International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 09 Issue: 07 | July 2022 www.irjet.net p-ISSN: 2395-0072

Handwritten Text Recognition and Translation with Audio

Patibandla Vishnu Deepak[1], Vishnu Vardhan[2], Yogesh Gautam[3], Jayashree[4]

[1]VIII Semester, Dept. of CSE, BNMIT
[2]VIII Semester, Dept. of CSE BNMIT
[3]Asst. Professor, Dept of CSE, BNMIT, Karnataka, India

Abstract - *This project attempts to classify any unique handwritten text in order to convert the text material to a digital format. To complete this objective, we used two different methods: direct word classification and separating each character. Using a Convolutional Neural Network (CNN) with different architectures, we train a model that can accurately categorize words for the former. To create boxes for each particular character, we employ a special LSTM with convolution. Where LSTM is Long Short-Term Memory. These separated characters are then forwarded into a CNN for categorizing, and each particular word is subsequently reassembled using the classification and separation findings.*

Key Words: Optical character recognition, Convolutional Networks, Text detection and recognition, translation, audio speech.

1.INTRODUCTION

Despite the prevalence of electronic writing devices, many people still choose taking notes on paper and a pen. Though, there are certain advantages there are specific disadvantages to writing text by hand and are provided below. It's tough to efficiently keep and access physical documents, as well as to efficiently search, sort them from a huge pile of documents. Sharing these documents can also stressful.

Consequently, lots of valuable information is misplaced or never analysed since these documents are never converted to digital format. We chose to address this issue in our research because we believe that the far better ease of managing digital text compared to written language will enable people to more easily access, search, share, and analyse their records while still allowing them to use their preferred writing method. The purpose of this project is to delve deeper into the task of classifying handwritten text and converting it to a digital representation. Handwritten text is a broad phrase, and we intended to reduce the scope of the project by defining handwritten writing specifically for our needs. We took on the task of identifying the visual of any handwritten word, whether it was in cursive or block script, for this research. This work can be combined with algorithms that separate words from lines in a given line image, which can then be used with algorithms that separate lines from lines in a given image of a whole handwritten page. With these classes added, our project could be in the form of a distributable file used by the end user and would be a fully functional template that would prompt the user to take a picture of the notes page, allowing users solve the problem of converting handwritten documents to digital format. Note that while some extra layers need to be added to our model to create a fully functional deliverable product to end users, we believe the categorization of the problem is interesting. the most delicious and the hardest, that's why we chose to approach it. instead of line-to-word, document-to-line, etc.

Because CNNs function better on raw input pixels data rather than specific features or sections of an particular image, we can address this problem with complete word pictures. We expect the improvement by extracting characters from each particular word image and then classifying each word character individually to re-construct a whole word, based on our findings using entire word images. In summary, our models in both ways take in an image/picture of the word and then output the word's name.

2. RELATED WORK

2.1 Early Scanners

The first motivation for the classification of handwritten texts was the digit classification of mail. Early postal readers, such as Jacob Rabinow's, used integrated scanning and processing devices to recognize fixed-spaced fonts. Allum et al. He refined this by developing a sophisticated scanner that allowed more variation in the way text was written and put the data into a barcode that was printed directly on the letter.in text input and that also encoded the information onto a bar-code that was printed directly on the letter.

2.2 To the Digital Age

In 1974, Ray Kurzweil developed the first known piece of OCR software that could identify any typeface. In this software, the matrix technique was used more frequently (pattern matching). In essence, this would determine which character it most closely resembled by comparing the bitmaps of the template character with the read character. The drawback of this programme was that it was sensitive to variations in font size and writing style. To improve templating, OCR software switched from using templates to feature extraction. Software would look for each character's geometric moments, zoning, and projection histograms.

2.3 Machine Learning

The Lecun concentrated on utilizing gradient-based learning approaches with multi-module ML models, which served as the forerunner for the first end-to-end modern deep learning models.

Adoption of a Hidden Markov Model for OCR was the next big advancement in attaining high OCR accuracy. This method uses each letter as a specific state, and then considering the context of the character when choosing the next hidden variable. In comparison to feature extraction technique and the Naive Bayes approach, this resulted in higher accuracy. The manual extraction elements remained the biggest drawback, since they required prior language expertise and were not especially robust to the diversity and complexity of handwriting.

Ng and colleagues used a sliding window to apply CNNs to the issue of identifying text that is observed in the real world (signs, written, etc.) within an image. A sliding window scans the picture to see if any characters might be present. Each character was classified using a CNN with two of each convolutional layers, and average pooling layers lastly a fully connected layer.

Scan, and Read: End-to-End MDLSTM Handwritten Paragraph Recognition One of the most renowned publications for the task of handwritten text detection is Attention. An LSTM layer was utilised to encode the raw image data to a feature map for each scanning direction.

After then, the model would use attention to give particular feature maps more priority than others. The attention map was created, then it was given to the decoder, which used the visual summary and state to anticipate the character. This approach was distinctive in that it combined the segmentation and classification processes into a single model rather than separating them. Due to the lack of a language model for producing character and word sequences, this approach has one drawback. Without considering the context of the invented word, it solely relies on each character's aesthetic categorization.

We also found that using a project from CS 231N in the past helped us with our job. Yan uses the Faster R- CNN model to categorize and identify specific characters within a word.. This method initially determines whether an object exists within the image's bounds by dragging a window across it. The bounded image is then classified to the character that corresponds to it. Yan also uses edit distance, which lets users to make changes to the separated word to see if another separated word (for example, xoo versus zoo) is more likely to be correct.

3. DATA

The IAM Handwriting Dataset was our primary source for handwriting recognition training. The handwritten text in this dataset totals 5500 phrases and 11500 words, and it was written on more than 1500 forms (a form is a sheet of paper with lines of text) by more than 600 authors. The words were segmented and meticulously checked before being delivered with all relevant form label metadata in associated XML files. The source material was taken from the Lancaster-Oslo/Bergen (LOB) corpus, which comprises over 1 million words of entire English sentences. There are another 1,066 forms in the database, written by roughly 400 distinct authors.

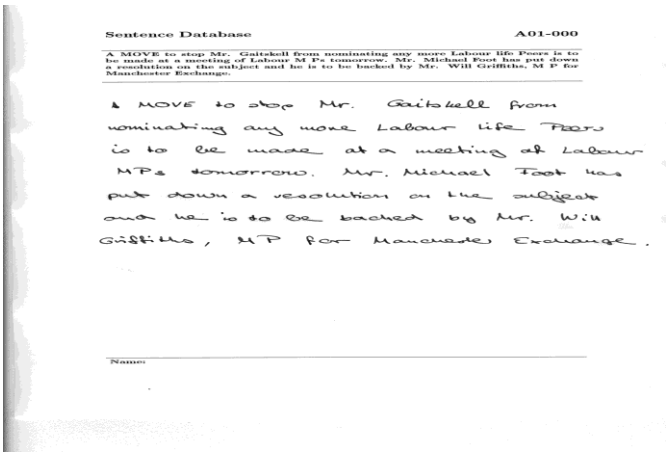


Fig -1: a form from the IAM Handwriting dataset as an example. The dataset's word pictures were taken from such formats.

Because of its breadth, depth, and quality, this database is often used as the foundation for many handwriting recognition tasks, this is the reason for choosing the IAM Handwriting Dataset as the foundation of our training, validation, and test data for our models. Lastly, large datasets—even those that contain multiple pre-trained models—are essential for deep learning, and this dataset, which contained over 100K word instances, satisfied those requirements. Deep learning models need at best 105 -106 training examples to work properly, regardless of transfer learning.

4. PRE-PROCESSING STAGE

Pre-processing is a sequence of procedures applied to the captured image. In essence, it improves the depiction of the image to make it segmentation-ready. A lot of procedures are carried out on an image as part of its pre-processing as given below.

4.1 Noise Removal

This pre-processing stage raises the quality of the raw image and increases the data relevance. This stage is also known as pixel level or low-level processing. Skew correction of an image is carried out during the pre-processing stage to accurately display the article text lines, threshold to convert a colour or grayscale image into a binary image, and noise reduction to reduce superfluous data, as illustrated in Figure 2.

Dust, spots, dots, and lines are all examples of noise that can contaminate scanned documents and significantly alter recognition outcomes. A scanned article image needs to be cleaned of any noise in order to make it suitable for further processing. To enhance the image that a machine or person is viewing, image enhancement techniques are used.

Noise reduction uses smoothing and nonlinear processes like morphological operations.

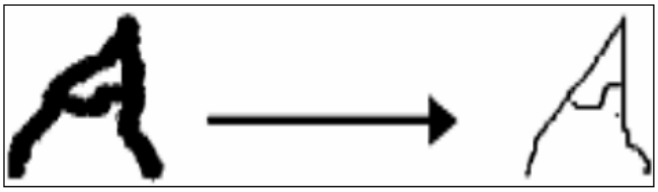


Fig -2: Results from the pre-processing phase, including noise removal and skew correction.

4.2 Skew Correction

If the text line in scanned document images is not horizontally aligned after scanning, skew correction is employed to fix it. On the document or line level, the skew correction can be implemented. A global skew correction is implemented on the document level during the pre-processing stage.

4.3 Slant Correction

To adjust the writing style's inclination, slant correction is used. The writings are altered from their slant to their upright state by means of a shear transformation. Numerous directions are affected by a shear transformation. The pixel values of the same image that have been vertically moved by distances d and -d are added to the converted image data for each direction, as illustrated in Figure 3.

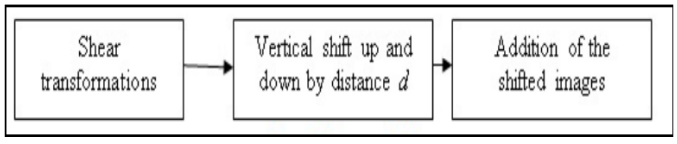


Fig -3: Steps in Slant correction.

4.4 Line Segmentation

An image with an alphabetic sequence is broken down into smaller images that represent individual characters during this segmentation stage of pre-processing. By giving each alphabet a number via a labelling procedure, the pre-processed input image is separated into distinct alphabets. The labelling indicates how many characters are there in the image. For the categorization and identification stage, every single character is uniformly reconstructed into 90*60 pixels.

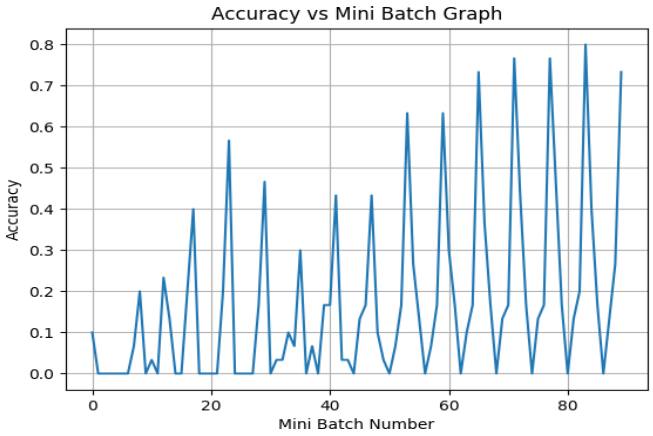
5. METHODOLOGY

5.1 Training Level

Following SoftMax activation, we trained each of our models on a cross-entropy loss function using the ADAM Optimizer. With the exception of changing the last completely connected output layer to map to the number of classes (word/character vocabulary), we kept the identical architectures for the Visual Geometry Group (VGG) and Residual Networks (RESNET) models. We started from scratch while training the word and character classifier. We reasoned that the enormous SoftMax layer output size (more than 10,000 words in our training vocabulary and well over 100,000 words in only the English language) and the difficulty of fine-grained word picture identification were impacting our performance for word-level classification. We came to the conclusion that character-level classification might be a more effective strategy because the SoftMax's computational complexity can be greatly reduced by using a character vocabulary that is significantly less complex than a word vocabulary (the characters A-Za-z0-9 only have 62 distinct symbols). In addition, due to the constrained character set, character identification in an image is an easier challenge than word recognition in an image. The separation of word specific images into their component character images will be the first significant obstacle we would have to overcome in order to evaluate this strategy. The second major difficulty would be identifying word breaks in images and connecting consecutively identified letters in between these word breaks to make words. In this section, we'll focus on the first of these problems. We used the brand-new CNN/LSTM engine powered by Tesseract 4.0 neural networks to carry out this task. This model is set up as a text line recognizer that can automatically identify more than 100 languages. It was originally developed by HP and is now maintained by Google. The Tesseract model had been trained on roughly 400000 text lines spread among about 4500 fonts for Latin-based languages, including English. On our IAM handwriting dataset, we then adjusted this pretrained model's parameters. The original word input images were split into their hypothetical component character images after the model was tweaked, and these output segmented character images were fed into our character-level categorization system.

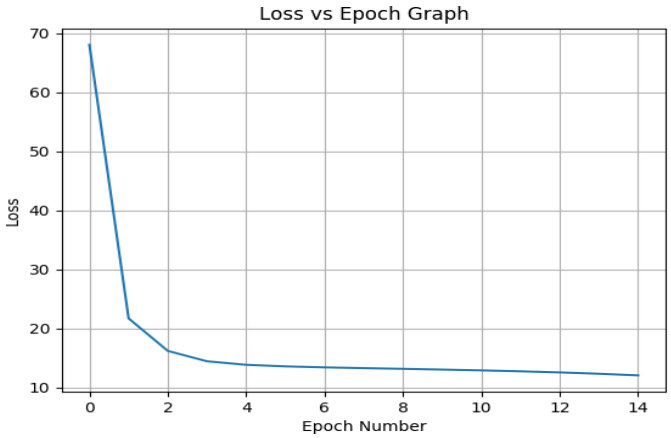
5.2 Each Word-Level Classification

We started using VGG-19 to train our Word-level classification model. First off, considering the quantity of parameters it needs, we discovered that this model is difficult to train. Initially, we tried using the whole vocabulary, but we discovered that the run time was too slow. We limited the vocabulary to 50 terms that were chosen at random and showed up at least 20 times in our dataset in order to get results. Because some of the chosen words, like "the," contained hundreds of training examples, training was still quite slow. In order to have enough data to benchmark outcomes with a better trade-off on the run time, we limited each word to 20 occurrences. Our VGG-19 model was subsequently trimmed, and training with numerous parameters was started. Our learning rate was too high during our first runs, which led to low end training and validation accuracy. We increased the learning rate of our Adam Optimizer, which achieved training accuracy of 28% and validation accuracy of 22%.



Graph -1: When we first started training, Our learning rate was set too quickly, which resulted in inconsistent accuracy.

We looked for alternative architectures after realizing that VGG-19 was fairly slow due to the quantity of parameters it needs. We discovered CNN-benchmarks that suggested RESNETi-18 could deliver comparable results at substantially faster speeds. Furthermore, we also thought that our model's ability to train the residual layer of the RESNET architecture would improve our accuracy, especially with deeper networks, as our model would be modified or taught to achieve optimality for the residual layer.



Graph -2: We found that our loss was steadily decreasing after modifying our learning rate and model input.

We have discovered that limiting the number of epochs used when experimenting with various parameters is a good method. We trained our model on two epochs, compared the results, and then reran it on more epochs with a subset of parameters that produced the best results and those whose loss/accuracy graphs looked the most promising because our training and validation processes took a long time and we wanted to find the best parameters. In terms of training, this method undoubtedly saved us a ton of time, but it also had drawbacks. Some parameters, for example, might have taken longer to converge but eventually attained their best value in the end. If their graphs did not appear sufficiently promising and their findings were subpar when compared to the other parameters, we might have overlooked such parameters. Due to a lack of resources, this presented a hurdle for us, but we made an effort to overcome it by taking into account the parameter graphs in addition to the raw data they supplied.

Because only one class dominated the weight updates at the beginning of our training efforts and at the end, all of the instances were assigned to that class, the validation accuracy was initially practically random. We discovered that the problem was in fact with the weights and that the updates were too large, saturating weight values at the beginning and inhibiting learning after printing the weights and their changes for two samples.

Table -1: Results of Classification at the Word and Character Levels.

Architecture	Training Accuracy	Validation Accuracy	Test Accuracy
VGG-19	28%	22%	20%
RESNET-18	31%	23%	22%
RESNET-34	35%	27%	25%
Char-Level	38%	33%	31%

RESNET-34 was implemented to deepen the network in response to our RESNET-18 results, which we used in conjunction with our model. The accuracy we achieved during training was 35%, while accuracy during validation was 27%. In order to avoid having parameter updates occur too frequently and slow down our model, we decided to utilize a reasonably large mini-batch size of 50. A bigger mini-batch size would collect too much data all at once to indicate the gradient's direction. Due to the need for semi-regular updates and the avoidance of the gradient's value being distorted, this was crucial for our model.

Qualitatively, the findings of word classification are encouraging; they demonstrate that, despite data constraints, we can still obtain respectable accuracy for a highly complex issue. We are aware, however, that larger vocabulary sizes will make it more difficult to gather the essential data and will drastically impair accuracy, undermining the capacity to effectively convert handwritten text to a digital format. This motivated us to start looking into the character segmentation technique.

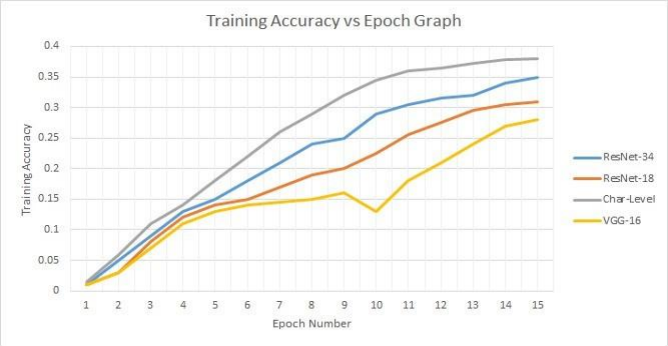


Fig -4: When we first started training, Our learning rate was set too quickly, which resulted in inconsistent accuracy.

5.3 Character Classification Level

We then move on to letter segmentation and word reconstruction by classifying each character separately to improve the direct word classification results. The Tesseract LSTM model with convolution was firstly downloaded and pre-trained on the aforesaid English datasets before it being refined onto our dataset. We then had to modify each of our input data so that it included not only input image, but also bounding box labels of each particular character in that image and the closest approximation in the four top left, bottom top left, top right, and top left of each personality character, which we extracted from the XML data.

Given the pertinent data, the Tesseract contains programs to automatically construct this dataset. Following that, we created a character vocabulary made up of single-digit numerals, uppercase, and lowercase letters. Adam, the optimization technique we ultimately used in our final model was also used, but we used a different kind of loss that was more appropriate for the issue: CTC loss. In a nutshell, CTC (Connectionist Temporal Classification) is a technique/loss function created by Graves et al. for teaching recurrent neural networks to label output sequences (labels for characters in our example) from unsegmented input data the input word picture in our case. RNNs with CTC loss have been shown to be more efficient than more conventional methods like CRFs (Conditional Random Fields) and HMMs for tasks like labelling speech signal data with word-level transcriptions because of their automated learning process, need for only an input/output data representation as opposed to substantial hand-engineered features, and ability to more generally capture context over time and space in their hidden state. CTC can separate and label each of the unstructured input data available by assessing network outputs as a probability distribution over all possible label sequences, conditioned on a given input sequence" created by considering segmentation of the input sequence given a certain minimum segment size and starting training of the network from there. We provided these final segmented character photos to our model after honing and completing our segmentation model, as previously described. With the same model designs for classification and even with the possibility of segmentation error, we discovered that character-level classification was more successful, as seen in the accuracy graph for character vs. word-level classification. These findings corroborated our hypotheses that the performance was improved by the model's initial feature representation problem, which for characters was significantly less in scope than for words, and its final labelling problem. Due to a lack of data that was enough for the extent of our issue and flaws in the segmentation model, we believed that our model did not perform any better.

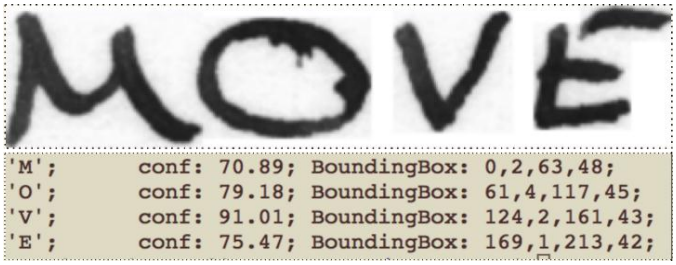


Fig -5: For each word, we can segment each character using Tesseract LSTM. For example, we can extract the start and end coordinates of each character in the word "MOVE".

Due to the breakdown in the borders between some cursive letters, our model, like most segmentation models, has difficulty segmenting cursive characters. In fact, because our segmentation model and character classification model were trained independently, we were unable to precisely pinpoint whether our classification model or segmentation model was to blame for the inaccuracy. Last but not least, writers' handwriting varies greatly, making it difficult to distinguish between all of the many ways that a word or character is

written. We reasoned that, as is frequently the case with deep learning challenges, even additional data (over the course of millions more words) would have assisted in teaching our model a more generalized feature representation for the issue, which would have improved its performance.

3. CONCLUSIONS

Deep learning has gained popularity in recent years and has started to be used in a variety of fields, including facial recognition, target identification, and optical character recognition (OCR). Target detection is consistently used by researchers. Networks of various types, from the Faster and RCNN OCR area, used to find texts. in comparison to text recognition in the classical sense by embedded. With consistent noise immunity, deep learning is more accurate and durability. It has the capacity to fend off effects like alterations in the backdrop. The deep learning end-to-end network is more precise when applied to text detection, when compared to customary picture processing and cutting. Machine learning enables character classification. Apart from the text detection and conversion it into digital format this application also provides with translating the text into other languages mainly focusing on English and Hindi.

ACKNOWLEDGEMENT

Vishnu Deepak Patibandla, Vishnu Vardhan and Yogesh Gautam are grateful to Mrs. Jayashree, Assistant Professor, BNMIT for her constant support and technical guidance over the course this project. Also, Vishnu Deepak, Vishnu Vardhan and Yogesh Gautam would like express our immense gratitude towards Mrs. Chayadevi, M L HOD at BNMIT for her constant support.

REFERENCES

[1] A Novel Approach to Classify English Handwritten Character Using Convolutional Neural Networks by Rupesh Sharma, Prashant Sharma, Naresh Purohit, International Journal of Advanced Computer Science and Applications, 21-23 Mar. 2021, West Yorkshire, Malayasia.

[2] Handwritten character recognition using Convolutional neural networks by Verdha Chaudhary, Mahak Bansal and Guarav Raj, IEEE Seventh International Conference on Intelligent Systems Design and Applications, 09 Dec. 2020, Rio de Janeiro, Brazil.

[3] CNN Based Common Approach to handwritten character recognition of multiple scripts by Durjoy Sen Maitra, Ujjwal Bhattacharya and Swapan K. Paru, IEEE Proceedings of 2nd International Conference on Document Analysis and Recognition, 20-22 Oct. 2020, Tsukuba, Japan.

[4] Handwritten Character Recognition from Images using CNN-ECOC by Mayur Bhargab Bora, Dinthisrang Daimary, Khwairakpam Amitab, Debdatta Kandar., International Conference on Innovative Practices in Technology and Management, 17-19 Feb. 2020, Noida, India.

[5] Handwritten character recognition using CNN by S. Anandh kishan, J. Clinton David, P. Sharon Femi, IEEE Transactions on Pattern Analysis and Machine Intelligence, 05 May. 2020, Melbourne, Australia.

[6] Transfer learning using CNN for handwritten Devanagari Character Recognition by Nagender Aneja and Sandhya Aneja., International Journal of Engineering and Computer Science, 06-09 Feb. 2020, Delhi, India.

[7] Handwritten Character Recognition of Devanagari Script by Breijeshwar Dessai Amit Patel, IEEE Proceedings of 2nd International Conference on Document Analysis and Recognition, 20-22 Oct. 2019, Tokyo, Japan.

[8] An End-to-End System for Hindi Online Handwriting Recognition, by Soumik Bhattacharya, DurjoySen Maitra, Ujjwal Bhattacharya and Swapan K. Parui, IEEE third International Conference, 2016.

[9] Automatic image-to-Text-to-Voice Conversion for Interactively Locating Objects in Home Environments, by Nikolas Bourbk, 20th IEEE Interactive conference on Tools with ArtificialIntelligence,2008,1082-3409/08.

[10] Review on conversion of image to text as well as speech using edge detection and image segmentation by Mrunmayee patil, Ramesh kagalkar, international journal of science and research (IJSR),2319-7064,2012,3.358.