An Efficient End-to-End Neural Model for Handwritten Text Recognition

2 authors, including:

Lovekesh Vig
Tata Consultancy Services Limited
94 PUBLICATIONS **1,147** CITATIONS

Some of the authors of this publication are also working on these related projects:

Prognostics View project

Time Series Anomaly Detection View project

An Efficient End-to-End Neural Model for Handwritten Text Recognition

Arindam Chowdhury
chowdhury.arindam1@tcs.com

Lovekesh Vig
lovekesh.vig@tcs.com

TCS Innovation Labs
Delhi

Abstract

Offline handwritten text recognition from images is an important problem for enterprises attempting to digitize large volumes of handmarked scanned documents/reports. Deep recurrent models such as Multi-dimensional LSTMs [11] have been shown to yield superior performance over traditional Hidden Markov Model based approaches that suffer from the Markov assumption and therefore lack the representational power of RNNs. In this paper we introduce a novel approach that combines a deep convolutional network with a recurrent Encoder-Decoder network to map an image to a sequence of characters corresponding to the text present in the image. The entire model is trained end-to-end using Focal Loss [18], an improvement over the standard Cross-Entropy loss that addresses the class imbalance problem, inherent to text recognition. To enhance the decoding capacity of the model, Beam Search algorithm is employed which searches for the best sequence out of a set of hypotheses based on a joint distribution of individual characters. Our model takes as input a downsampled version of the original image thereby making it both computationally and memory efficient. The experimental results were benchmarked against two publicly available datasets, IAM and RIMES. We surpass the state-of-the-art word level accuracy on the evaluation set of both datasets by 3.5% & 1.1%, respectively.

1 Introduction

Handwritten Text Recognition (HTR) has been a major research problem for several decades [7] [28] and has gained recent impetus due to the potential value that can be unlocked from extracting the data stored in handwritten documents and exploiting it via modern AI systems. Traditionally, HTR is divided into two categories: offline and online recognition. In this paper, we consider the offline recognition problem which is considerably more challenging as, unlike the online mode which exploits attributes like stroke information and trajectory in addition to the text image, offline mode has only the image available for feature extraction.

Historically, HTR has been formulated as a sequence matching problem: a sequence of features extracted from the input data is matched to an output sequence composed of characters from the text, primarily using *Hidden Markov Models* (HMM) [9][21]. However, HMMs fail to make use of the context information in a text sequence, due to the Markovian assumption that each observation depends only on the current state. This limitation was addressed by the use of *Recurrent Neural Networks* (RNN) which encode the context information in the hidden states. Nevertheless, the use of RNN was limited to scenarios in which

the individual characters in a sequence could be segmented, as the RNN objective functions require a separate training signal at each timestep. Improvements were proposed in form of models that have a hybrid architecture combining HMM with RNN [6] [4], but major breakthrough came in [12] which proposed the use of *Connectionist Temporal Classification* (CTC) [10] in combination with RNN. CTC allows the network to map the input sequence directly to a sequence of output labels, thereby doing away with the need of segmented input.

The performance of RNN-CTC model was still limited as it used handcrafted features from the image to construct the input sequence to the RNN. *Multi-Dimensional Recurrent Neural Network* (MDRNN) [11] was proposed as the first end-to-end model for HTR. It uses a hierarchy of multi-dimensional RNN layers that process the input text image along both axes thereby learning long term dependencies in both directions. The idea is to capture the spatial structure of the characters along the vertical axis while encoding the sequence information along the horizontal axis. Such a formulation is computationally expensive as compared to standard convolution operations which extract the same visual features as shown in [24], which proposed a composite architecture that combines a *Convolutional Neural Network* (CNN) with a deep one-dimensional RNN-CTC model and holds the current state-of-the-art performance on standard HTR benchmarks.

In this paper, we propose an alternative approach which combines a convolutional network as a feature extractor with two recurrent networks on top for sequence matching. We use the RNN based *Encoder-Decoder* network [8] [27], that essentially performs the task of generating a target sequence from a source sequence and has been extensively employed for Neural Machine Translation (NMT). Our model incorporates a set of improvements in architecture, training and inference process in the form of Batch & Layer Normalization, Focal Loss and Beam Search to name a few. Random distortions were introduced in the inputs as a regularizing step while training. Particularly, we make the following key contributions:

- We present an end-to-end neural network architecture composed of convolutional and recurrent networks to perform efficient offline HTR on images of text lines.

- We demonstrate that the Encoder-Decoder network with *Attention* provides significant boost in accuracy as compared to the standard RNN-CTC formulation for HTR.

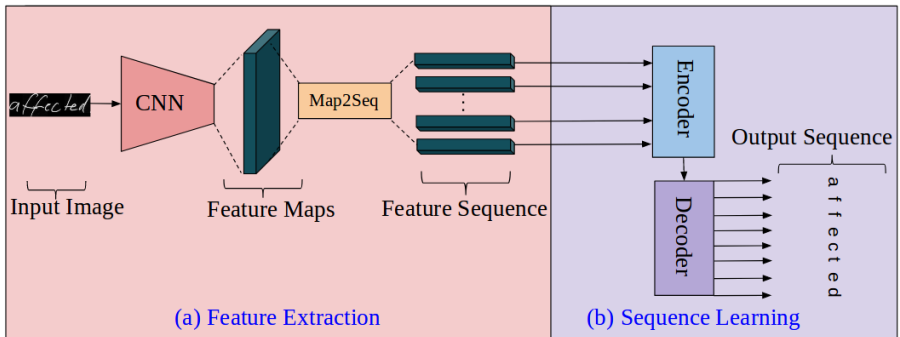(a) Feature Extraction   (b) Sequence Learning

Figure 1: Model Overview: (a) represents the generation of feature sequence from convolutional feature maps and (b) shows the mapping of the visual feature sequence to a string of output characters.

- We show that a reduction of 61% in computations and 17% in memory consumption can be achieved by downsampling the input images to almost a sixteenth of the original size, without compromising with the overall accuracy of the model.

2 Proposed Method

Our model is composed of two connectionist components: (*a*) *Feature Extraction* module that takes as input an image of a line of text to extract visual features and (*b*) *Sequence Learning* module that maps the visual features to a sequence of characters. A general overview of the model is shown in Figure 1. It consists of differentiable neural modules with a seamless interface, allowing fast and efficient end-to-end training.

2.1 Feature Extraction

Convolutional Networks have proven to be quite effective in extracting rich visual features from images, by automatically learning a set of non-linear transformations, essential for a given task. Our aim was to generate a sequence of features which would encode local attributes in the image while preserving the spatial organization of the objects in it. Towards this end, we use a standard CNN (without the fully-connected layers) to transform the input image into a dense stack of feature maps. A specially designed *Map-to-Sequence* [26] layer is put on top of the CNN to convert the feature maps into a sequence of feature vectors, by depth-wise detaching columns from it. It means that the *i*-th feature vector is constructed by concatenating the *i*-th columns of all the feature maps. Due to the translational invariance of convolution operations, each column represents a vertical strip in the image (termed as *Receptive field*), moving from left to right, as shown in Figure 2. Before feeding to the network, all the images are scaled to a fixed height while the width is scaled maintaining the aspect ratio of the image. This ensures that all the vectors in the feature sequence conform to the same dimensionality without putting any restriction on the sequence length.
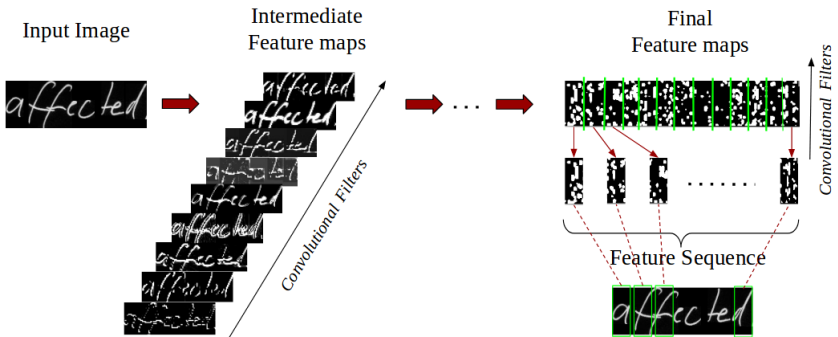

Figure 2: Visualization of feature sequence generation process and the possible Receptive fields of the feature vectors. Intermediate feature maps are stacked depth-wise in correspondence with the convolutional filters that generate them while the final feature maps are stacked row-wise.

2.2 Sequence Learning

The visual feature sequence extracted by the CNN is used to generate a target sequence composed of character tokens corresponding to the text present in the image. Our main aim, therefore, was to map a variable length input sequence into another variable length output sequence by learning a suitable relationship between them. In the Encoder-Decoder framework, the model consists of two recurrent networks, one of which constructs a compact representation based on its understanding of the input sequence while the other uses the same representation to generate the corresponding output sequence.

The encoder takes as input, the source sequence $\mathbf{x} = (x_1, \ldots, x_{T_s})$, where T_s is the sequence length, and generates a context vector c, representative of the entire sequence. This is achieved by using an RNN such that, at each timestep t, the hidden state $h_t = g(x_t, h_{t-1})$ and finally, $c = s(h_1, \ldots, h_{T_s})$, where g and s are some non-linear functions. Such a formulation using a basic RNN cell is quite simple yet proves to be ineffective while learning even slightly long sequences due to the vanishing gradient effect [14][5] caused by repeated multiplications of gradients in an unfolded RNN. Instead, we use the *Long Short Term Memory* (LSTM)[13] cells, for their ability to better model and learn long-term dependencies due to the presence of a memory cell $c \in \mathbb{R}^n$. The final cell state c_{T_s} is used as the context vector of the input sequence. In spite of its enhanced memory capacity, LSTM cells are unidirectional and can only learn past context. To utilize both forward and backward dependencies in the input sequence, we make the encoder *bidirectional* [25] , by combining two LSTM cells, which process the sequence in opposite directions, as shown in Figure 3. The output of the two cells, forward $\overrightarrow{h}_t$ and backward $\overleftarrow{h}_t$ are concatenated at each timestep, to generate a single output vector $h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t]$. Similarly, final cell state is formed by concatenating the final forward and backward states $c_{T_s} = [\overrightarrow{c}_{T_s}; \overleftarrow{c}_{T_s}]$.
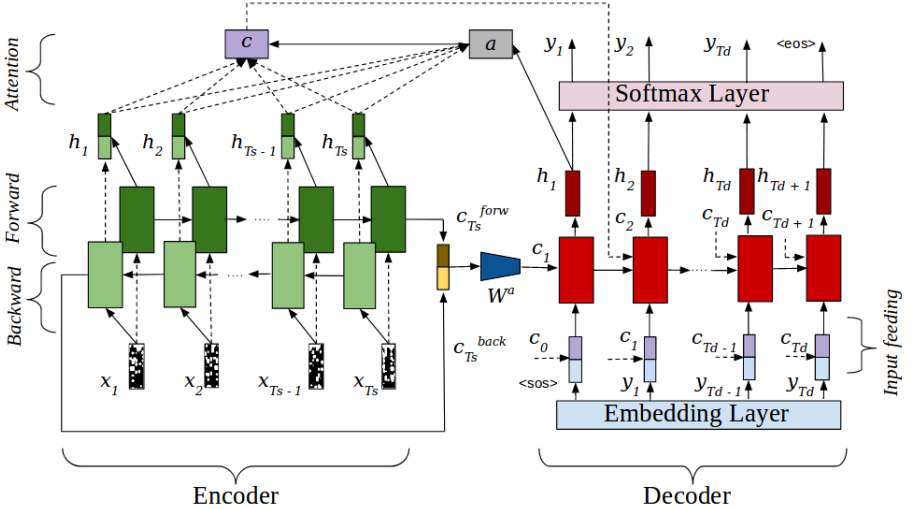

Figure 3: RNN *Encoder-Decoder* Network with Attention layer. Encoder is a bidirectional LSTM whose outputs are concatenated at each step while decoder is an unidirectional LSTM with a *Softmax* layer on top. The character inputs are sampled from an embedding layer.

The context vector c_{T_s} is fed to a second recurrent network, called decoder which is used

to generate the target sequence. Following an affine transformation, $c_1 = W^a c_{T_s}$, where W^a is the transformation matrix, c_1 is used to initialize the cell state of the decoder. Unlike the encoder, decoder is unidirectional as its purpose is to generate, at each timestep t, a token y_t of the target sequence, conditioned on c_1 and its own previous predictions $\{y_1, \ldots, y_{t-1}\}$. Basically, it learns a conditional probability distribution $p(\mathbf{y}) = \prod_{t=1}^{T_d} p(y_t | \{y_1, \ldots, y_{t-1}\}, c_1)$ over the target sequence $\mathbf{y} = \{y_1, \ldots, y_{T_d}\}$, where T_d is the sequence length. Using an RNN, each conditional is modeled as $p(y_t | \{y_1, \ldots, y_{t-1}\}, c_1) = Softmax(g(y_{t-1}, h_{t-1}, c_1))$, where g is a non-linear function and h_{t-1} is the RNN hidden state. As in case of the encoder, we employ an LSTM cell to implement g.

The above framework proves to be quite efficient in learning a *sequence-to-sequence* mapping but suffers from a major drawback nonetheless. The context vector that forms a link between the encoder and the decoder often becomes an information bottleneck[8]. Especially for long sequences, the context vector tends to forget essential information that it saw in the first few timesteps. *Attention* models are an extension to the standard encoder-decoder framework in which the context vector is modified at each timestep based on the similarity of the previous decoder hidden state $h_{t-1}^{decoder}$ with the sequence of annotations $\{h_1^{encoder}, \ldots, h_{T_s}^{encoder}\}$ generated by the encoder, for a given input sequence. As we use a bidirectional encoder, Bahdanau [3] attention mechanism becomes a natural choice for our model. The context vector at the i-th decoder timestep is given by,

$$c_i = \Sigma_{j=1}^{T_s} \alpha_{ij} h_j^{encoder}$$

The weight α_{ij} for each $h_j^{encoder}$ is given as,

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\Sigma_{k=1}^{T_s} \exp(e_{ik})}$$

$$where, \quad e_{ij} = a(h_{i-1}^{decoder}, h_j^{encoder})$$

Here, a is a feedforward network trained along with the other components.

Therefore, the context vector is modified as an weighted sum of the input annotations, where the weights measure how similar the output at position i is with the input around position j. Such a formulation helps the decoder to learn local correspondence between the input and output sequences in tandem with a global context, which becomes especially useful in case of longer sequences. Additionally, we incorporate the attention *input feeding* approach used in Luong [19] attention mechanism in which the context vector from previous timestep is concatenated with the input of the current timestep. It helps in building a local context, further augmenting the predictive capacity of the network.

We train the model by minimizing a cumulative *categorical cross-entropy* (CE) loss calculated independently for each token in a sequence and then summed up. For a target sequence $\mathbf{y} = \{y_1, \ldots, y_{T_d}\}$, the loss is defined as $CE(\mathbf{y}) = -\Sigma_{t=1}^{T_d} \log(p(y_t))$ where $p(y_t)$ is the probability of true class at timestep t. The input to the decoder at each timestep is an embedding vector, from a learnable embedding layer, corresponding to the gold prediction from previous step, until the *end-of-sequence* or <eos> token is emitted. At this point, a step of gradient descent is performed across the recurrent network using *Back Propagation Through Time* (BPTT) followed by back propagation into the CNN to update the network parameters.

Although CE loss is a powerful measure of network performance in a complex multi-class classification scenario, it often suffers from class imbalance problem. In such a situation, the CE loss is mostly composed of the easily classified examples which dominate

the gradient. *Focal Loss* [13] addresses this problem by assigning suitable weights to the contribution of each instance in the final loss. It is defined as $FL(p) = -(1-p)^{\gamma}\log(p)$, where p is the true-class probability and γ is a tunable focusing parameter. Such a formulation ensures that the easily classified examples get smaller weights than the hard examples in the final loss, thereby making larger updates for the hard examples. Our primary motivation to use focal loss arises from the fact that, in every language, some characters in the alphabet have higher chances of occurring in regular text than the rest. For example, vowels occur with a higher frequency in English text than a character like z. Therefore, to make our model robust to such an inherent imbalance, we formulate our sequence loss as $FL(\mathbf{y}) = -\Sigma_{t=1}^{T_d}(1-p(y_t))^{\gamma}\log(p(y_t))$. We found that $\gamma = 2$ worked best for our model.

To speed up training, we employ mini-batch gradient descent. Here, we optimize a batch loss which is a straightforward extension of the sequence loss, calculated as

$$L = -\frac{1}{M}\Sigma_{i=1}^{M}\Sigma_{t=1}^{T_d}(1-p(y_{it}))^{\gamma}\log(p(y_{it}))$$

where M is the batch size and y_{it} represents the t-th timestep of the i-th instance of the batch.

For any sequence model, the simplest approach for inference is to perform a *Greedy Decoding* (GD) which emits, at each timestep, the class with the highest probability from the softmax distribution, as the output at that instance. GD operates with the underlying assumption that the best sequence is composed of the most likely tokens at each timestep, which may not necessarily be true. A more refined decoding algorithm is the *Beam Search* which aims to find the best sequence by maximizing the joint distribution,

$$p(y_1, y_2, \ldots, y_{T_d}) = p(y_1) \times p(y_2|y_1) \times p(y_3|\{y_1, y_2\}) \times \cdots \times p(y_{T_d}|\{y_1, y_2, \ldots, y_{T_d-1}\})$$

over a set of hypotheses, known as the beam. The algorithm selects top-K classes, where K is the beam size, at the first timestep and obtains an output distribution individually for each of them at the next timestep. Out of the $K \times N$ hypotheses, where N is the output vocabulary size, the top-K are chosen based on the product $p(y_1) \times p(y_2|y_1)$. This process is repeated till all the K rays in the beam emit the \<eos\> token. The final output of the decoder is the ray having the highest value of $p(y_1, y_2, \ldots, y_{T_d})$ in the beam.

3 Implementation Details

3.1 Image Preprocessing

The input to our system are images that contain a line of handwritten text which may or may not be a complete sentence. The images have a single channel with 256 intensity levels. We invert the images prior to training so that the foreground is composed of higher intensity on a dark background, making it slightly easier for the CNN activations to learn. We also scale down the input images from an average height of 128 pixels to 32 pixels while the width is scaled maintaining the aspect ratio of the original image to reduce computations and memory requirements as shown in Table 2. As we employ minibatch training, uniformity in dimensions is maintained in a batch by padding the images with background pixels on both left and right to match the width of the widest image in the batch. In preliminary experiments, our model had shown a tendency to overfit on the training data. To prevent such an outcome, as a further regularization step, we introduced random distortions [24] in the training images, so that, ideally, in every iteration the model would process a previously unseen set of inputs.

Table 1: Network configuration

Configuration	Layers
	$1-2-3-4-5-6-7$
Conv. filters	16 32 64 64 128 128 128
Maxpool(2x2)	✓- ✓- ✗- ✗- ✗- ✗- ✗
Maxpool(2x1)	✗- ✗- ✗- ✗- ✓- ✓- ✗

Table 2: Effect of Image Downsampling on Model Performance while Training

Size	Computations (Tflops)	Memory (GB)
$128 \times W$	1.5×10^4	9.5
$32 \times W'$	5.9×10^3	7.9

Every training batch is subjected to a set of four operations viz. translation, rotation, shear and scaling. Parameters for all the operations are sampled independently from a Gaussian distribution. The operations and the underlying distribution were chosen by observing a few examples at the beginning of experimentations and were fixed then on.

3.2 Convolutional Network

Our model consists of seven convolutional (conv) layers stacked serially, with *Leaky ReLU* [20] activations. The first six layers use a kernel size of 3x3 pixels with 1 pixel wide input padding while the final layer uses a kernel size of 2x2 pixels without input padding. Kernel strides are of 1 pixel in both vertical and horizontal directions. Activations of the conv layers are *Batch Normalized* [15], to prevent internal covariate shift and thereby speed up training, before propagating to the next layer. Pooling operations are performed on the activations of certain conv layers to reduce the dimensionality of the input. A total of four max-pooling layers are used in our model, two of which have a kernel size of 2x1 to preserve the horizontal spatial distribution of text and the rest use standard 2x2 non-overlapping kernels. Table 1 shows the network configuration used in each conv layer.

3.3 RNN Encoder-Decoder

Encoder & decoder use LSTM cells with 256 hidden units. We allow both networks to extend to a depth of 2 layers to enhance their learning capacity. *Residual connections* [16] are created to facilitate gradient flow across the recurrent units to the layers below. Further, we use *dropout* [24] along depth connections to regularize the network, without modifying the recurrent connections, thereby preserving the network's capacity to capture long-term dependencies. To prevent covariate shift due to minibatch training, the activities of the cell neurons are *Layer Normalized* [2], which proved to be quite effective in stabilizing the hidden state dynamics of the network. For the final prediction we apply a linear transformation $W \in \mathbb{R}^{256 \times N}$ on the RNN predictions, where N is the output vocabulary size, to generate the logits. *Softmax* operation is performed on the logits to define a probability distribution over the output vocabulary at each timestep.

3.4 Training & Inference

In our experiments, while training, the batch size is set to 16. We use *Adam* [17] algorithm as the optimizer with a learning rate of 0.001. The model was trained till the best validation accuracy, achieved after 30 epochs. For inference, we use a beam size equal to the number of classes in the output.

4 Dataset

We use the following publicly available datasets to evaluate our method.

IAM Handwriting Database v3.0 (English) [22] is composed of 1539 pages of text, written by 657 different writers and partitioned into writer-independent training, validation and test sets of 6161, 940, 1861 segmented lines, respectively. The line images have an average height of 124 pixels and average width of 1751 pixels. There are 79 different characters in the database, including whitespace.

RIMES Database (French) [1] has 12,723 scanned pages of mails handwritten by 1300 people. The dataset consists of 11333 segmented lines for training and 778 for testing. Original database doesn't provide a separate validation set and therefore we randomly sampled 10% of the total training lines for validation. Final partition of the dataset contains 10203 training lines, 1130 validation lines and 778 test lines. Average width of the images is 1658 pixels and average height is 113 pixels. There are 99 different characters in the dataset.

5 Experiments

We evaluate our model on the evaluation partition of both datasets using mean *Character Error Rate* (CER) and mean *Word Error Rate* (WER) as performance metrics determined as the mean over all text lines. They are defined as,

$$CER = \frac{\text{Number of characters in a sentence wrongly classified}}{\text{Total characters in a sentence}}$$

$$WER = \frac{\text{Number of words in a sentence spelled incorrectly}}{\text{Total words in a sentence}}$$

Our experiments were performed using an Nvidia Tesla K40 GPU. The inference time for the model is 1.5 seconds.

5.1 Results

Table 3 shows the effect of Layer Normalization (LN), Focal Loss and Beam Search on the base model. LN improved the performance of the base model by around 3%. The use of Focal Loss also increased the accuracy by $1 - 3\%$ but major improvement was achieved by replacing greedy decoding with beam search which boosted the model accuracy by $4 - 5\%$.

Table 3: Effect of Layer Normalization, Focal Loss & Beam Search on Model Performance

	IAM		RIMES	
System	CER(%)	WER(%)	CER(%)	WER(%)
Baseline	17.4	25.5	12.0	19.1
+ LN	13.1	22.9	9.7	15.8
+ LN + Focal Loss	11.4	21.1	7.3	13.5
+ LN + Focal Loss + Beam Search	**8.1**	**16.7**	**3.5**	**9.6**

5.2 Comparison with the state-of-the-art

We provide a comparison of the accuracy of our method with previously reported algorithms in Table 4 and a comparison of the efficiency, in terms of maximum GPU memory consumption and number of trainable parameters, with the state-of-the-art in Table 5.

Table 4: Comparison with previous methods in terms of accuracy

	IAM		RIMES	
Methods	CER(%)	WER(%)	CER(%)	WER(%)
2DLSTM [10], reported by [24]	8.3	27.5	4.0	17.7
CNN-1DLSTM-CTC [24]	6.2	20.2	2.6	10.7
Our method	8.1	**16.7**	3.5	**9.6**

Table 5: Comparison with state-of-the-art in terms of efficiency

Methods	Memory (GB)	# of Parameters (Mi)
CNN-1DRNN-CTC [24]	10.5	9.3
Our method	**7.9**	**4.6**

Although we beat the state-of-the-art [24] at word level accuracy, our character level accuracy is slightly lower in comparison. It implies that our model is prone to make additional spelling mistakes in words which have already got mislabeled characters in them, but overall makes fewer spelling mistakes at the aggregate word level. This arises out of the inference behavior of our model, which uses the previous predictions to generate the current output and as a result, a prior mistake can trigger a sequence of future errors. But, higher word accuracy proves that most often, our model gets the entire word in a line correct. Essentially, the model is quite accurate at identifying words but when a mistake does occur, the word level prediction is off by a larger number of characters.

6 Summary and Extensions

We propose a novel framework for efficient handwritten text recognition that combines the merits of two extremely powerful deep neural networks. Our model substantially exceeds performance of all the previous methods on a public dataset and beats them by a reasonable margin on another. While the model performs satisfactorily on standard testing data, we intend to carry out further evaluations to ascertain its performance on completely unconstrained settings, with different writing styles and image quality.

An extension to the present method would be to develop a training procedure that would optimize a loss dependant on the correctness of a full sequence instead of a cumulative loss of independent characters, resulting in similar behavior of the model at training and inference. Also, a language model can be incorporated in the training scheme to further augment the performance of the model and correct for mistakes, especially for rare sequences or words.

References

[1] Emmanuel Augustin, Matthieu Carré, Emmanuèle Grosicki, J-M Brodin, Edouard Geoffrois, and Françoise Prêteux. Rimes evaluation campaign for handwritten mail processing. In *International Workshop on Frontiers in Handwriting Recognition (IWFHR'06),*, pages 231–235, 2006.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Yoshua Bengio. Markovian models for sequential data. *Neural computing surveys*, 2 (199):129–162, 1999.

[5] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[6] Herve A Bourlard and Nelson Morgan. *Connectionist speech recognition: a hybrid approach*, volume 247. Springer Science & Business Media, 2012.

[7] Horst Bunke. Recognition of cursive roman handwriting: past, present and future. In *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, pages 448–459. IEEE, 2003.

[8] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[9] A El-Yacoubi, Michel Gilloux, Robert Sabourin, and Ching Y. Suen. An hmm-based approach for off-line unconstrained handwritten word modeling and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):752–760, 1999.

[10] Alex Graves and Jürgen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in neural information processing systems*, pages 545–552, 2009.

[11] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.

[12] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5): 855–868, 2009.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[14] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[16] Jaeyoung Kim, Mostafa El-Khamy, and Jungwon Lee. Residual lstm: Design of a deep recurrent architecture for distant speech recognition. *arXiv preprint arXiv:1701.03360*, 2017.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2017.

[19] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[20] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[21] U-V Marti and Horst Bunke. Using a statistical language model to improve the performance of an hmm-based cursive handwriting recognition system. In *Hidden Markov models: applications in computer vision*, pages 65–90. World Scientific, 2001.

[22] U-V Marti and Horst Bunke. The iam-database: an english sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, 5(1):39–46, 2002.

[23] Vu Pham, Théodore Bluche, Christopher Kermorvant, and Jérôme Louradour. Dropout improves recurrent neural networks for handwriting recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 285–290. IEEE, 2014.

[24] Joan Puigcerver. Are multidimensional recurrent layers really necessary for handwritten text recognition? In *Document Analysis and Recognition (ICDAR), 2017 14th IAPR International Conference on*, volume 1, pages 67–72. IEEE, 2017.

[25] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[26] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(11):2298–2304, 2017.

[27] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[28] Alessandro Vinciarelli. A survey on off-line cursive word recognition. *Pattern recognition*, 35(7):1433–1446, 2002.