International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 06 Issue: 05 | May - 2022 Impact Factor: 7.185 ISSN: 2582-3930

A Research on Handwritten Text Recognition

Naman Bhardwaj , Pawan Tanay Tripathi, Avinash Sharma
ABES Institute of Technology

Abstract: Over the years we have started accumulating handwritten documents, like pdfs, doc files and numerous other formats for reading, writing and studying. Often we come across situations where we need to utilize the text of those documents. Manually transcribing large amounts of handwritten data is an arduous process that's bound to be fraught with errors. Automated handwriting recognition can drastically cut down on the time required to transcribe large volumes of text, and also serve as a framework for developing future applications of machine learning. Recognition can be offline or online and both can be implemented in applications to progressively learn based on the user's feedback while performing offline learning on data in parallel. Some recognition systems identify strokes, others apply recognition on a single character or entire words. Some steps involved in the area(in no particular order): Image preprocessing, Segmentation, Classification & Recognition, Feature Extraction. Finally, some limitations of the direction of current and future research are presented.

Keywords: CNN, DTCN, TrOCR, DTCN, RNN

1. INTRODUCTION

Handwritten digit recognition is the ability of a computer to recognize the human handwritten texts from different sources like images, papers, touch screens, etc, and classify them into 10 predefined classes (0-9). This has been a topic of boundless-research in the field of deep learning. Handwriting recognition has many applications like handwritten docs recognition, postal mail sorting, bank check processing, etc. In Handwritten recognition, we face many challenges because of different styles of writing of different peoples as it is not an Optical character recognition. This research provides a comprehensive comparison between different machine learning and deep learning algorithms for the purpose of handwritten digit recognition. For this, we have used Support Vector Machine, Multilayer Perceptron, and Convolutional Neural Network. The comparison between these algorithms is carried out on the basis of their accuracy, errors, and testing-training time corroborated by plots and charts that have been constructed using matplotlib for visualization. The accuracy of any model is paramount as more accurate models make better decisions. The models with low accuracy are not suitable for real-world applications. Ex- For an automated bank cheque processing system where the system recognizes the amount and date on the check, high accuracy is very critical. If the system incorrectly recognizes a digit, it can lead to major damage which is not desirable. That's why an algorithm with high accuracy is required in these real world applications. Hence, we are providing a comparison of different algorithms based on their accuracy so that the most accurate algorithm with the least chances of errors can be employed in various applications of handwritten digit recognition. This paper provides a reasonable understanding of machine learning and deep learning algorithms like SVM, CNN, and MLP for handwritten digit recognition. It furthermore gives you the information about which algorithm is efficient in performing the task of digit recognition. In further sections of this paper, we will be discussing the related work that has been done in this field followed by the methodology and implementation of all the three algorithms for the fairer understanding of them. Next, it presents the conclusion and result bolstered by the work we have done in this paper. Moreover, it will also give you some potential future enhancements that can be done in this field. The last section of this paper contains citations and references used.

1.1 Deep Learning:

Two or two artificial neural networks can readily be defined as deep learning architectures (ANNs). More hidden layers are being added in order to improve prediction accuracy. DL employs more hidden layers than standard artificial neural networks. Weighting is used in classic deep neural networks (DNN). To obtain the output 18, the bias-corrected input value is run through a nonlinear activation function such as ReLu or a Softmax function. As a result, the goal of DNN training is to optimize the network weights so that the loss function is as little as possible. The higher level characteristic is defined using each lower level feature. Speech recognition 19, 20, image analysis 21, text mining 22, health monitoring 26, 24, drug discovery 25, computer vision 27, object identification 28, and deep learning model technology 29-32 are all examples of deep learning technology applied in HCS applications. It has a wide range of applications.

1.1.1. Convolutional Neural Network

Convolutional Neural Networks (CNN) is a deep learning architecture that is supervised. It's primarily used in image analysis software. A convolutional layer, a pooling layer, and a fully connected layer are the three layers that make up a CNN. The input image is passed through the kernel or filters in the convolution layer to generate various feature maps.

To keep the number of weights low, the pooling layer shrinks each feature map. Downsampling or subsampling is the term for this procedure. Global pooling, maximum pooling, and average pooling are three different forms of pooling procedures. After these layers, the 2D feature map is transformed into a 1D vector and then classified using a fully connected layer. The CNN method for classifying images is shown in Figure 1.

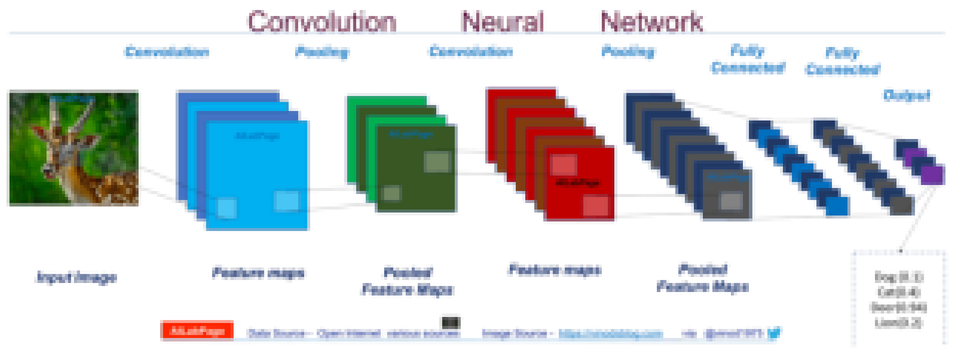


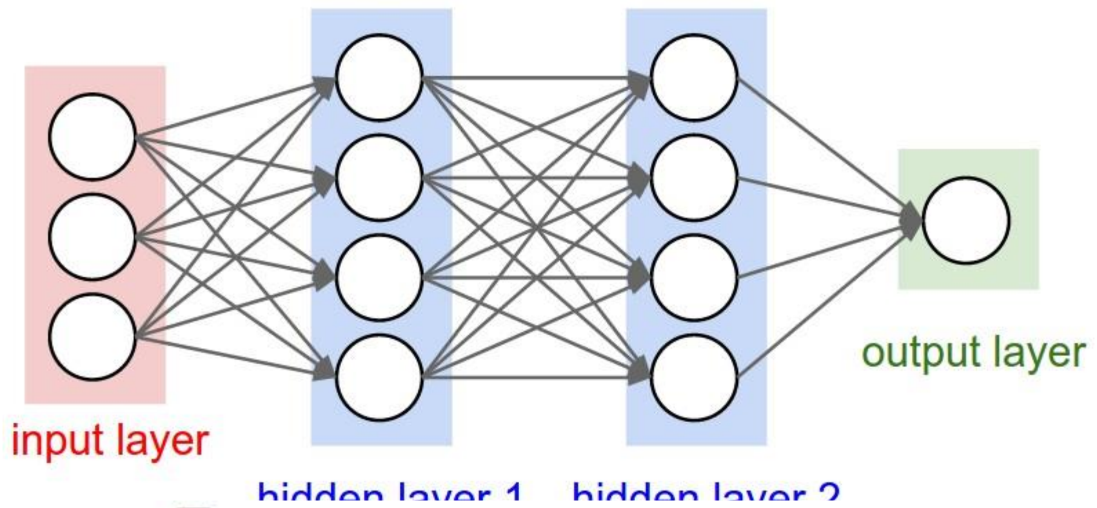
Figure. 1: Adapted to 35 CNN architecture with two folding layers. A pooling / subsampling layer follows each convolution layer. The completely linked layer and the final output layer receive the output of the last pooling layer ID. Figure 1: A CNN architecture with two folding layers is depicted. A pooling / subsampling layer follows each convolution layer. The last pooling layer's output is fully connected. supply Layer 35 is the final output layer. Deepr, a new end-to-end DL for extracting essential features from medical records and predicting anomalies, was proposed in 36. To forecast unplanned restart after a discharge, convolutional neural networks are used to a set of discrete elements. Then Cheng [1] used DL technology to investigate the temporal aspects of the patient's electronic health record. The convolution operator was conducted in the temporal dimension of the patient's EHR matrix in the second layer of the proposed DL. The model employs temporal fusion tactics such as early fusion, late fusion, and delayed fusion to achieve leverage. In the learning process, the EHR's temporal smoothness.

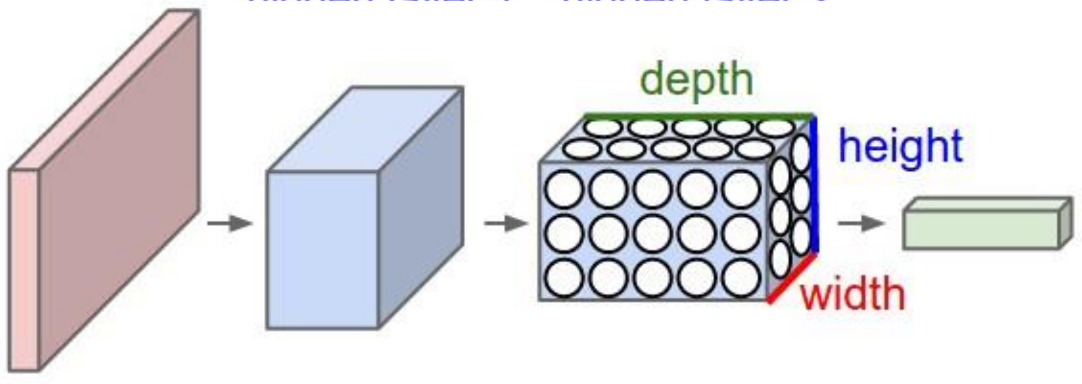
Subsampling layer follows each convolution layer. The last pooling layer's output is fully connected. supply Layer 35 is the final output layer. Deepr, a new end-to-end DL for extracting essential features from medical records and predicting anomalies, was proposed in 36. To forecast unplanned restart after a discharge, convolutional neural networks are used to a set of discrete elements. Then Cheng [1] used DL technology to investigate the temporal aspects of the patient's electronic health record. The convolution operator was conducted in the temporal dimension of the patient's EHR matrix in the second layer of the proposed DL. The model employs temporal fusion tactics such as early fusion, late fusion, and delayed fusion to achieve leverage. In the learning process, the EHR's temporal smoothness.

Architecture Overview

Normally the neural nets are not able to fully scalable the full-sized images. Like in CIFAR-10, images with the dimensions of 32X32X3 (32 wide, 32 high, 3 color channels), hence any fully connected singular neuron will have these dimensions in their 1st hidden layer - 32*32*3 =3072 weights. So we can say that although the above amount looks to be somehow processable it clearly is not fit to scale large images. Like an image of dimensions - 200 X 200 X 3, will have neurons with these many weights - 200*200*3 = 120,000, also to add up we would need numerous neurons similar to this so as to sum up the parameters in a hasty way. As one can easily decipher from this, it will lead up to overfitting due to numerous parameters rendering this full connectivity redundant.

3D volumes of neurons. The fact that the CNN takes only input as images, this limits the architecture but in a feasible manner. A regular CNN has layers arranged in the 3 dimensional way like - width X height X depth, which is quite different from a conventional neural network. Also, the dimension of the ultimate output for CIFAR-10 has this dimension - 1X1X10, as the full-sized image gets reduced to a single vector consisting of class scores by the end of CNN architecture which is arranged along its depth dimension. Following visualization shows this:





Left: A 3-layer Neural Network, Right: A CNN which has its neurons arranged in the following manner: width, height, a depth which is depicted over a single layer. In the CNN, each layer converts the three-dimensional input volume to the volume of neuron activations. Like here, the red layer holds the image, hence its dimension in terms of its height & width will be the same as that of the image while the depth would be 3 (Red, Green, Blue).

A CNN consists of various layers where each layer is a straightforward functionality: It converts the given input 3D volume to an output 3D volume that might have any/no parameters.

Layers used to build ConvNets

As discussed above, a simple CNN consists of layers sequence and each layer of it converts a volume of activations to another with the help of a differentiable function. Here we utilize 3 main types of layers to construct CNN architectures: Convolutional Layer, Pooling Layer & Fully-connected layer (just like that of a regular Neural Network). All these layers are then put on top of each other to create a CNN architecture.

Example - A simple straight-forward CNN for CIFAR-10, classification consists of the architecture [INPUT - CONV - RELU - POOL - FC], to explain this-

- The Input [32X32X3] would have the raw pixel values of that image, in our case the image's having following dimensions - (width -32 X height 32) & with 3 color channels - R, G, B.
- CNN layers will have to calculate the neuron's output which are joined to local regions in that input. The Result would be an output [32X32X12] in case of 12 filters.
- The RELU layer is applied on element-wise activation functions like the max(0,x) which thresholds at 0, this makes the changed dimension of volume unchanged ([32X32X12]).
- POOL layer is used to perform a downsampling operation which will result in volume like: [16X16X12].
- FC is used to calculate the class scores, which ultimately makes the volume [1X1X10], as each layer has 10 numbers corresponding to a single class like among the 10 categories of CIFAR-10. Similar to neural networks as each neuron of this layer is connected to all of its in previous volumes.

Similarly, CNN converts the original image layer from the native image layer as compared to original pixel values to the final class scores. We can see that layers have parameters which others don't have. The CNN layers do the transformations which are a function of the activations of the input volume & also of the parameters (the weights & the biases of the neurons). On one hand, the RELU/POOL layers will have a fixed function. The parameters of the CNN layers, needs to be trained with gradient descent leading to class scores that the CNN is calculated are consistent with those labels in that of the training set of each image.

In summary:

- A CNN architecture is the most simple case of list of layers that transforms the image volume into an output volume (ex- holding the class scores)
- There are different types of layers (ex- CONV/FC/RELU/POOL are the most famous ones)
- At every layer takes an input 3D volume which transforms it to a 3D volume output by a differentiable function.
- At each layer will or will not have parameters (ex- conv/fc and not relu/pool)
- At each layer will or will not have additional hyperparameters (ex- conv/fc/pool do, relu doesn't)

Convolutional Layer

The Conv layer is the core building block of a Convolutional Network that does most of the computational heavy lifting.

Let's first discuss what the CONV layer computes without brain/neuron analogies. The CONV layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a ConvNet might have size 5x5x3 (i.e. 5 pixels width and height, and 3 because images have depth 3, the color channels). During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. Now, we will have an entire set of filters in each CONV layer (e.g. 12 filters), and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.

The brain view. If you're a fan of the brain/neuron analogies, every entry in the 3D output volume can also be interpreted as an output of a neuron that looks at only a small region in the input and shares parameters with all neurons to the left and right spatially (since these numbers all result from applying the same filter).

We now discuss the details of the neuron connectivities, their arrangement in space, and their parameter sharing scheme.

Local Connectivity. When dealing with high-dimensional inputs such as images, as we saw above it is impractical to connect neurons to all neurons in the previous volume. Instead, we will connect each neuron to only a local region of the input volume. The spatial extent of this connectivity is a hyperparameter called the receptive field of the neuron (equivalently this is the filter size). The extent of the connectivity along the depth axis is always equal to the depth of the input volume. It is important to emphasize again this asymmetry in how we treat the spatial dimensions (width and height) and the depth dimension: The connections are local in 2D space (along width and height), but always full along the entire depth of the input volume.

Example 1. For example, suppose that the input volume has size [32x32x3], (e.g. an RGB CIFAR-10 image). If the receptive field (or the filter size) is 5x5, then each neuron in the Conv Layer will have weights to a [5x5x3] region in the input volume, for a total of 5*5*3 = 75 weights (and +1 bias parameter). Notice that the extent of the connectivity along the depth axis must be 3, since this is the depth of the input volume.

Normalization Layer

Many types of normalization layers have been proposed for use in ConvNet architectures, sometimes with the intention of implementing inhibition schemes observed in the biological brain. However, these layers have since fallen out of favor because in practice their contribution has been shown to be minimal, if any. For various types of normalizations, see the discussion in Alex Krizhevsky's cuda-convnet library API.

Fully-connected layer

Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset. See the Neural Network section of the notes for more information.

Converting FC layers to CONV layers

It is worth noting that the only difference between FC and CONV layers is that the neurons in the CONV layer are connected only to a local region in the input, and that many of the neurons in a CONV volume share parameters. However, the neurons in both layers still compute dot products, so their functional form is identical. Therefore, it turns out that it's possible to convert between FC and CONV layers:

- For any CONV layer there is an FC layer that implements the same forward function. The weight matrix would be a large matrix that is mostly zero except for at certain blocks (due to local connectivity) where the weights in many of the blocks are equal (due to parameter sharing).
- Conversely, any FC layer can be converted to a CONV layer. For example, an FC layer with
- K=4096 that is looking at some input volume of size- 7×7×512 can be equivalently expressed as a CONV layer with- F=7,P=0,S=1,K=4096. In other words, we are setting the filter size to be exactly the size of the input volume, and hence the output will simply be- 1×1×4096 since only a single depth column "fits" across the input volume, giving identical results as the initial FC layer.

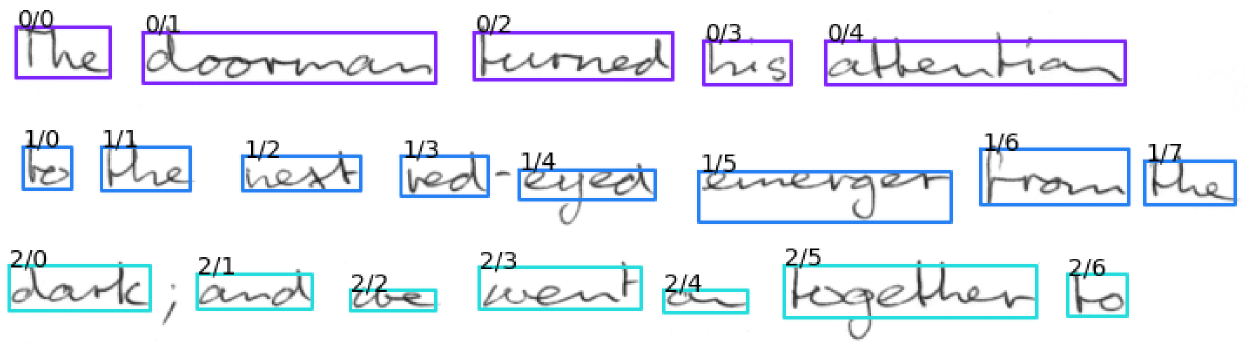
1.2 Dilated temporal convolution network

We use a Dilated temporal convolution network structure, as proposed by Lea et al. (2017), to extract the time sequenced features from a temporal sequence. Fig. 1 illustrates a schematic representation of DTCN. In abstract form, it can be understood as a structure of B sequential blocks of L layers each. The layers in each block perform 1-D causal convolutions with a filter f with a dilation of $2d$, where d is the layer number in the block structure. These dilations allow a much wider receptive field at the top layer in the block. Furthermore, this fully convolutional structure enables a high level of parallelized computations in the structure (Bai et al., 2018; Lea et al., 2017). In the generic TCN network, each layer is implemented as a residual block consisting of 1-D convolutions with filters f , nonlinear activation and residual connection. This helps in stabilizing the network training for deeper networks (Bai et al., 2018). The same number of filters f enables to combine the input and output of each layer with residual connection. The output of DTCN is a sequence with the same number of time steps as the input. The number of output feature maps are the same as the number of filters f at the final block of the DTCN structure. As shown in Fig. 1, the outputs of blocks are also combined with a skip connection. This not only avoids the vanishing gradient in the lower layers of the network structure, but also allows faster training convergence without any additional parameters (Lea et al., 2017). In contrast to the generic DTCN, we use a DTCN architecture with non-causal convolutions. This allows layers to have bi-directional access to features from previous layers/input.

Our approach for detecting words

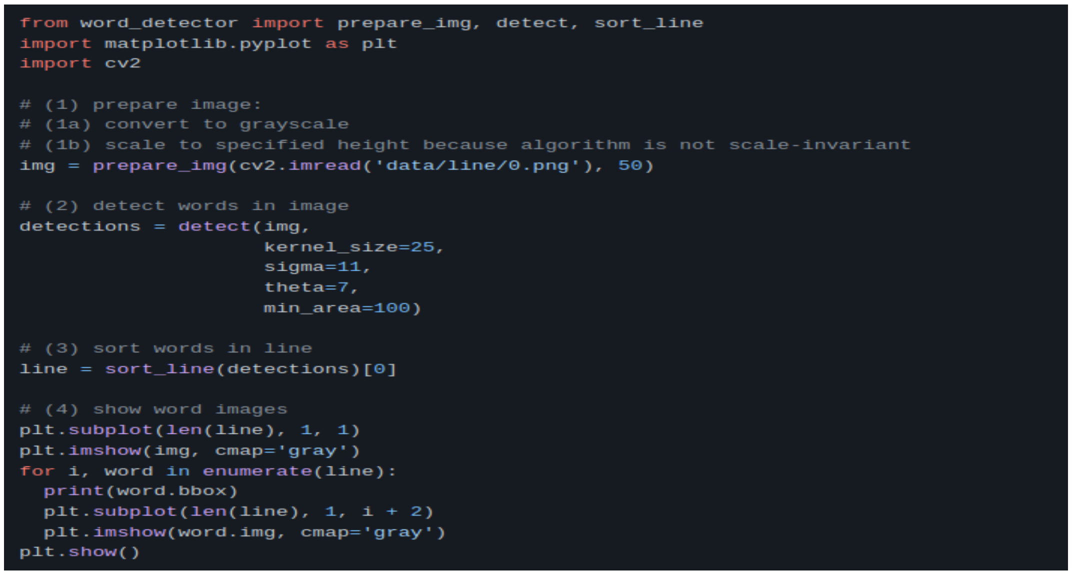
For our handwritten text recognition through CNN we are using our word recognition through space recognition in between the words segregating each word by word and then passing through the trained cv2 algorithms to determine each word present in the line.

- Implementation of the scale space technique for word segmentation proposed by R. Manmatha and N. Srimal. Even though the paper is from 1999, the method still achieves good results, is fast, and has a simple implementation.
- The algorithm takes an **image containing words as input** and **outputs the detected words**. Optionally, the words are sorted according to reading order (top to bottom, left to right).



Usage -

- This example loads an image of a text line, prepares it for the detector (1), detects words (2), sorts them (3), and finally shows the cropped words (4)-



```
from word_detector import prepare_img, detect, sort_line
import matplotlib.pyplot as plt
import cv2

# (1) prepare image:
# (1a) convert to grayscale
# (1b) scale to specified height because algorithm is not scale-invariant
img = prepare_img(cv2.imread('data/line/0.png'), 50)

# (2) detect words in image
detections = detect(img,
                    kernel_size=25,
                    sigma=11,
                    theta=7,
                    min_area=100)

# (3) sort words in line
line = sort_line(detections)[0]

# (4) show word images
plt.subplot(len(line), 1, 1)
plt.imshow(img, cmap='gray')
for i, word in enumerate(line):
  print(word.bbox)
  plt.subplot(len(line), 1, i + 2)
  plt.imshow(word.img, cmap='gray')
plt.show()
```

The package contains the following functions -

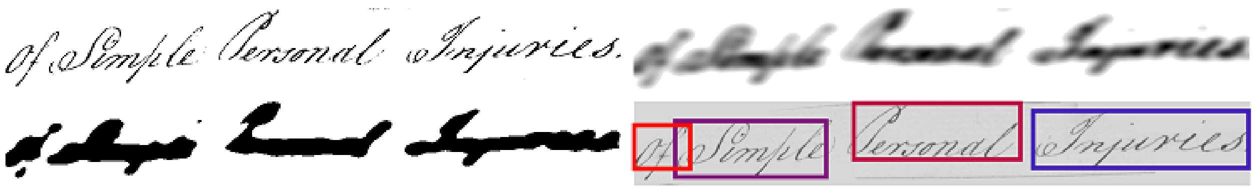
1. prepare_img: prepares input image for detector
2. detect: detect words in image
3. sort_line: sort words in a (single) line
4. sort_multiline: cluster words into lines, then sort each line separately

For more details on the functions and their parameters use help(function_name), e.g. help(detect).

Algorithm

The illustration below shows how the algorithm works:

- top left: input image
- top right: apply filter to the image
- bottom left: threshold filtered image
- bottom right: compute bounding boxes



How to select parameters -

- The algorithm is not scale-invariant
  - The default parameters give good results for a text height of 25-50 pixels
  - If working with lines, resize the image to 50 pixels height
  - If working with pages, resize the image so that the words have a height of 25-50 pixels
- The sigma parameter controls the width of the Gaussian function (standard deviation) along the x-direction. Small values might lead to multiply detection per word (over-segmentation), while large values might lead to a detection containing multiple words (under-segmentation)
- The kernel size depends on the sigma parameter and should be chosen large enough to contain as much of the non-zero kernel values as possible
- The average aspect ratio (width/height) of the words to be detected is a good initial guess for the theta parameter

The best way to find the optimal parameters is to use a dataset (e.g. IAM) and optimize the parameters w.r.t. some evaluation metric (e.g. intersection over union).

Results of this approach -

This algorithm gives good results on datasets with large inter-word-distances and small intra-word-distances like IAM. However, for historical datasets like Bentham or Ratsprotokolle results are not very good and more complex approaches should be preferred (e.g., a neural network based approach as implemented in the WordDetectorNN).

Approach for recognising words

We consider the handwritten image as a sequence of data with an imaginary time axis in the direction of writing. Thus, for both English and French handwritten images the time axis is left to right. Firstly, the input image is converted to grayscale. Let the grayscale image and label pair be represented as ($\mathrm{I}m\times n\times 1$, $\mathrm{L}l$). The image is considered as a sequence of n time steps with m features and the label as a sequence of length l corresponding to the transcription of the image using a character set of $numClass$ different symbols. The spatial arrangement of pixels gives a very distinguishing feature for the handwritten characters. We first used a convolution block to extract the spatial features. The output feature map from this block is as follows :

$\mathrm{F}sp\times q\times r = conv\_block(\mathrm{I}m\times n\times 1)$

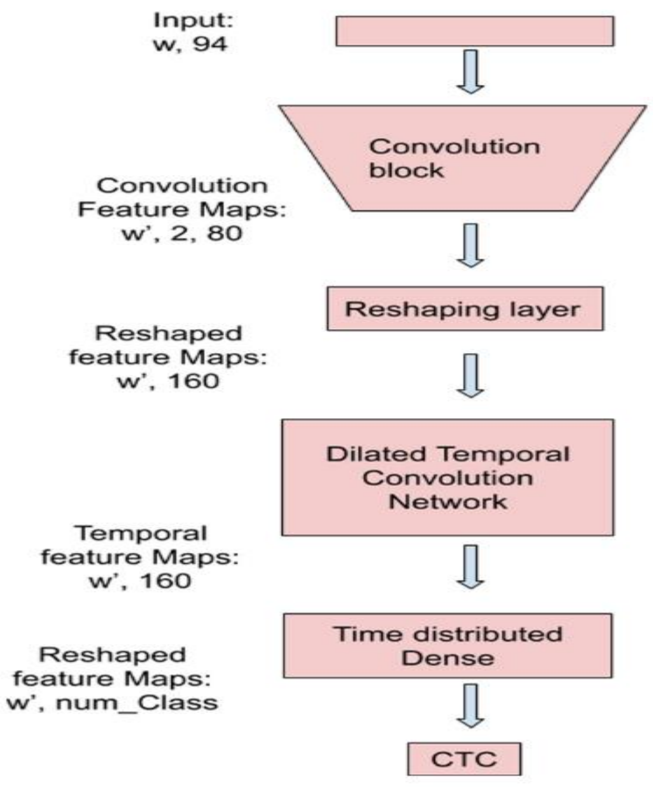

Input:
w, 94
Convolution block
Convolution Feature Maps: w', 2, 80
Reshaping layer
Reshaped feature Maps: w', 160
Dilated Temporal Convolution Network
Temporal feature Maps: w', 160
Time distributed Dense
Reshaped feature Maps: w', num_Class
CTC

Fig. 2. Architecture of the proposed model. The dimensions are shown for an input image of height 84. This brings height of the input to 94 pixels along with the padding.

where, p is the height dimension and $p \leq m$; q is the width dimension and $q \leq n$; r is the number of feature maps from the last convolution operation in the module. The convolution block is composed of an arrangement of multiple 2D convolutions and max pooling layer. The details of this block are shown in Section 5. In order to extract the temporal features, from these output feature maps $\mathrm{F}p\times q\times rs$, we rearranged it into a temporal sequence by a flattening operation on height and width axis. We represent the rearranged temporal sequence as $\mathrm{T}t\times rs$, where t is $p \times q$. The feature maps thus calculated by the convolution block are the input to the DTCN block. The architectural details of the DTCN are already shown in Section 3. DTCN block output has the same number of time steps as input. This can be represented as:

$\mathrm{T}t\times f$

$t = DT\ CN\_block(\mathrm{T}st\times r)$ (2)

These temporal features are then used by the CTC output layer for the sequence learning task. A dense layer, with $numClass + 1$ units, is used

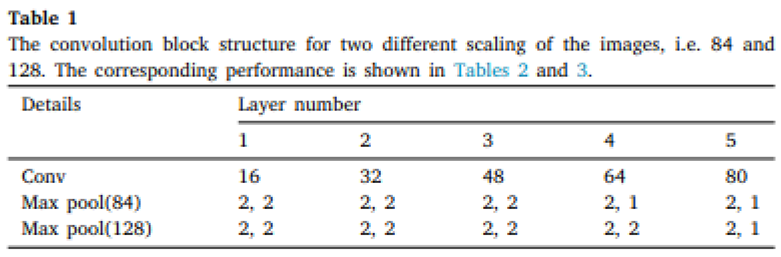


Table 1
The convolution block structure for two different scaling of the images, i.e. 84 and 128. The corresponding performance is shown in Tables 2 and 3.

Details	Layer number				
	1	2	3	4	5
Conv	16	32	48	64	80
Max pool(84)	2, 2	2, 2	2, 2	2, 1	2, 1
Max pool(128)	2, 2	2, 2	2, 2	2, 2	2, 1

t each time step of $\mathrm{T}t\times ft$. This operation is required to bring the time sequenced features in the form suitable to process through a CTC layer for sequence learning task. The CTC gives out the probability of a label (out of $numClass$ symbols from the character set and a blank symbol) be present at a time step in the sequence.

Dataset

For the experiments in this work, we have considered the IAM dataset, HW-AES dataset for English and RIMES dataset for French handwriting recognition. All these dataset have handwritten line images with unconstrained handwriting along with the transcripts for each image. The character set is composed of language specific alphabets and special symbols. IAM lines dataset composed of 6161 line images in training, 976 lines in testing and 2915 lines in validation dataset (Marti & Bunke, 2002). We have used the same partition of the dataset as used by Puigcerver (2017). It consists of 79 characters including the space character. The RIMES dataset is a french handwriting dataset composed of 11 333 lines in training and 778 in test dataset (Grosicki et al., 2009). The original release of the dataset does not mark the explicit validation set. We have used the same partition of the dataset as used by Puigcerver (2017) to have 10% images from training set to be used for validation. This partitions the dataset into training, validation and test images of 10 203, 1130 and 778 lines, respectively. It has 99 different characters. We further used our in-house collected dataset Handwritten Automated Essay Scoring(HW-AES) from 70 writers. HW-AES dataset is the collection of handwritten English essays based on ASAP-AES1 competition dataset (Sharma & Jayagopi, 2018a) The dataset has line level transcriptions available for 122 essays and score (in the range 0–4) assigned by a human grader for each essay. We have used the handwritten line images and corresponding transcriptions for the hand-writing recognition task. The images of hand writings are available as a collection of 1208, 404, 401 line images for training, validation and testing, respectively. The dataset has 81 characters including the space character.

Experiments and results

A set of experiments was conducted to analyze various aspects of our model and handwritten data. We considered scaling images with different factors and compared the model performance. We iteratively increased network size to come up with a trade-off between network size and performance. We also tried to include a data augmentation pipeline to see the effect on network performance. Model performance is measured in terms of CER, training and testing time. All these experiments are detailed below.

Scaled images for training

The raw line images in the dataset are of variable size. Current works scale images to have the same height and use padding on the width axis to have same-size images. We experimented with scaling the images with different ratios. Tables 2 and 3 summarizes the results for the model trained with scaled image height of 84 and 128, respectively.

As opposed to RNN encoder decoder network with attention where the recognition results with CER as the measure does not differ much with scaled height (Chowdhury & Vig, 2018), our model is very sensitive to the scaling of the images. The number of parameters is not sensitive to image height but blocks and layers in the DTCN module. Therefore, our model with the same number of blocks and layers has the same parameters in Tables 2 and 3.

Network size

In order to understand the sequence learning abilities of the DTCN, we experimented with different numbers of blocks and layers in TCN structure while maintaining the same 2-D convolution module. Table 2 summarizes experiment results with 4 layers and a varying number of blocks and scaled image height of 84. The 4 layers in each block are chosen empirically. The model without data augmentation pipeline is shown as a baseline model with input image height mentioned in brackets. The same model structure with added data augmentation is also shown in both the tables with scaled height of the images mentioned in brackets. The details of the data augmentation pipeline are shown in Section 7.3. The performance of the model is compared with CER and WER as the measure. Table 3 shows performance with models trained with a scaled height of 128. Tables 2 and 3 show the model performance with data augmentation shows less CER than the baseline model. This is true for models trained with images with scaled heights of 84 and 128. Furthermore, a comparison of tables shows that increasing the image height from 84 to 128 helps train a better model.

Effect of data augmentation

We added simple data augmentation in the training pipeline to experiment with network generalization. We incorporated rotation, shear and translation for augmenting the data. The parameters for these operations were drawn from a fixed distribution. The results are shown in Tables 2 and 3 for scaled image height of 84 and 128, respectively. Baseline model performance is shown in the top row and with data augmentation pipeline in bottom row. A comparison of baseline models with corresponding versions with data augmentation shows that data augmentation helps improve both CER and WER performance. This is true for all three datasets under review. For HW-AES and RIMES dataset, the model trained with images of scaled height 128 performs better than the one with a scaled height of 84. Unlike this observation, IAM dataset shows slightly better results with smaller images i.e. scaled height 84 though the difference is 0.1% in CER and 0.4% in WER. Tables 2 and 3 also show that data augmentation, in general, enhances the performance of the corresponding model although for a deeper model with 9 blocks the improvement is less.

Table 4
Comparison with existing literature on IAM lines test dataset. The best performing model from Tables 2 and 3 is selected for the analysis. All the results are presented here without language/lexicon model.

Model	CER (%)	WER (%)
Voigtlaender et al. (2016)	8.3	27.5
Puigcerver (2017)	6.2	20.2
Krishnan et al. (2018)	9.7	32.8
Chowdhury and Vig (2018)	8.1	16.7
Ours (image height = 84, with data augmentation B = 6, L = 4)	7.9	23.0

Table 5
Comparison with existing literature on RIMES lines test dataset. The best performing model from Tables 2 and 3 is selected for the analysis. All the results are presented here without language/lexicon model.

Model	CER (%)	WER (%)
Voigtlaender et al. (2016)	3.5	9.3
Puigcerver (2017)	2.6	10.7
Chowdhury and Vig (2018)	3.5	9.6
Ours (image height = 128, with data augmentation B = 6, L = 4)	6.0	15.3

Transcription results and efficiency

Tables 4 and 5 compare our proposed best performance model with literature on IAM and RIMES dataset. For IAM data, our best model is from Table 2 with 6 blocks and 4 layers. This model features a data augmentation pipeline during training and accepts input images of height 84. Different works present results with and without language/lexicon models. To fairly compare the capabilities of different network architectures, we used no language or lexicon model. Our model shows lower CER than MDLSTM (Voigtlaender et al., 2016) and hybrid CNN-RNN (Chowdhury & Vig, 2018; Krishnan et al., 2018) models without language models. The CNN-RNN model by Puigcerver (2017) is superior to our model in terms of both CER and WER. Table 6 compares various approaches to handwriting recognition in terms of CER, WER, per epoch training time and number of parameters. In this comparison, we consider models trained with raw handwritten line images and without post processing by language/lexicon model. Here, we have used MDLSTM based model (Voigtlaender et al., 2016), state-of-the-art CNN-RNN structure (Puigcerver, 2017) and our best baseline model (marked with * in Tables 2 and 3) for comparison. We refer to our model as a CNN-DTCN architecture.

Conclusion

In this paper, we presented a CNN and dilated TCN-based deep network for offline handwriting recognition. The model structure is refined empirically. We further showed experimental results including data augmentation and image downsampling/scaling. The work showed that the proposed CNN-DTCN structure along with CTC cost function enables faster handwriting recognition. Our model has less parameters and takes less training and testing time as compared to the state-of-the-art architectures which are mainly recurrent networks, viz MDLSTM, CNN-LSTM etc. This in turn could lead to not only less carbon footprint but also better experience model use for low resource devices. An analysis of baseline models is also performed to show that the recognition performance is better than the other state-of-the-art methods without data augmentation. Though the model with data augmentation does not beat the recent state-of-the-art results, extensive study and experiments can be investigated with TCN and its variants to enable a faster and robust handwriting recognition.

References

1. Cheng Y, Wang F, Zhang P, Hu J. Risk prediction with electronic health records: A deep learning approach. Proceedings of the 2016 SIAM International Conference on Data Mining: SIAM; 2016. p. 432-40.

2. Arindam Chowdhury, Lovekesh Vig. An Efficient End-to-End Neural Model for Handwritten Text Recognition arXiv:1807.07965v2 [cs.CL] 26 Jul 2018.

3. 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.

4. 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.

5. 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.

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

7. 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.

8. 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.

9. Vu Pham, Théodore Bluche, Christopher Kermorvant, et 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.

10. Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016.

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

12. Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014

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

14. Annapurna Sharma, Dinesh Babu Jayagopi. Towards efficient unconstrained handwriting recognition using Dilated Temporal Convolution Network ISSN: 0957-4174.

15. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems (pp. 1097–1105).
16. R. Manmatha, Nitin Srimal Scale Space Technique for Word Segmentation in Handwritten Documents (pp. University of Massachusetts, Amherst 1999) URL: http://ciir.cs.umass.edu/pubfiles/mm-27.pdf
17. Word Segmentation github repo: https://github.com/githubharald/WordDetector

18. Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In Y. Bengio, & Y. LeCun (Eds.), 3rd international conference on learning representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, conference track proceedings. URL: http://arxiv.org/abs/1409.1556.

19. Puigcerver, J. (2017). Are multidimensional recurrent layers really necessary for handwritten text recognition?. In 2017 14th IAPR international conference on document analysis and recognition (ICDAR), Vol. 01 (pp. 67–72). http://dx.doi.org/10.1109/ ICDAR.2017.20.

20. Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249–256)

21. Graves, A., & Schmidhuber, J. (2009). Offline handwriting recognition with multidimensional recurrent neural networks. Advances in neural information processing systems (pp. 545–552).

22. Sharma, A., & Jayagopi, D. B. (2018a). Automated grading of handwritten essays. In 2018 16th international conference on frontiers in handwriting recognition (ICFHR) (pp. 279–284). http://dx.doi.org/10.1109/ICFHR-2018.2018.00056.

23. Minghao Li , Tengchao Lv, Lei Cui , Yijuan Lu, Dinei Florencio, Cha Zhang , Zhoujun Li, Furu Wei. TrOCR: Transformer-based Optical Character Recognition with Pre-trained Models arXiv:2109.10282v3 [cs.CL] 25 Sep 2021.

24. Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. arXiv preprint arXiv:1508.07909

25. Batuhan Balci, Dan Saadati , Dan Shiferaw Handwritten Text Recognition using Deep Learning http://cs231n.stanford.edu/reports/2017/pdfs/810.pdf

26. Tesseract Model: https://github.com/tesseractocr/tesseract/wiki/TrainingTesseract-4.00

27. Fathma Siddique, Shadman Sakib, Md. Abu Bakr Siddique Recognition of Handwritten Digit using Convolutional Neural Network in Python with Tensorflow and Comparison of Performance for Various Hidden Layers..

26. Y. LeCun, "The MNIST database of handwritten digits," http://yannlecun.com/exdb/mnist/, 1998.