Dissecting Multi-Line Handwriting for Multi-Dimensional Connectionist Classification

Martin Schall*†
Email: martin.schall@htwg-konstanz.de
*Institute for Optical Systems
University of Applied Sciences
Konstanz, Germany

Marc-Peter Schambach†
Email: marc-peter.schambach@siemens.com
†Siemens Logistics GmbH
Konstanz, Germany

Matthias O. Franz
Email: mfranz@htwg-konstanz.de
Institute for Optical Systems
University of Applied Sciences
Konstanz, Germany

Abstract—**Multi-Dimensional Connectionist Classification is a method for weakly supervised training of Deep Neural Networks for segmentation-free multi-line offline handwriting recognition. MDCC applies Conditional Random Fields as an alignment function for this task. We discuss the structure and patterns of handwritten text that can be used for building a CRF. Since CRFs are cyclic graphical models, we have to resort to approximate inference when calculating the alignment of multi-line text during training, here in the form of Loopy Belief Propagation. This work concludes with experimental results for transcribing small multi-line samples from the IAM Offline Handwriting DB which show that MDCC is a competitive methodology.**

I. Introduction

Offline handwriting recognition is the automatic transcription of natural handwritten text from images to computer-processable character strings. Often this involves the transcription of paragraphs of multiple text lines. One approach to process multi-line texts is to segment the paragraph image into multiple line images and then transcribe each line on its own. This approach in combination with *Connectionist Temporal Classification (CTC)* [1] has lead to state-of-the-art systems [2] [3] [4] [5] in recent years.

A well known problem with this general approach is that both segmentation and transcription are prone to errors which accumulate. Errors in segmentation may lead to larger errors in transcription. This dependency is described as *Sayre's knot* [6]: Correct segmentation requires correct transcription; correct transcription requires correct segmentation.

One way to untangle these dependencies is to not treat segmentation and transcription as two separate procedures but as two aspects of one single procedure. A recently proposed method [7] [8] simultaneously uses a *Deep Neural Network (DNN)* with an attention-mechanism for segmentation by steering attention and CTC for transcription. This is an explicit transformation of the multi-line text to a one-dimensional sequence. Another method is *Multi-Dimensional Connectionist Classification (MDCC)* [9] which proposes a loss function and decoding algorithm that allows for training a DNN to transcribe multi-line text without segmentation and without explicit transformation to a one-dimensional sequence.

MDCC consists of two separate procedures: first, an Expectation-Maximization-style training using *Conditional Random Fields (CRFs)* [10] and *Loopy Belief Propagation (LBP)* [11] [12, p. 769] to align the truth label string to the two-dimensional DNN output and defining a loss function to train the DNN to estimate the correct probabilities of individual 'pixels' belonging to certain glyphs. CRFs and LBP are well known methods in computer vision. Second, decoding this probabilistic DNN output to retrieve a computer-processable character string of the multi-line text. These procedures are shown schematically in Figure 1.

MDCC requires the DNN to accept a two-dimensional image of text as input and to produce a two-dimensional output where each 'pixel' is a probability vector estimating that this 'pixel' is part of a certain glyph from the alphabet. The actual DNN topology can be chosen according to the problem at hand. DNNs based on *Multi-Dimensional Long Short-Term Memory (MDLSTM)* [13] [14] have been used for MDCC before. We used a hybrid CNN+LSTM network [15] for this work.

Training the DNN using MDCC is based on stochastic or mini-batch gradient descent using backpropagation. Estimation of glyph probabilities, which are also necessary for later decoding and transcription, is done by the DNN. MDCC then sets up a loss function to improve this estimation. Correct probabilities are not known since the training data only consists of the input images and the corresponding truth label strings. No information about the location or size of the characters is included in the training data. The correct probabilities can be approximated by constructing a CRF that encodes the truth label string and relies on approximate inference to obtain the correct probabilities. This is called the 'alignment' of the truth label string to the two-dimensional DNN output. Constructing the CRF is specified in the following Sections II and III. The DNN parameters are then adapted by calculating the *multi-nomial cross-entropy loss* of the estimated and corrected probabilities, performing backpropagation and gradient descent using one of the standard gradient descent algorithms for neural networks. This training loop is repeated until a satisfactory estimation of the probabilities is achieved by the DNN.

This work improves the original MDCC by introducing 8- instead of 4-neighborhood relations for alignment. This solves difficult cases were text lines are aligned along a diagonal in the pixel space. It also exchanges the potential functions

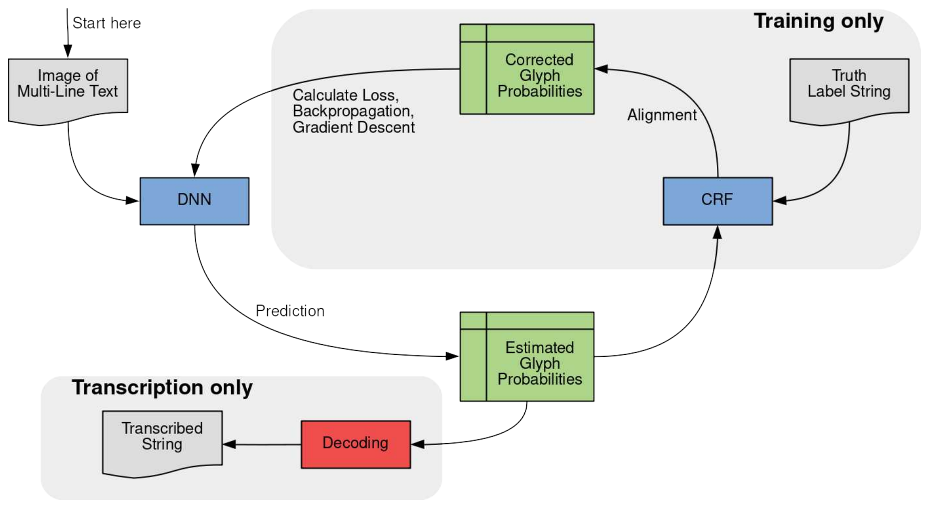

Fig. 1. Overview of both training and transcription. Training is an Expectation-Maximization-style approach were first the DNN parameters are held constant and the CRF alignment is updated, then the CRF alignment is held constant and the DNN parameters are updated.

from a Potts model for ones that are specifically designed for handwritten text.

Decoding the two-dimensional probabilistic output from the DNN is done for transcription. The decoding mechanism is based on a horizontal scan-line that is initialized on the very top of the two-dimensional output. It is then moved top-to-bottom and left-to-right while alternating between visible lines and line separators. Visible lines are decoded to text lines while the scan line moves through them and newline characters are added while moving through the line separators. This transcribes a multi-line text from the two-dimensional probabilities output estimated by the DNN. Please see [9] for more information on the decoding mechanism.

The main body of this work focuses on the ideas and theoretical aspects of MDCC. Comparison with a previous variant of MDCC and the attention-based transcription method [7] is done by an experiment and evaluation of error rates.

Section II discusses the structure of handwritten text and how to derive the topology of the CRF used in MDCC from it. Section III defines the CRF potential functions and completes the definition of the CRF. Section IV discusses the loss function for training a DNN using MDCC. Section V gives a brief overview on why approximate inference is necessary for MDCC. Section VI includes experimental results for transcribing small multi-line samples from the IAM Offline Handwriting DB. Section VII concludes this work with a brief discussion.

II. Topology

CRFs are undirected cyclic graphical models of multi-variate probability distributions. As such we need to define the topology of the graphical model in order to work with a CRF, e.g. for inference. Nodes in the case of MDCC are ‘pixels’ of the DNN output and states the individual positions within the truth label string for supervised training. We can use this to infer the probabilities of individual ‘pixels’ belonging to certain glyphs. The CRF in MDCC is constructed by defining two separate graphs, a *pixel graph* and a *label graph*, and then computing the *graph tensor product* of both as discussed later in this section. Nodes of the pixel graph translate to the nodes of the CRF and nodes of the label graph to the states of the CRF. Both graphs are directed but directions are dropped after calculating the graph tensor product and the resulting CRF will be an undirected graph. This is possible since both graphs define neighborhoods, which are undirected in nature, and directions are only used for easier understanding of the two graphs’ topologies.

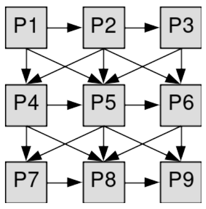

Fig. 2. Example pixel graph for a DNN output of 3×3 ‘pixels’ in size.

The topology of the pixel graph is rather simple: it is a regular grid of nodes, each node s corresponding to one ‘pixel’ of the DNN output. As such the edges of the pixel graph are one edge in each of the four directions of the directed graphs. Figure 2 shows one such example pixel graph.

Defining the topology of the label graph requires some thought about the structure of multi-line handwritten text. The nodes of the label graph correspond to individual positions within the truth label string. The truth label string contains newline characters to separate lines. Edges of the label graph still represent steps of one ‘pixel’ distance in one of the four directions. The visual shape of glyphs and structure of the text thus define the topology of the label graph.

We will use the term *glyph* for a visible/printable element of the alphabet. *Character* denotes one specific instance of a

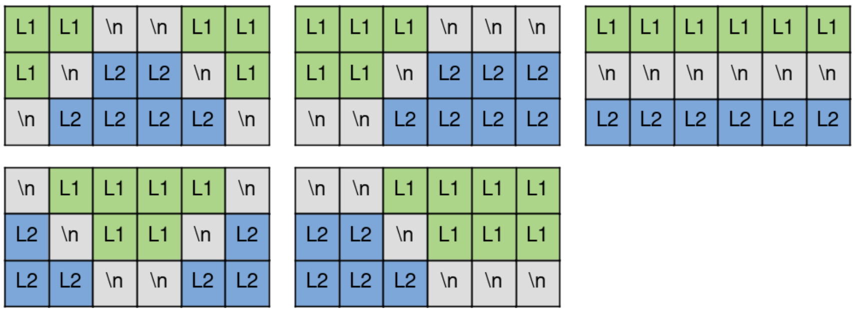

Fig. 3. Possible patterns of the boundary between two text lines. First/Upper line in green, second/lower line in blue.

glyph. For example in ‘hello’ the glyph ‘l’ exists once in the alphabet but the character ‘l’ occurs twice in the string.

The language and writing system in which the text is written allows us to define some basic parameters for the structure of the text. For English handwriting these would be that lines are written from left to right and lines are ordered from top to bottom.

Further parameters arise out of the requirements that the decoding algorithm sets. We are free to define these parameters ourselves since the decoding algorithm is part of MDCC, but it must be ensured that these parameters are respected during both training and decoding. We assume that there are more ‘pixels’ in the DNN output than the truth label is in length. Repetition of lines and/or glyphs is thus necessary since all ‘pixels’ must belong to some position in the truth label string. Otherwise said, the sum of state probabilities of the CRF must sum to one per node. This means that lines and/or glyphs are ‘blobs’ in pixel space. Further we assume that these ‘blobs’ are continuous, e.g. a glyph ‘O’ is a continuous area and not a circle with a dot of background in the middle. Modeling glyphs any other way than ‘blobs’ would require knowledge about the shape of glyphs and thus make it necessary to model prototypical glyphs. MDCC avoids this by only aligning characters by their location and size, not shape.

A special *line separator* is introduced to the alphabet. This line separator must span from the left to the right borders. Its occurrence means that the ‘pixels’ above and below belong to two different lines. If two ‘pixels’ are not separated by a line separator then they are assumed to belong to the same line. We can further specify that two characters in the same line can be distinguished if they belong to two different glyphs. A *glyph separator* label is introduced to distinguish between two adjacent characters that are the same glyph. An example for this would be the string ‘hello’ which makes it necessary to recognize the two ‘l’ glyphs as two different characters. The solution is to introduce the glyph separator within the glyph repetition: ‘helϵlo’. The approach of adding separators is proposed by CTC, but in a different way than MDCC: CTC adds optional separators between all character pairs and mandatory separators between character pairs of the identical glyph. MDCC only adds mandatory separators and omits optional ones.

Figure 3 shows some possible patterns of the boundary between two text lines. Please note that these are examples and the full multi-line text in pixel space is a combination and repetition of these simple patterns. These simple patterns already lead to some observations: in pixel space a horizontal move to the right could lead to a change to both the previous or the next line. This is because of the two directions in which text lines can be slanted. Also a single pixel row can hold both the first line, second line and then the first again. We should not assume that one pixel row always corresponds to only one text line or that the order of text lines in a pixel row follows a strict linear order. On the other hand, a movement from top to bottom in pixel space will always traverse the text lines in strict ascending order.

Figure 4 contains some example patterns for the boundary between two characters of the same text line. Again these are examples that must be combined and repeated to contain the full multi-line text in pixel space. These patterns support similar observations as in text lines: horizontal movement results in an ascending order of the characters whereas a vertical movement can mean a jump to both the previous or next character.

We can now continue to define the topology of the label graph using this knowledge about the structure of the given writing system. Each node x_s of the label graph corresponds to one text position in the truth label string. Label separators and glyph separators are introduced where necessary. The rules for introducing edges to the label graph are as follows, see Figure 5:

1) All characters as well as the glyph and label separators can repeat themselves in all four directions. This introduces four loops (four directions) at each node.
2) A transition to the next character in the same text line is possible for horizontal right, vertical down and diagonal down-right moves in pixel space.
3) Vertical and diagonal down-left moves in pixel space allow a transition to the previous character of the same text line.
4) A switch to the next text line is possible in all four directions in pixel space.
5) A horizontal right movement allows the transition back to the previous text line.

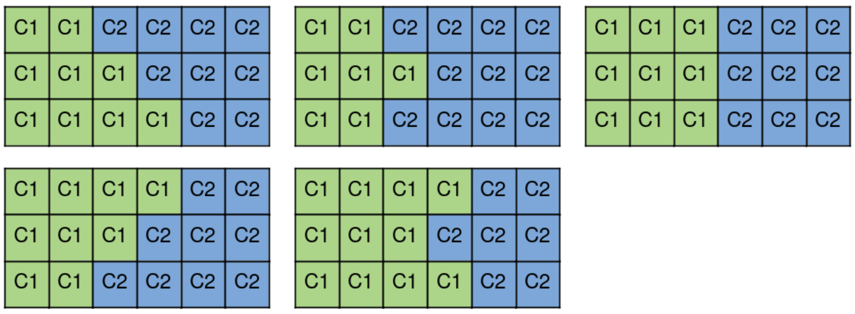

Fig. 4. Possible patterns of the boundary between two characters within the same text line. First/Left character in green, second/right character in blue.

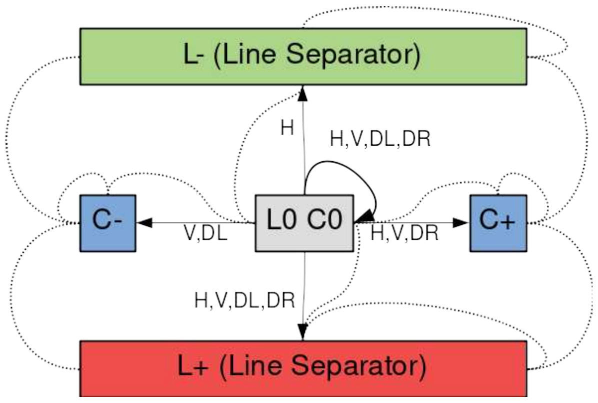

Fig. 5. Abstract label graph showing the transitions from one character to its neighboring characters and lines. Solid directed arrows show actual edges in the label graph. Dotted undirected lines indicate neighborhoods that must be realized according to the described rules.

Figure 5 shows an abstracted label graph around one character. The outgoing edges from the character 'L0 C0' are constructed using the above rule set. The missing edges are indicated by dotted undirected edges and must be realized while building the label graph.

The graph tensor product used in MDCC is restricted by only allowing the product of edges that share the same direction. In total there are four edge directions in the directed graphs: horizontally to the right, vertically downwards, diagonal down-left and diagonal down-right. This results in 8-neighborhoods in the CRF when dropping the directions. According to the MDCC variant of the graph tensor product, an edge $(s, x_s) \sim (t, x_t)$ exists in the CRF iff $s \sim_d t \wedge x_s \sim_d x_t$ with d being the edge direction, s, t nodes in the pixel graph and x_s, x_t nodes in the label graph. (s, x_s) and (t, x_t) are node-state combinations in the resulting CRF. The meaning of the edges in the two graphs is the answer to the following question: when moving one 'pixel' in this direction, which nodes are possible neighbors?

A CRF is defined by its *node potential function* and *edge potential function* that define the 'compatibility' between nodes and their states. The potential functions of a CRF may contain *structural zeros*, potential values of exactly zero that disallow the corresponding node-state combination completely. These structural zeros allow us to think about the CRF first in terms of graph topology were allowed node-state combinations and edges have potential values greater than zero and disallowed ones a potential value of exactly zero. Computing the graph tensor product out of the described pixel and label graphs produces the topology of the CRF. The actual node and edge potential function will be defined in Section III, but the topology already defines which potentials are greater than zero and which are structural zeros.

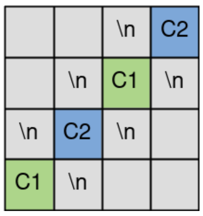

Fig. 6. Example of a text line that is incorrectly aligned along a diagonal. The correct label sequence is 'C1 C2' and not 'C1 C2 C1 C2' in this example.

The topology presented in this work is based on 8-neighborhoods in the pixel grid instead of 4-neighborhoods as used in the original MDCC publication [9]. This was inspired by the observation that text lines along a diagonal could be incorrectly aligned using 4-neighborhoods. One example for this is shown in Figure 6 where the text line 'C1 C2' should be aligned, but instead 'C1 C2 C1 C2' is occurring as a pattern. This is solved by introducing diagonal dependencies into the topology.

III. Potential Functions

In Section II we have discussed the topology of the CRF in use for MDCC. For building and using an actual CRF we further need to define node and edge potential functions. These define the 'compatibility' between nodes and states of the CRF. In the MDCC case, the node potential function $\psi_s(x_s)$ defines the compatibility between 'pixel' s of the DNN output and position x_s within the truth label string. Furthermore the edge potential function $\psi_{s,t}(x_s, x_t)$ defines the compatibility between two neighboring 'pixel' s, t and two positions x_s, x_t and as such controls the edges in the CRF.

The edge potential function $\psi_{s,t}(x_s, x_t)$ is derived out of the topology of the CRF. Structural zeros are modeled in the edge potential function and as such, $\psi_{s,t}(x_s, x_t) = 0$ iff no edge $(s, x_s) \sim (t, x_t)$ exists in the CRF topology. $\psi_{s,t}(x_s, x_t)$ is positive otherwise.

The actual choice of the potential function values greater than zero is more or less free depending on the problem at hand. Greater values of the potential functions mean higher compatibility. It has some benefits to choose exponential functions in the form of $\psi_{s,t}(x_s, x_t) = \exp(...)$:

1) LBP computes repeated multiplications of the potential function values for message passing. Computation in log-space improves numerical stability and exponential functions are easily integrated into a log-space implementation.
2) Choosing exponents greater than one for the potential functions leads to a reliable convergence of beliefs in LBP in our experience.

We can further make the following assumptions about handwritten text:

1) A horizontal movement in pixel space will less likely result in a change of text lines than staying within the same text line.
2) A vertical movement in pixel space will more likely result in either continuing the same character or changing the text line than changing the character within the same text line.

Based on this we have chosen the following values for the edge potential function $\psi_{s,t}(x_s, x_t)$:

1) $e^{1.5}$ if s, t are in the same pixel row and x_s, x_t are in the same text line.
2) e^1 if s, t are in the same pixel row and x_s, x_t are not in the same text line.
3) $e^{1.5}$ if s, t are not in the same pixel row and x_s, x_t are either identical or not in the same text line.
4) e^1 if s, t are not in the same pixel row and x_s, x_t are different characters in the same text line.

The node potential function $\psi_s(x_s)$ also models structural zeros. These occur whenever a truth label position x_s cannot occur in 'pixel' s. This is the case if such a combination of s, x_s would make it impossible to fit the remaining truth label string. For example the character 'e' of 'hello' can never occur in the leftmost pixel column since this retains no space for the 'h' character. This means $\psi_s(x_s) = 0$ iff the s, x_s combination leads to an invalid configuration and $\psi_s(x_s) > 0$ in all other cases.

DNN training with MDCC is based on estimating the probabilities of individual pixels of the input image belonging to certain glyphs from the alphabet. This is what the DNN is trained for in MDCC. The CRF in MDCC is used to include the knowledge of the truth label string to correct the estimated probabilities, set up a loss function and optimize the DNN parameters. This means that over time the estimation by the DNN is improving and MDCC integrates these improvements in the CRF. This is why the CRF node potential function respects the estimation by the DNN.

Again we choose exponential functions in the form of $\psi_s(x_s) = \exp(...)$ for the node potential function:

$$\psi_s(x_s) = \exp(k_1 + k_2 \times \mathrm{FA}_s(x_s) + k_3 \times \mathrm{DNN}_s(x_s)) \quad (1)$$

Constants k_1, k_2 and k_3 of Equation 1 are chosen to weight the three influences in the node potential against each other. k_1 is introduced to ensure that $\psi_s(x_s) > exp(1)$ for non structural zeros and to improve the convergence of beliefs in LBP.

Function $\mathrm{DNN}_s(x_s)$ is the current estimation of the DNN that s belongs to x_s. The DNN actual estimates the probability of a 'pixel' s belonging to a certain glyph g, whereas x_s is a position within the truth label string. As such we need to introduce a mapping: $\mathrm{DNN}_s(x_s) = \mathrm{DNN}_s(g), g = S(x_s)$ with S being the truth label string.

Function $\mathrm{FA}_s(x_s)$ gives the probability of s belonging to x_s based on a two-dimensional *Forced Alignment (FA)* [16]. This two-dimensional variant of FA assumes that all text lines are of the same height in pixel space (± 1 pixel) and that text lines are separated by a line separator of exactly one pixel in height. Furthermore all characters of all text lines are assumed to be of the same width (again ± 1 pixel). Using this information, text lines and characters are then ordered top-to-bottom and left-to-right. This results in a spatially uniform placement of the characters and lines. To achieve smoothness at character overlaps, only line separators are aligned with a probability of one, all other characters probabilities are calculated by a normal distribution with half a glyph width of standard deviation.

We chose the following values for the three constants to complete the node potential function: $k_1 = 1$, $k_2 = 5$ and $k_3 = 10$. This gives the most weight to the DNN estimation but ensures a reliable alignment even at the beginning of training or for erratic DNN estimations.

The CRF of MDCC is now defined in the form specified by Equation 2:

$$P(C|\,\mathrm{DNN}) = \frac{1}{Z(\mathrm{DNN})} \prod_s \psi_s(x_s) \prod_{t \in \mathrm{nbr}\, s} \psi_{s,t}(x_s, x_t) \quad (2)$$

Equation 2 gives the probability $P(C|\,\mathrm{DNN})$ of a configuration C being a valid alignment of the truth label string given the DNN estimation. In this case, x_s and x_t are specific assignments of characters to 'pixels' defined by the configuration C. Function Z is called *Zustandssumme* and is the sum over all possible configurations:

$$Z(\mathrm{DNN}) = \sum_C \prod_s \psi_s(x_s) \prod_{t \in \mathrm{nbr}\, s} \psi_{s,t}(x_s, x_t) \quad (3)$$

We have modified the potential functions in this work as compared to the original MDCC [9] to ensure that non-zero potential values are always greater than $exp(1)$. Also edge potentials are not based on a Potts model anymore but implement specific ideas about the structure of handwriting. Node potentials were updated to stronger reflect the influence of the DNN and FA. This was made necessary since in larger pixel spaces the edge potentials seemed to have a too strong influence on the result when compared with the node potentials. This then may result in an alignment that does not reflect the DNN estimations of the character positions and sizes.

IV. Network Training

Training of the DNN is implemented by estimating $\mathrm{DNN}_s(g)$ and correcting it using the described CRF, see Sections II and III, to obtain the corrected probabilities $\mathrm{CRF}_s(g)$. Approximation of $\mathrm{CRF}_s(x_s)$ is done using LBP in *Sum-Product Mode* and are also called *Beliefs*, which are proportional to the actual probability of x_s occurring in s: $\mathrm{CRF}_s(x_s) \propto P_s(x_s)$.

LBP in sum-product mode on this CRF approximates the mean probabilities of 'pixel' s belonging to a certain character x_s given the DNN estimation and under the condition that each configuration C encodes the truth label string. See Equation 4 for this marginal probability. It is worth noting that LBP and message passing in general does at no point explicitly create any configuration C of the graphical model. Instead the marginal probabilities are estimated without observing any configuration.

$$\mathrm{CRF}_s(x_s) \approx \frac{1}{|C|} \sum_{C:C(s)=x_s} P(C|\,\mathrm{DNN}) \tag{4}$$

MDCC uses multi-nomial cross-entropy as its loss function L, see Equation 5. Corrected probabilities $\mathrm{CRF}_s(g) = \sum_{x_s:S(x_s)=g} \mathrm{CRF}_s(x_s)$ are computed by summing up all truth label positions that are the glyph in question. This is necessary since the same glyph g may occur multiple times in the truth label string S.

$$L = -\sum_s \sum_g \mathrm{CRF}_s(g) \times \log(\mathrm{DNN}_s(g)) \tag{5}$$

The derivative of the loss function L from Equation 5 is as follows:

$$\frac{\partial L}{\partial\,\mathrm{DNN}_s(g)} = -\frac{\mathrm{CRF}_s(g)}{\mathrm{DNN}_s(g)} \tag{6}$$

The loss function L and its derivative are then applied to train the DNN for MDCC. Note that $\mathrm{CRF}_s(g)$ is held constant during the update of the DNN weights in our EM-like training procedure, thus we do not need an inner derivative of $\mathrm{CRF}_s(g)$ in Equation 6. After training the DNN will be able to perform segmentation-free multi-line offline handwriting recognition. A decoding function for producing a computer-processable character string from the DNN estimation is necessary as specified and discussed in [9].

V. Comparison with Exact Inference

LBP approximates marginal probabilities as specified in Section IV. We would like to show that approximation of these probabilities is the only computationally tractable way of performing this style of training. The graphical model of a multivariate probability distribution, here a CRF, includes cycles. These cyclic dependencies make it impossible to apply optimized algorithms such as *Forward-Backward* [17] or non-loopy *Message Passing* [18]. One example inference algorithm to calculate the exact marginal probabilities without approximation is to enumerate all valid configurations C of the cyclic graphical model. Exact inference in general cyclic graphical models is known to be NP-hard [19] [20]. There are chain-structured cyclic graphical models, e.g. alignment in CTC, that can be separated into two directed acyclic models that enable exact inference in polynomial time. Unfortunately this chain-structure does not hold true for the CRF in MDCC, which has a grid-structure.

The number of valid configurations for the string 'aa$\backslash n$bc' in pixel grids of sizes 4x4, 5x4 (width x height), 5x5 and 6x5 are shown in Table I. We have stopped the enumeration of the valid configurations within a 6x6 grid after several hours of run time. Run time measurements using approximate inference were continued for some larger pixel grid sizes. A grid size of 100x100 was added to show that MDCC can be applied to paragraph-sized outputs with reasonable run time. Time measurements shown in Table I were performed on a 2.2 GHz Intel Core i7-6560U.

Run time of LBP is dependent on the number of message passing iterations that are required to converge to a stable point. The convergence criteria used in this work is to check the mean change in LBP messages from one iteration to the next. LBP is stopped if this mean change falls below a threshold of 3×10^{-6} at any iteration. Our experience with MDCC is that 10 to 100 iterations of message passing are enough in the described CRF.

The mean difference between approximated probabilities $\mathrm{CRF}_s(x_s)$ and exact probabilities $P_s(x_s)$ for this example in a 6x5 grid was 0.0407163 with a standard deviation of 0.0834019.

VI. Results

We evaluated MDCC on the identical data set as used in the original work [9]. Examples of 2 lines with 3 words each were extracted from the *IAM Offline Handwriting Database* [21]. The two lines were always two consecutive lines of the source paragraph as well as the 3 words were consecutive in those lines. This results in a data set of 11508 examples. One such example is shown in Figure 7. Training was performed on 80 percent of the data (9208 examples) while 10 percent (1150 examples) each were used for validation and evaluation.

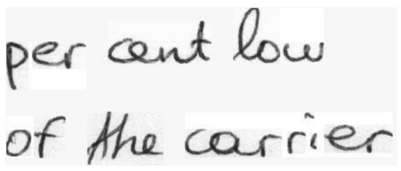
per cent low
of the carrier

Fig. 7. Example input with 2 lines of 3 words each. Constructed from the IAM Offline Handwriting DB.

The DNN topology for the experiment is a hybrid CNN+LSTM and specified in Table II. Each convolutional block consists of a 2D convolution with added padding to keep the output the same size as the input. The convolution is followed by batch normalization [22] and a leaky ReLU

TABLE I
COMPARISON OF ENUMERATING ALL VALID CONFIGURATIONS AND APPROXIMATE INFERENCE USING LBP.

Pixel Grid Size	Num. Configurations	Run time Enumeration	Run time LBP
4x4	3440	0.091s	0.002s
5x4	56480	1.360s	0.003s
5x5	3033992	82.035s	0.005s
6x5	74116576	2301s (38m)	0.007s
6x6			0.010s
10x10			0.047s
100x100			69.434s

[23] activation function. The feature maps of the DNN are two dimensional at any layer. The bidirectional LSTM layers are column-wise (vertical) or row-wise (horizontal) with each column or row being treated independently from all others.

TABLE II
DEEP NEURAL NETWORK TOPOLOGY. THE COLLAPSE LAYER WAS ONLY USED FOR PRE-TRAINING, NOT FOR MULTI-LINE TRAINING USING MDCC.

Layer Type	Parameters
Input image	Gray scale.
Conv. block	Kernel size 5x5. 64 neurons.
Max. pooling	Window size 3x3.
Conv. block	Kernel size 5x5. 96 neurons.
Max. pooling	Window size 3x3.
Conv. block	Kernel size 5x5. 128 neurons.
Max. pooling	Window size 2x2.
Conv. block	Kernel size 5x5. 196 neurons.
Conv. block	Kernel size 5x5. 256 neurons.
Vertical BLSTM	256 cells total. 128 cells per direction.
Horizontal BLSTM	256 cells total. 128 cells per direction.
Vertical BLSTM	256 cells total. 128 cells per direction.
Horizontal BLSTM	256 cells total. 128 cells per direction.
Conv. layer	Kernel size 1x1. One neuron per glyph.
(Collapse layer)	Column-wise summation.
Softmax	

The DNN was pre-trained on IAMDB words using CTC. Pre-training was for 10 epochs with a mini-batch size of 16 examples. Training using MDCC and the data set consisting of examples of 2 text lines was started after these 10 epochs. Mini-batch size for MDCC training was 4 examples. The optimizer for both training phases was RMSProp [24] with a learning rate of 0.001.

Error rates were measured in *Character Error Rate (CER)* as specified in Equation 7. CER is the fraction of the Edit-distance [25] [26] of the decoded string d and the truth string t divided by the length of the truth string. The convergence of CER during MDCC multi-line training is shown in Figure 8.

$$\mathrm{CER}(d,t) = \frac{\mathrm{Edit}(d,t)}{|t|} \times 100 \qquad (7)$$

Error on the validation data set reached its minimum after 115 epochs at a CER of 6.00, while a CER of 0.59 was measured on the training data set. CER on the independent evaluation data set was 5.47 after 115 epochs. This is an improvement over the CER of 10.4 as reported by [9] on the identical data set. It also indicates a potential improvement over a CER of 10.9 as reported by [7, p. 6] on a similar, but not publicly available data set.

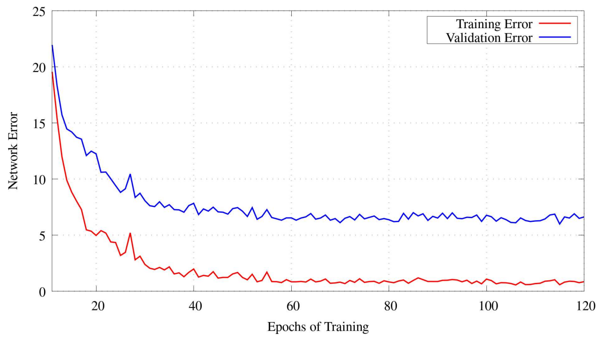

Fig. 8. Convergence rate of the DNN error while training for transcription of 2 lines with 3 words each. First 10 epochs were pre-training using CTC.

PyTorch [27] was used to implement the described DNN and training. PyTorch allows easy utilization of a NVIDIA GPU for training. The DNN in this experiment was executed on a NVIDIA GeForce GTX 1080 Ti. Both the CTC and MDCC implementations are optimized for execution on a CPU which made memory transfers between the main memory and GPU memory necessary. An Intel Core i5-6500 with 3.2GHz was used for calculating the CTC and MDCC loss during training.

Each epoch of MDCC training took approximately 58 minutes. This results in a speed of 3.4 examples per second during training. Transcription speed for evaluation was 14.5 examples per second. The difference is due to the execution of LBP for approximate inference during training.

Figures 9 and 10 show heat maps of glyph probabilities for the example of Figure 7. Figure 9 shows glyph probabilities for 'e' and has nearly correct localization of the characters. It also shows that the aligned characters are continuous areas, but do not necessarily have to be of a rectangular shape. Figure 10 shows the glyph 'r' that has two adjacent occurrences which are separated by a glyph separator to distinguish the two characters of the same glyph.

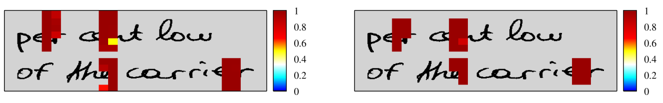

Fig. 9. Heat maps of DNN (left) and CRF (right) probabilities for glyph 'e'.

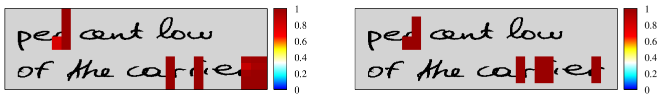

Fig. 10. Heat maps of DNN (left) and CRF (right) probabilities for glyph 'r'.

VII. Discussion

In this work we have discussed the structure of handwritten text and how to model its properties with cyclic graphical models in the form of Conditional Random Fields. We provided details on how to use this CRF to calculate an alignment of multi-line text over two-dimensional images. This sets the stage for training Deep Neural Networks for segmentation-free multi-line offline handwriting recognition by applying an approach similar to Connectionist Temporal Classification but in two dimensions: first calculate the alignment of the truth label string over the DNN output and then use this alignment to set up a loss function for DNN training. MDCC thus implements weakly supervised training of DNNs for multi-line handwriting recognition.

We further believe that MDCC can serve as a framework for application in higher-dimensional spaces as well. Problems based on the classification of multiple objects (in this case glyphs) in a multi-dimensional space can in theory be modeled by MDCC. Necessary is the specification of the space in which the alignment takes place, which in this case is a 2-dimensional pixel grid but could also be a discrete 3- or 4-dimensional space. Knowledge about the geometric relations between the objects is also required, in the case of handwriting the writing system on how to order lines and characters.

The experimental results of this work are an improvement on the previous variant of MDCC on the same data set. Error rates are competitive and show that MDCC can be used for practical applications of offline handwriting recognition. We believe the improved error rates are because of changing to 8- instead of 4-neighborhoods and adopting CRF potential functions that are specific to handwritten text.

Next steps for applying MDCC are to transcribe whole paragraphs of the IAM Offline Handwriting DB and to transcribe industrial data.

Acknowledgment

The authors would like to thank the Siemens Logistics GmbH for funding this work.

References

[1] A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber, "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks," in *Proceedings of the 23rd international conference on Machine Learning*. ACM Press, 2006, pp. 369–376.

[2] P. Voigtlaender, P. Doetsch, and H. Ney, "Handwriting Recognition with Large Multidimensional Long Short-Term Memory Recurrent Neural Networks," in *Proceedings of International Conference on Frontiers in Handwriting Recognition, ICFHR*, 2017, pp. 228–233.

[3] P. Voigtlaender, P. Doetsch, S. Wiesler, R. Schlüter, and H. Ney, "SEQUENCE-DISCRIMINATIVE TRAINING OF RECURRENT NEURAL NETWORKS," in *ICASSP*, no. 2, 2015, pp. 4565–4569.

[4] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, "Dropout Improves Recurrent Neural Networks for Handwriting Recognition," *Proceedings of International Conference on Frontiers in Handwriting Recognition, ICFHR*, vol. 2014-Decem, pp. 285–290, 2014.

[5] P. Doetsch, M. Kozielski, and H. Ney, "Fast and robust training of recurrent neural networks for offline handwriting recognition," *International Conference on Frontiers in Handwriting Recognition*, 2014.

[6] K. M. Sayre, "Machine recognition of handwritten words: A project report," *Pattern Recognition*, vol. 5, no. 3, pp. 213–228, 1973.

[7] T. Bluche and R. Messina, "Scan, Attend and Read: End-to-End Handwritten Paragraph Recognition with MDLSTM Attention," pp. 1–10, 2016. [Online]. Available: http://arxiv.org/abs/1604.03286

[8] T. Bluche, "Joint Line Segmentation and Transcription for End-to-End Handwritten Paragraph Recognition," in *NIPS 2016: Neural Information Processing Systems*, 2016.

[9] M. Schall, M.-P. Schambach, and M. O. Franz, "Multi-Dimensional Connectionist Classification: Reading Text in One Step," in *Proceedings - 13th IAPR International Workshop on Document Analysis Systems, DAS 2018*, 2018.

[10] J. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," *ICML '01 Proceedings of the Eighteenth International Conference on Machine Learning*, vol. 8, no. June, pp. 282–289, 2001.

[11] K. Murphy, Y. Weiss, and M. Jordan, "Loopy-belief Propagation for Approximate Inference: An Empirical Study," *15*, pp. 467–475, 1999.

[12] K. P. Murphy, *Machine learning: a probabilistic perspective (adaptive computation and machine learning series)*. MIT Press, 2012.

[13] A. Graves and J. Schmidhuber, "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks," in *Advances in Neural Information Processing Systems 21, NIPS'21*, 2008, pp. 545–552.

[14] A. Graves, "Offline Arabic Handwriting Recognition with Multidimensional Recurrent Neural Networks," *Guide to OCR for Arabic Scripts*, pp. 297–313, 2012. [Online]. Available: http://link.springer.com/10.1007/978-1-4471-4072-6_12

[15] J. Puigcerver, "Are Multidimensional Recurrent Layers Really Necessary for Handwritten Text Recognition?" 2017.

[16] M.-P. Schambach and S. F. Rashid, "Stabilize sequence learning with recurrent neural networks by forced alignment," in *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, 2013, pp. 1270–1274.

[17] G. D. Forney Jr, "The Viterbi Algorithm: A Personal History," *arXiv:cs/0504020*, 2005.

[18] J. Pearl, "Probabilistic Reasoning in Intelligent Systems," p. 552, 1988.

[19] V. Chandrasekaran, N. Srebro, and P. Harsha, "Complexity of inference in graphical models," *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, vol. 2008, pp. 70–78, 2008.

[20] N. Peyrard, M.-J. Cros, S. de Givry, A. Franc, S. Robin, R. Sabbadin, T. Schiex, and M. Vignes, "Exact and approximate inference in graphical models: variable elimination and beyond," pp. 1–47, 2015. [Online]. Available: http://arxiv.org/abs/1506.08544

[21] U. V. Marti and H. Bunke, "The IAM-database: An English sentence database for offline handwriting recognition," *International Journal on Document Analysis and Recognition*, vol. 5, no. 1, pp. 39–46, 2003.

[22] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[23] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011.

[24] Y. N. Dauphin, J. Chung, and Y. Bengio, "RMSProp and equilibrated adaptive learning rates for non-convex optimization," Tech. Rep., 2014.

[25] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[26] R. A. Wagner and M. J. Fischer, "The String-to-String Correction Problem," *Journal of the ACM*, vol. 21, no. 1, pp. 168–173, 1974.

[27] A. Paszke, G. Chanan, Z. Lin, S. Gross, E. Yang, L. Antiga, and Z. Devito, "Automatic differentiation in PyTorch," *31st Conference on Neural Information Processing Systems*, no. Nips, pp. 1–4, 2017.