten text. Some of the prominent reasons why the model is so effective are:

1. Sequential Data Handling: Due to the fact that hand-written text is inherently sequential, the RNN would be better capable of understanding temporal dependencies that might be present. This is particularly useful for writing, where characters may connect or overlap. The RNN enables the model to remember previous characters, making it much more capable of reading flowing-style letters in a writer's hand.

2. Contextual Understanding: The RNN's memory will help contextualize a character in its surroundings, thus bettering the number of correct recognitions of characters. Uniqueness of context in surrounding text will cause related characters to appear unique when identical in isolation.

3. Consistent Training and Validation Loss: NotesAI has a validation loss that is low and stable as the epochs go by, which indicates high generalization ability in the model; that is, it does well at new writing styles without overfitting. This is because of good preprocessing steps and due to the intrinsic capacity that RNNs have for modeling sequential data themselves.

4. Effective Training Time: Since the overall training time is almost 660 seconds, NotesAI also demonstrates good computational efficiency. This, in turn, means that retraining or fine-tuning would be relatively easy if required.

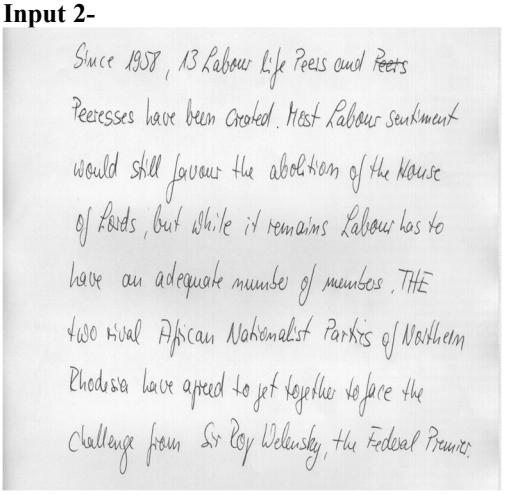
In summary, the performance of RNN-based NotesAI is way better than conventional OCR systems, considering the variability and sequential nature of handwritten text. This capability to use sequential dependencies and relations between contexts of characters provides high accuracy and robustness to the model, making it a great tool for the transformation process of handwritten notes into editable digital forms.

Input 1 -

Output 1-

```
Initial seed_tensor shape: torch.Size([1, 1, 16384])
Initial Hidden state shape: torch.Size([2, 1, 256]), torch.Size([2, 1, 256])
ABCDEFGHIKLM

NOPORSTUVINXYZ

abeclefghijkim

noparstuvwxyZz

nl2_

23456789IT
```

Input 2-



Output 2-

```
Since 958 3 Labour life Peers and Peeresses have been created Most Labour sen

timent would still favour the abolition of the House of Lords but while it remains

Labour has to have an adequate number of members THE two rival African Nation

from Sir Roy Welensky the Federal Premier
```

Inference

The training logs show that NotesAI indeed converges very quickly for both training and validation losses. The most prominent fall can be noticed in the period loss for initial epochs from 0.130 Epoch 1, which stabilizes from Epoch 4 to 0.009. The validation also behaves in the same pattern, with a significant fast stabilization at 0.009 by Epoch 2 and a pretty stable state throughout the whole process of the training.

```
Epoch: 1 | train_loss: 0.130 | val_loss: 0.013
Epoch: 2 | train_loss: 0.010 | val_loss: 0.009
Epoch: 3 | train_loss: 0.009 | val_loss: 0.009
Epoch: 4 | train_loss: 0.009 | val_loss: 0.009
Epoch: 5 | train_loss: 0.009 | val_loss: 0.009
Epoch: 6 | train_loss: 0.009 | val_loss: 0.009
Epoch: 7 | train_loss: 0.009 | val_loss: 0.009
Epoch: 8 | train_loss: 0.009 | val_loss: 0.009
Epoch: 9 | train_loss: 0.009 | val_loss: 0.009
Epoch: 10 | train_loss: 0.009 | val_loss: 0.009
Epoch: 11 | train_loss: 0.009 | val_loss: 0.009
Epoch: 12 | train_loss: 0.009 | val_loss: 0.009
Epoch: 13 | train_loss: 0.009 | val_loss: 0.009
Epoch: 14 | train_loss: 0.009 | val_loss: 0.009
Epoch: 15 | train_loss: 0.009 | val_loss: 0.009
Epoch: 16 | train_loss: 0.009 | val_loss: 0.009
Epoch: 17 | train_loss: 0.009 | val_loss: 0.009
Epoch: 18 | train_loss: 0.009 | val_loss: 0.009
Epoch: 19 | train_loss: 0.009 | val_loss: 0.009
Epoch: 20 | train_loss: 0.009 | val_loss: 0.009
Total Training Time : 660.581seconds
```

Low and stable validation loss suggests that the model generalizes well to unseen data, mostly indicating the absence of overfitting. Hence, this practically indicates that the RNN-LSTM architecture effectively learned to recognize the patterns in the handwritten text dataset, apparently because of the robust feature extraction strength of RNNs and the capability of LSTMs to capture sequential dependencies.

The total training time taken was approximately 660 seconds (~11 minutes), which showcased an efficient training process-good for scalability and practical application. The results affirm that the above-described model of NotesAI is suitable for converting handwritten text to editable formats with a high degree of accuracy and uniformity.

References:

1. *Khalkar, Rohini & Dikhit, Adarsh & Goel, Anirudh. (2021). Handwritten Text Recognition using Deep Learning (CNN & RNN). IARJSET. 8. 870-881. 10.17148/IARJSET.2021.86148.*
2. *Sherstinsky, A. (2020). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. Physica D Nonlinear Phenomena, 404, 132306. 10.1016/j.physd.2019.132306*
3. *Yuming He (2020) Research on text detection and Recognition based on OCR recognition technology. (n.d.). 10.1109/ICISCAE51034.2020.9236870*
4. *Hemanth G R, Jayasree M, Keerthi Venii S, Akshaya P, Saranya R (2021). ICTACT Journals - view articles. (n.d.). 10.21917/ijsc.2021.0351*