Recognising and Extracting Data From Handwriting

AntWorks White Papers

AntWorks

Index

Since the widespread introduction of office computing and desk-top publishing from the late 1980s enterprise documents have tended to be printed. However, many businesses still offer the option to complete documents by hand, which means that enterprises must still process handwriting.

Some examples include:

- Forms that required a 'wet signature' for compliance, typically requiring a handwritten application rather than online submission (e.g., insurance claims, checks, account opening forms)
- Medical forms, such as prescriptions or reports, filled in by nursing staff or doctors.
- Government forms, legal documentation.
- Legacy handwritten texts, historic documents in libraries in need of preservation.

This paper explores the challenges of recognizing and extracting handwritten content from documents.

1. What do we mean by the Recognition and Extraction of Handwritten Data?

Handwriting recognition and extraction is a category of Machine Learning designed to detect and extract handwritten letters and convert them to their equivalent digital form. Doing this manually, essentially reading handwritten text and typing it into a system, if often both laborious and error prone. To automate the process, algorithms needed to be designed that can:

A) Perform pattern matching to identify handwritten letters despite the wide diversity of human letter formation, spacing differences, and anomalies of textual representation.

B) Disambiguate unclear characters based on context and past feedback. This is analogous to the way our brains adapt to a new handwriting style and can read it better over time.

Given the nature of problem and the breadth of the problem space, it would be unreasonable, certainly at this stage, to expect to achieve the same results with handwriting as conventional OCR systems achieve with standard typeface. However intelligent handwriting recognition and extraction can deliver good outcomes by deploying a multi-stage ensemble of sophisticated algorithms.

When we talk about handwriting, we mean text that is written in block letters and cursive form.

A) Texts in manuscript style are easier to recognise as the characters are written separately as block letters.

B) With cursive handwriting characters are joined as they are written. This means that a handwriting recognition algorithm needs to be able to distinguish each separate character correctly and identify them accurately.

2. Challenges in recognising and extracting handwriting

2.1. Nature of content

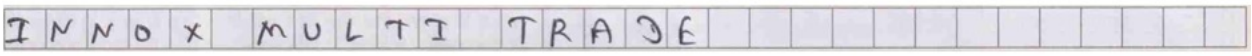
- The way that strokes are used to form letters varies hugely from person to person. They're often indistinct. An individual's handwriting style also varies. Humans are inconsistent in a way that machines aren't.
- Document images containing handwriting are typically of poorer quality that those that are printed.
- Text in printed documents is laid out in a straight line whereas handwriting tends to be skewed and people vary the angle at which they write.
- Handwritten text is often rotated to the right whereas printed text all sits up straight.

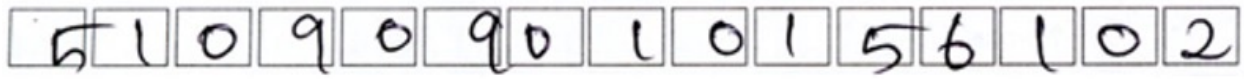
2.2. The surrounding environment

Forms designed to be filled in by hand often provide lines or boxes to guide the writer. While this helps organise the text better it also adds to the challenges for machine reading.

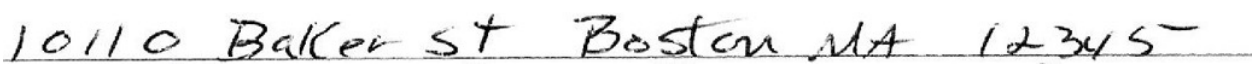
For example:

2.2.1. Content surrounded by boxes

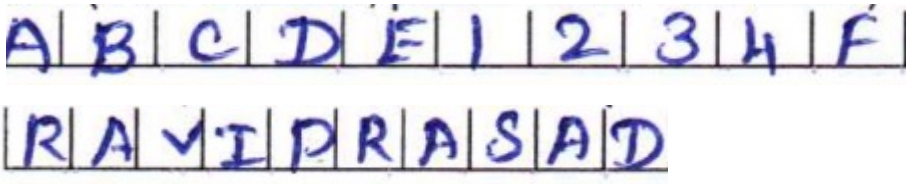




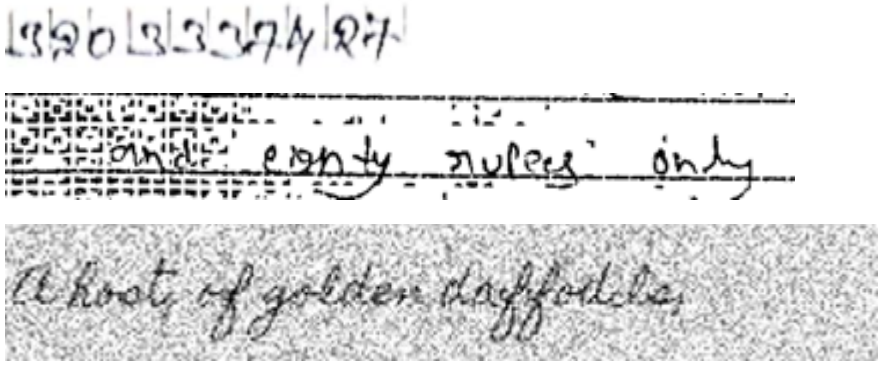
2.2.1. Content surrounded by boxes



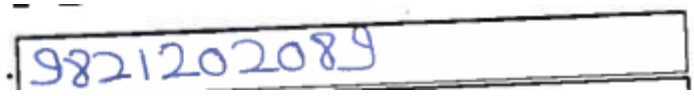
2.2.3. Content surrounded by half box



2.2.4. Content obscured by noise



2.2.5. Content present skewed to the horizontal



2.2.6. Light fonts or distorted content

Believing what we don't believe
Does not exhilarate.

3. How intelligent recognition and extraction of handwriting works

In general, deep learning requires a lot of training data and finding a huge body of labelled handwriting images in different languages is a huge task. To avoid the workload this would require, we use Generative Adversarial Networks to generate training data, and the algorithms deliver substantial accuracy even with smaller data sets.

4. Process Architecture

The process is executed as a blend of two modules.

4.2.1. Identifying the region of interest

It is the process of detecting the region or space of the document from which the text is to be extracted. Region detection is done as a separate process either through image alignment or pattern-based search.

Image alignment is relevant for structured documents documents where data stays at the same position on every document. Structured documents are made up of clearly defined forms with fields that are always in the same place. The only change is the information populated in each field while the document structure is always the same.

Image alignment is achieved by the process of anchor-based registration and text segmentation. Anchor-based image registration is a digital image-processing technique that helps to align different images to a same plane. The process aligns the input image to the same angle as a reference image. The module helps in warping the input image so that the features in the two images line up perfectly. Text segmentation is used when anchor-based image registration ROI alignment fails due to the semi-structured format where data organised to some degree. This degree is typically achieved with some form of tags or other elements with defined properties that introduce a hierarchy and a system into a file. However, the order and number of such structuring tags and elements may differ. Segmentation is the task of breaking the whole image into subparts in order to process them further. The approach used is line level as word or character level will not yield useful results due to background noise or underlinings.

Pattern search is applicable for unstructured documents where data follows a free-form format and therefore there is no predefined structure. The pattern search algorithm searches for configured patterns and taxonomy variations of the patterns. Patterns that fulfill the selection criteria are considered for content identification. The content identification searches the area proximate to the pattern being sought. The content from the selected proximate area is curated and retrieved.

4.2.2. Digitising Handwriting

Digitisation is the process of extracting the data present in the region.

Figure 1: Process Overview

5. AntWorks handwriting extraction engine

5.1. Model Overview

We use an artificial neural network to implement the extraction algorithm. It consists of a convolutional neural network (CNN) layer, recurrent neural network (RNN) layers, and a final Connectionist Temporal Classification (CTC) layer. The model developed by AntWorks consists of 5 CNNs (feature extraction), a pair of RNN layers and a CTC layer (calculate the loss).

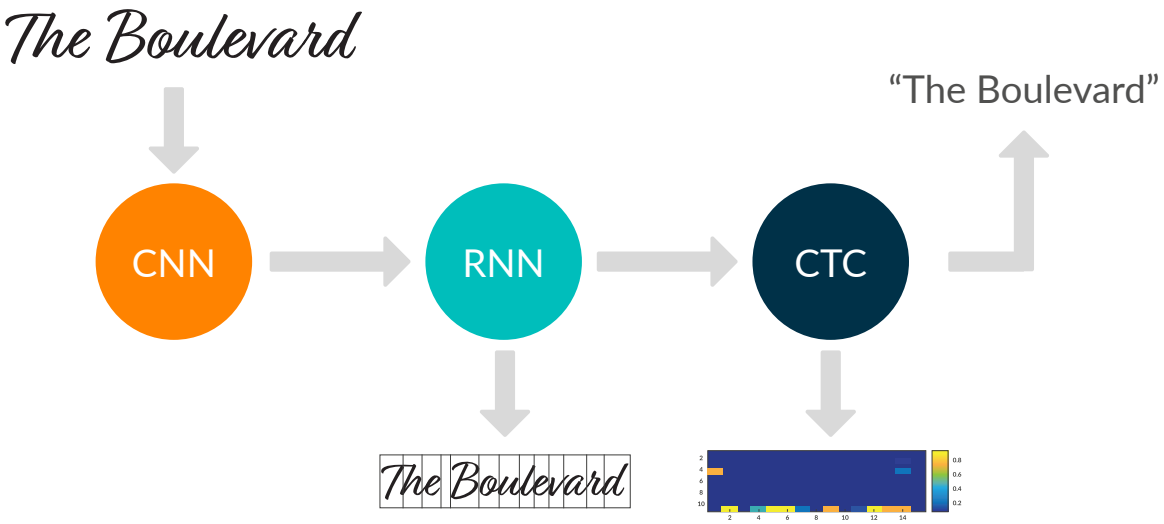


Figure 2: Model Overview

5.2. Model

5.2.1. Convolutional Neural Network (CNN)

The input image is given to the CNN layers. These layers are trained to take out relevant features from the image. Each layer consists of three operations. First, the convolution operation; the input passes through a 5×5 filter in the first two layers and a 3×3 filter is used in the last three layers. Then, the non-linear ReLU (Rectified Linear Unit) function is applied. Finally, a pooling layer recapitulates image regions and outputs a downsized (smaller) version of the input. While the height of image size is condensed by 2 in each layer, feature channels are added, so that the output feature sequence has a size of 32×256.

5.2.2. Recurrent Neural Network (RNN)

The feature sequence consists of 256 features per time-step. The RNN passes context information through this sequence. The chosen Long Short-Term Memory (LSTM) implementation of RNNs is employed because it can transmit information over longer distances and provides more robust training-characteristics than vanilla RNN. The RNN output sequence is mapped to a matrix of 32×80. We use an IAM data-set format as a gold standard corpus, trained with in-house data. The IAM dataset contains 79 different characters. One additional character is required for the CTC operation (CTC blank label), so that there are 80 entries for every of the 32 time-steps.

RNN converts the self-regulating activations into dependent activations by providing the same weights and biases to all the layers, thus reducing the complexity of increasing parameters and memorising each previous output by presenting each output as an input to the next hidden layer. Hence these three layers can be joined together as a single recurrent layer in such a way that the weights and biases of all the hidden layers are the same.

The formula for calculating the current state; ht = f (ht-1, Xt)

The formula for applying the activation function (tanh); ht=tanh (Whhht-1 + Wxh,xt)

Formula for calculating the output; Yt = Whyht

5.2.3. Connectionist Temporal Classification (CTC)

While training the neural network, the input to the CTC is the output matrix from RNN and the ground truth text (a copy of the original for reference) and CTC calculates the loss value. While in prediction mode, CTC consumes the RNN output matrix and decodes it into the final text. The loss is calculated by using both the ground truth text and the matrix within the operation. The ground truth text is encoded as a sparse tensor[1]. The length of the input sequences must be given to the CTC operation. In general, the sequence length is 32 characters long.

[1] A sparse tensor is a dataset in which most of the entries are zero, one such example would be a large diagonal matrix.

5.3. Data

5.3.1. Input

The input format is a greyscale image of 128×32. Usually, the pictures from datasets do not conform exactly to these dimensions, therefore we resize them without distortion until either they have a width of 128 or a height of 32. Then we place the image in a white 128×32 template. This process is shown in Fig. 3. Finally, we normalise the grey-values of the image so that it simplifies the task for the NN. Data augmentation can easily be integrated by copying the image to random positions rather than aligning it to the left or by randomly resizing the image.

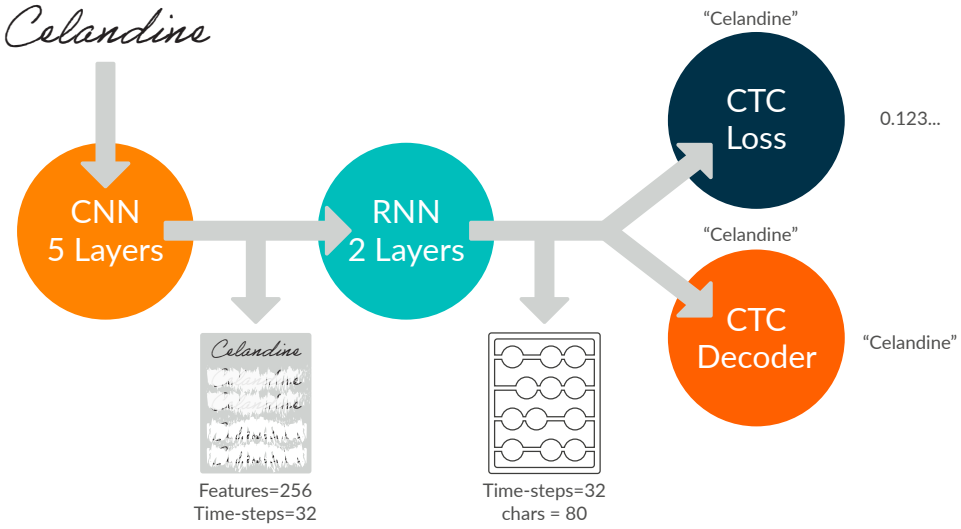


Figure 3: Model Operations

5.3.2. Convolutional Neural Network output

Figure. 4 shows the output of the CNN layers, typically a sequence of length 32. Each layer entry contains 256 features. There are some features which might regularly be confused with particular characters; for example "e", which might be confused with a "c", or with duplicate characters, for example a double "l" might be confused with an "il", and cursive handwriting that is written in a looped style.

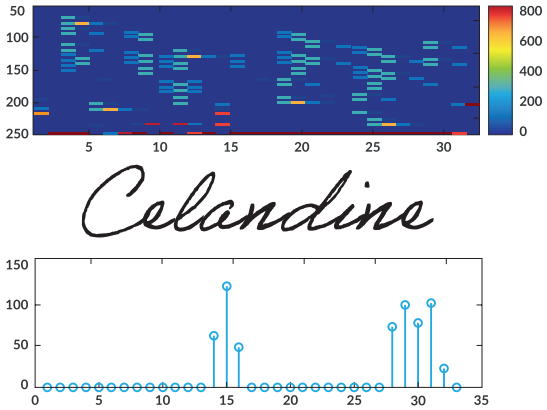


Figure 4: CNN Output
(X-axis indicates the sequence length and Y-axis indicates the number of features)

5.3.3. RNN output

Fig. 5 shows a visualisation of the RNN output matrix for a picture containing the text “Celandine”. The matrix shown within the top-most graph consists of the scores for the characters included in the Connectionist Temporal Classification blank label as its last entry. The matrix-entries, from top to bottom, correspond to the subsequent characters: From the bottom-most graph showing the scores for the characters “C”, “e, “l”, “a”, ”n”, ”d” and also the CTC blank label, the text can easily be decoded: we just take the foremost probable character from each time-step. This forms the so-called best path. Then we discard repeated characters and, lastly, all the blanks: “C---ee--l-a--n—d-i--n...-e” → “C---e---l--a—n—d--n...-e” → “Celandine”.

The CTC operation is segmentation-free and not reliant on absolute positions. The text can easily be decoded from the lower graph that shows the scores for the characters “C”, “e”, “l”, “a”, n”, “d”, “i” and also the CTC blank label: we just take the character with the highest probability from each time-step, this forms the so-called ‘best path’, then we throw away repeated characters and finally all the blanks: “C---ee--l-a--n—d—i--n...-e” → “C---e--l-a--n—d—i--n...-e” → “Celandine”.

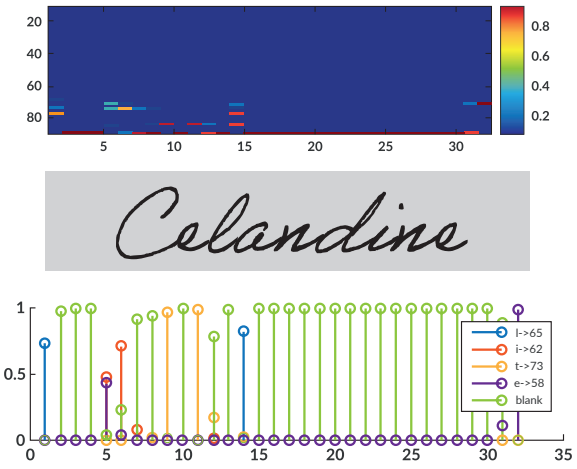


Figure 5: RNN Output
(Top: output matrix of the RNN layers. Middle: input image.
Bottom: Probabilities for the characters “C”, “e, “l”, “a”, ”n”, ”d” and the CTC blank label.)

5.4. Implementation using a deep learning framework

The implementation consists of 4 modules, and they can be implemented using any of the deep learning frameworks such as pytorch, TensorFlow, etc...

1. Image Enhancement

This process prepares pictures from in-house built, corpus datasets for the NN.

2. Data Loading

This process reads samples, arranges them in batches and passes them through the algorithm multiple times capturing them in a variety of ways.

3. Model Creation

This process generates the model as described above, loads, and saves models, manages the sessions, and provides an interface for training and inference.

4. Model Training

This process puts all previously mentioned modules together because the other processes are concerned with basic file IO (Data Loading) and image processing (Image Enhancement).

5.4.1. CNN

For each CNN layer, we created a kernel of size k×k to be used in the convolution operation. Then, the ReLU operation is added again to the pooling layer with size px x py and step-size sx x sy with the results of the convolution. These steps are repeated for all layers during an iteration.

5.4.2. RNN

We experimented and stacked the two RNN layers consisting of 256 units each. Then we produced a bidirectional RNN from it, such that input sequence is passed through from front to back and then back to front. As a result, we generate two output sequences forward and backward of size 32×256, which we later concatenated along the feature-axis to create a sequence of size 32×512. Finally, it is mapped to the output sequence (or matrix) of size 32×80 which is fed into the CTC layer.

5.4.3. CTC

For loss calculation, we fed both the ground truth text and therefore the matrix to the operation. The ground truth text is encoded as a sparse tensor. The length of the input sequences must be given to both CTC operations. We now have all the input files to perform the loss operation and therefore decoding operations are in place.

5.5. Training

The first step in training a neural network is to define the loss function . A loss function is a function that measures the error between a single prediction and the corresponding actual value. Loss functions are used to determine the error between the output of algorithms and the given target value. In general terms, the loss function expresses how far off the mark the computed output is. As handwritten extraction is a classification problem, the loss function is derived from the following standard formula:

$$y_n = y(x_n, w)$$

$$p(t_n|x_n, w) = y_n^{t_n} + (1 - y_n)^{1-t_n}$$

$$E(w) = -\sum_n t_n \cdot ln(y_n) + (1 - t_n) \cdot ln(1 - y_n)$$

Model training involves searching for the local minimum (minimum value of the function) of the cost functions in a parameter space (domain of the cost function). The loss function calculates the error per observation, whilst the cost function calculates the error over the whole dataset. This is done by starting with an initial random parameter and iteratively moving the parameter space downhill the cost function (following the graph to its lowest point).

5.5.1. Number of epochs selected

An epoch refers to one cycle through the full training dataset. It means training the neural network with all the training data for one cycle. In an epoch, we use all the data exactly once. An epoch is often confused with an iteration. Iterations are the number of batches or steps through partitioned packets of training data, needed to complete one epoch. The number of epochs required is determined by the validation results and rate of training errors. The training is continued until the error rate drops below a certain point while if the validation error starts increasing that is taken as an indication of overfitting. We set the number of epochs as high as possible and terminate training based on the error rates.

5.5.2. Number of Layers

We cannot calculate analytically the number of layers or the number of nodes to use per layer in an artificial neural network to address a specific real-world predictive modelling problem. Therefore, the number of layers and the number of nodes in each layer are model hyperparameters must be specified and learnt. We conducted multiple experiments using a robust test harness in our controlled environments using different number of layers, such as 3, 7 and 9, and we settled on 5 based on prediction accuracy.

5.6. Improving the model

To improve the recognition accuracy, we have trialled and implemented the following:

5.6.1. Data augmentation

Increased dataset size by applying further random transformations to the input images.

5.6.2. Generative Adversarial Network

We have experimented with a generative adversarial network architecture that generates realistic handwritten strokes. The architecture of Generative Adversarial Networks consists of two modules:

- A generative network G(.) that takes a random input z with density p_z and returns an output x_g = G(z) that should follow (after training) the targeted probability distribution.
- A discriminative network D(.) that takes an input x that can be a “true” one (x_t, whose density is denoted p_t) or a “generated” one (x_g, whose density p_g is the density induced by the density p_z going through G) and that returns the probability D(x) of x to be a “true” data.

We have designed discriminator-based feature extraction, followed by an auxiliary, feed-forward neural network to distinguish between genuine and forged handwriting. Discriminator is essentially a binary classifier to predict whether handwritten text in digital ink is forged or not. Feature extraction is important for successful classification. Since the target handwritten data is sequential, handling the task through a recurrent network seems a natural choice. However, digital ink representations of the same handwritten text (number of stroke points)may vary dramatically and can further cause significant classification discrepancies by recurrent network. To solve this issue, we used a technique that involves rendering stroke points as binary images. To stabilise the rendering technique and retain the sequential properties of handwritten data, we uniformly sample the stroke points and further utilise Path Signature Features (PSF) that have been shown to be effective in online handwritten text recognition. The input from this feature extraction is encoded into handwritten strokes using the path signature feature.

We have used Handwritten Generative Adversarial Networks (HWGANs) to obtain more realistic English handwritten text. In general, HWGANs generate handwritten text that distributes spatially ,more uniformly and samples synthesised by HWGANs are neater and display more diverse styles compared to other generators.

5.6.3. Increased weightage for cursive writing style in the input images

5.6.4 Increase input size - if an input of NN is large enough, complete text-lines can be used

5.7. Auto Correction

The checking of spelling is almost always a basic requirement for any text processing or analysis. ML-based auto correction is included as a feature. It seeks out words that are incorrectly digitised and suggests possible corrections. The model is constantly upgraded based on user feedback so that the algorithm is always learning and improving.

6. The Architecture of the Proposed Model

The proposed network has different layers, as shown in the figure below:

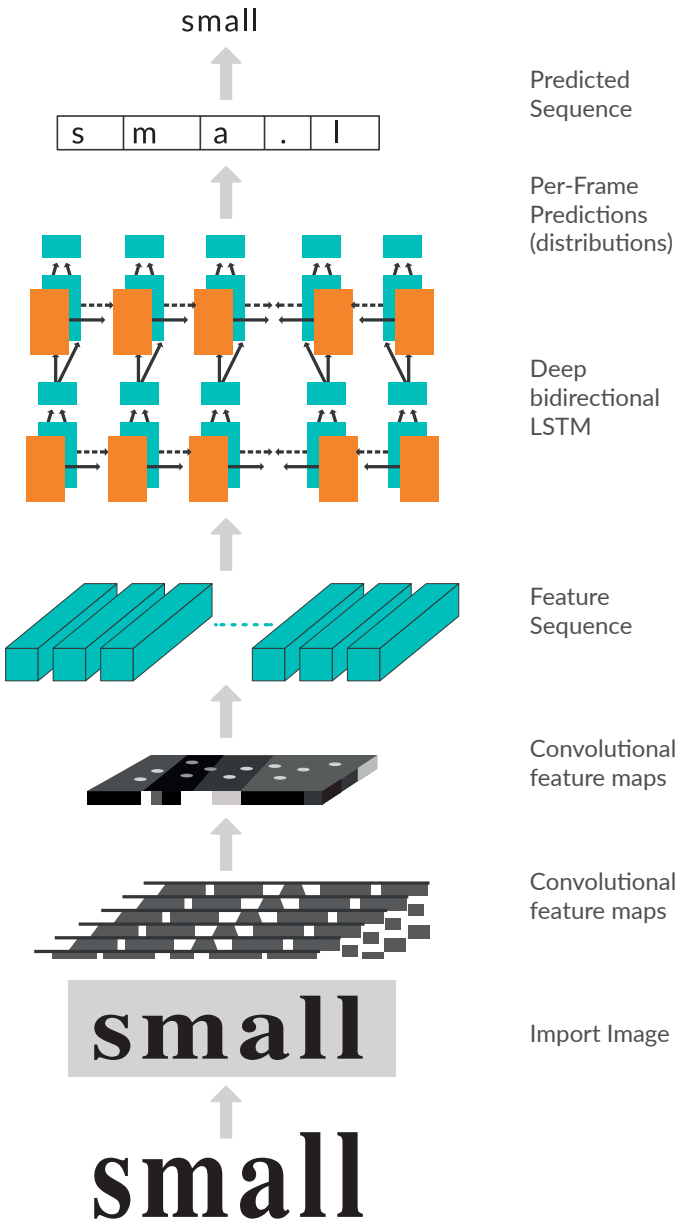


Figure 6: Model Architecture

6.1.1. CNN layers

The CNN layer is meant to imitate human visual processing, and has structures that have been highly optimised to process 2D images. Furthermore, it can effectively learn to extract and abstract 2D features. In detail, the max-pooling layer of CNN is extremely effective at absorbing shape variations. Moreover, a sparse reference to tied weights means CNN involves fewer parameters than a totally connected network with similar size.

Most significantly, CNN is trainable with a gradient-based learning algorithm and suffers loss from the diminishing-gradient problem. Provided the gradient-based algorithm trains the entire network, CNN can produce highly optimised weight and a good generalisation performance.

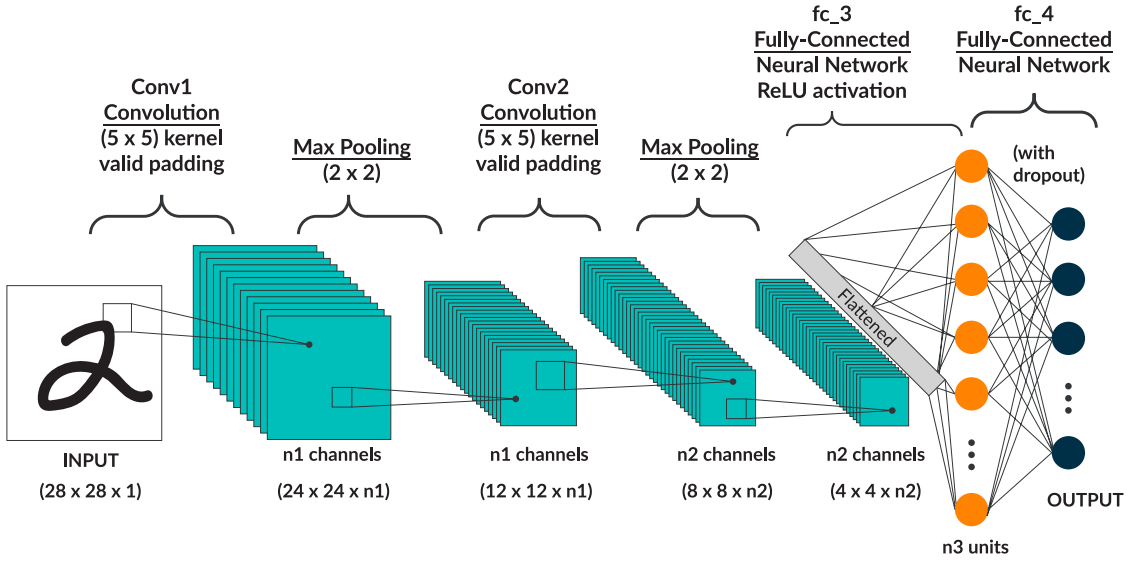


Figure 7: A diagrammatic representation of a CNN Layer

6.1.2. RNN layer

Recurrent Neural Networks have a 'memory' which 'remembers' all the information about what has been calculated. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce an output. This reduces the complexity of parameters, unlike other neural networks.

RNNs convert the independent activations into dependent activations by providing the same weights and biases to all the layers, thus reducing the complexity of increasing parameters and memorising each previous output by giving each output as input to the next hidden layer. Hence these three layers can be joined together such that the weights and biases of all the hidden layers are the same, into a single recurrent layer.

The formula for calculating current state:

ht = f (ht-1, Xt)

The formula for applying Activation function(tanh):

ht=tanh (Whhht-1 + Wxh,xt)

Formula for calculating output:

Yt = Whyht

Figure 8: RNN Operations

6.1.3. CTC layer

The CTC algorithm is alignment-free; it does not require an alignment between the input and the output. There's no 'must-align' data because CTC may assign a probability for any label. The loss is calculated by summing up the probabilities of all possible alignments between the input and the label. The probability for one alignment is calculated by multiplying the corresponding character scores. The algorithm assigns an output character to every input step and repeats for every step which ends within the input.

6.1.4. Blank token

The CTC alignment approach has two problems.

- It doesn't make sense to force every input step to align to some output.
- We have no way to produce outputs with multiple characters in a row. Consider the alignment [l, l, i, t, t, l, l,e]. Collapsing repeats will produce "litle" instead of "little".

To clear these problems, CTC introduces a new token to the set of allowed outputs, and it is called a 'blank token'. This token does not correspond to anything and is simply removed from the output.

In summary, the CTC algorithm applies the following steps:

- The CTC network assigns the character with the simplest probability to every input sequence.
- Repeats not having a blank token in between get merged.
- Finally, the blank token is removed. The CTC network can then give a probability label to the input.

6.2. Summary of Dataset

The Handwriting Corpus contains samples of handwritten English text that can be used to train and test handwritten text recognition. We have created an in-house body of data for our training and evaluation purposes that conforms to annotation gold standards.

Iteration	Training Data	Train: Test: Validation Split ratio
1	24000	8:1:1
2	40000	8:1:1

The above tables provide information about Training, Testing, and validation sets that is created with in-house data in an IAM data set format. The data set contains numerous images of similar types with a particular dimension and this data set also contains a label file with a text extension. It contains the image followed by the text it represents.

7. The benefits of intelligent recognition and extraction of handwriting

Intelligent extraction provides the following benefits:

7.1. It helps the user to achieve their automation goals by providing accurate results and by eliminating time-consuming and costly human intervention.

7.2. It is less labour-intensive as it reduces manual data entry operations.

7.3. It frees people from the tedious and routine tasks of data correction and frees them to focus on higher order tasks.

7.4. Over time, data management systems become more comprehensive and effective as the handwritten data is extracted accurately and in an actionable digital form. This drives the speed and agility of business process by reducing and eventually eliminating manual intervention.

7.5. It improves compliance through improved quality and by reducing human access to sensitive data.

8. Summary

Intelligent handwritten recognition and extraction makes use of the power of AI-based solutions that improve the efficiency of data management thereby making possible digital transformation to realise automation goals.

9. References

1. Scheidl - Word Beam Search: A Connectionist Temporal Classification Decoding Algorithm

2. Frinken, V., Bunke, H.: Continuous handwritten script recognition, in Doermann, D., Tombre, K. (eds.): Handbook of Document Image Processing and Recognition, Springer Verlag, 2014