Offline Fixed Handwriting Recognition

Clovis Rigout
260 587 407

COMP 652
Machine Learning
Winter 2017

Abstract

In this paper, we propose an original algorithm which aims to recognize words written in a fixed handwriting style offline. Firstly, we trained a generic convolutional neural network (CNN) using the Stanford OCR dataset, achieving poor character recognition accuracy of 23%. We were then able to greatly improve this raw accuracy to up to 74% by introducing bias to our model: we sampled and further trained our CNN using 26 images of our target handwriting style, one for each letter of the alphabet. Lastly, we are able to achieve over 90% character accuracy using a Bayes Network which takes into account the natural probability distributions of letters in words inherited from the English Language Model.

1 Introduction

Since the early 1900's, handwriting recognition has been a particularly appealing task, due to its obvious applications. In particular, Handwritten English Character Recognition (HECR) has seen plenty of fruitful research. Still, most of this research has been carried out using online information, in which images are processed as they are being created and which thus makes it easier to recover the exact stroke leading to a specific character. As such, Offline Handwriting Recognition (OHR) has remained a challenging task for researchers, due to the fact that handwritings are both unconstrained and topologically diverse. Consequently, it seemed as though focusing on a fixed handwriting style could leverage this variability, at the cost of possibly limited training samples. Because it is undesirable to require thousands of samples from a single writer, we constrained our problem to a realistic amount: the 26 letters of the alphabet.
While character segmentation plays a crucial part of handwriting recognition, we assume in this paper that all words are already pre-segmented.
Our OHR architecture is three-fold. First, we apply some basic pre-processing steps to both the training and testing datasets to normalize our data. We then build a generic OHR model we built, inspired from LeCun's *LeNet-5* CNN, to which we introduce bias tailored to our target handwriting style. Lastly, we use the resulting probabilities from the CNN as inputs to a Bayes Network, whose parameters depend on a pre-defined English Language Model. This graphical model is used to take into consideration the probability distribution of characters in English words, and consequently improve our results.

2 Features

2.1 The OCR Dataset

The Stanford OCR dataset is a subset of lowercase handwritten characters originally collected by Rob Kassel at MIT Spoken Language Systems Group. Each image is 8x16 pixels and has already been normalized and rasterized. In its entirety, the dataset is comprised of about 50,000 8x16 images, of which we used 30,000 for training, 10,000 for validation and 10,000 for testing. Due to the imbalance of classes in the training dataset, we generated random labeled data to equalize all classes as described in section 2.3, resulting in a training dataset of close to 80,000 images.

2.2 The alphabet

As it was previously mentioned, the goal of our algorithm is to recognize words from a fixed handwriting style. As far as training is concerned, this style is solely determined by a hand-written

alphabet of 26 letters. The idea behind setting such a strong constraint is that the model should simply be *biased* towards recognizing this specific handwriting. From an application point of view, it is reasonable to hope for a generic model which can, with a very limited amount of data, be quickly updated to recognize a fixed handwriting style.

2.3 The handwritten testing dataset

The handwritten dataset consists of 50 handwritten words which were written by a single writer. This dataset is used to evaluate the true success of the our model. It is important to note that even though well-formed words are irrelevant during the training of our CNN, they play a crucial part when considering the Bayes Network described in section 4.

2.4 Pre-processing

In similar literature, image sizes typically vary from 24*24 to 32*32 pixels. We decided to use 28*28 pixel images as it provides sufficient detail and computationally light (parameter can easily be fine-tuned with cross-validation). Consequently, the first pre-processing step consisted in resizing all images from the Stanford Dataset to the appropriate size. We then removed shades and noise from these images by converting them to binary images. Finally, we decided to crop each image so that the size of the handwritten character doesn't influence the CNN. While this step gets rid of the information brought by the height of a characters, it is an easy way to further normalize the inputs of our CNN (once again, the improvements of this step can be measured with cross-validation).

2.5 Generating Random Data

The need to generate random data was crucial. Firstly, it was necessary to balance the character classes of the training dataset. More importantly though, generating random data from the 26-letter alphabet was necessary to bias our CNN. Given an input image, we apply a random rotation (from -25^o to +25^o), followed by a random stretch along the x and y axis. Note that it is unnecessary to consider shifts because all input images are cropped before they are passed as input to the CNN. We obtained the best results by generating 3000 samples from each letter of my alphabet. Put into perspective, this means that the CNN is actually trained with as many inputs from the Stanford Dataset as inputs generated from the alphabet.

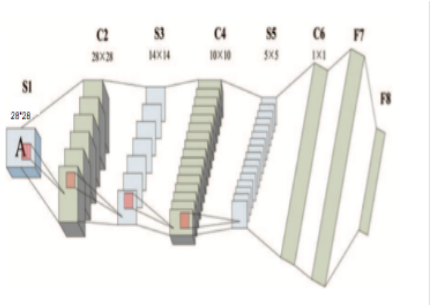

Figure 1: Convolutional Neural Network

3 The Convolutional Neural Network

3.1 LeNet-5 Architecture

The LeNet-5 model was introduced by LeCun et al. in 1998, as has remained a standard convolutional neural network for the task of handwritten and machine-printed character recognition. We use a very similar architecture for the model, shown in Figure 1, which we implement using TensorFlow. The input to our CNN is a 28x28 pixel image, to which we apply a sequence of convolution, pooling and dense layers.

3.1.1 Convolution Layers

A convolution layer (C2 or C4) is used to extract features from an input image. In C2, we apply six convolutions filters of size 5x5 pixels to the input image, assuming no padding so that the output feature maps are also 28x28 pixels. We thus obtain six feature maps, which are fed to our first pooling layer (S3). In C3, we apply 16 convolution filters (of 5x5 pixels) to each of the 6 feature maps of size 14*14 (after pooling). This time, we use a normal padding so that the resulting feature maps are 10x10 pixels.

3.1.2 Pooling

Pooling layers are used to reduce the dimensionality of the feature maps, and thus the processing time of the algorithm. All of the pooling layers (S3 and S5) in our CNN are simple 2x2 max pooling layers (2x2 kernel and stride of 2), and hence reduce a 2Nx2N image to an NxN image by iterating over the image with a 2x2 kernel which which retains the maximum value.

3.1.3 Dense Layers and Classification

In a dense layer, every neuron is connected to every other neuron in the previous layer. While the previous layers deal with feature extraction, these layers are used to learn the weights needed to perform classification. In the model, we have two dense layers (C6 and F1) of 100 and 120 neurons respectively. The weights on these neurons are learned using back-propagation during training.
The last dense layer (F1) is connected to our output layer (F8) by a softmax function which maps our dense layer to a 26-dimensional space, with the probabilities associated to each classification class.

3.1.4 Training

We train our CNN in by iterating 20,000 times over batches of 50 images. We use then Adam Optimizer (rather than a simple Gradient Descent Optimizer which consistently performed more poorly) and a learning rate of 0.0001. We also use a dropout rate of 50% to help with overfitting (all parameters were chosen after *naively* cross-validating them... a more thorough analysis can be done).

4 The Bayes Network

The CNN processes each character independently, which can be considered somewhat counter-intuitive considering that well-ordered sequences of characters make up words. This is the issue that the Bayes Network aims to solve. The English language naturally establishes a word-based language model, defined by the frequency of characters in words. Consequently, it seemed quite beneficial to take into account this language model for the task of handwritten word generation. The Bayes Network shown in Figure 2. thus takes this into account when predicting a word.

4.1 The parameters

We denote the value of the 26 classes we wish to predict by $s_1, ..., s_{26}$.
Given a handwritten word of length k $w = w_0 w_1 ... w_k$, the CNN outputs a sequence of k probability distributions that we denote $NN_{w_0}, ..., NN_{W_k}$, where NN_{w_i} is a 26-dimensional vector such that $NN_{w_i}[j] = p(w_i = j)$ outputted by the CNN.
This sequence of probability distributions is therefore fixed by the CNN, and is thus non-random.

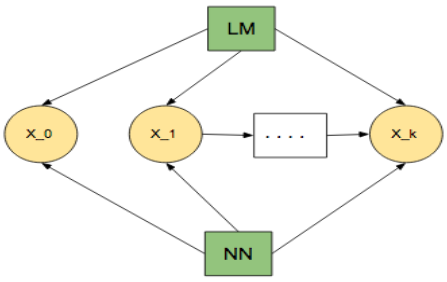

Figure 2: Bayes Network

Let $X_0, ..., Xk$ be random variables denoting the states outputted by our Bayes Network, so that a prediction is of the form $x_0, ..., x_k$.
We wish to find the most likely sequence $x_0, ..., x_k$ that explains the sequence $[NN_{w_i}]$ and a pre-computed language model described below.

4.2 The English model

4.2.1 Initial and State Probabilities

The probability of a letter appearing at the start of a word is quite different from that appearing anywhere inside the word, as it is shown in Table 1. The language model therefore naturally defines two probability distributions:
$P_{initial}(s_i)$, i = 1,...,26
$P(s_i)$, i = 1,...,26
These probabilities were simply taken from existing research.

4.2.2 Transition Probabilities

Certainly, it would be *Naive* to assume that each letter in a word is independent. For this basic model, I however assume that the Markov Property holds: $p(s_i|s_0, ..., s_{i-1}) = p(s_i|s_{i-1})$.
Consequently, this assumption suggests a 26x26 transition matrix T, used in our Bayes Network, such that:
$T[i][j] = P(s_{t+1} = j|s_t = i), i, j = 1, ..., 26$ and $t \geq 0$

4.2.3 Ending Probabilities

The language model used in the Bayes Network is particularly simple, and can appear to be somewhat unfinished. In fact, a simple improvement that could be made is to consider the probabilities of a letter *ending* a word. For example, while the letter "a" is a commonly occurring letter in English words, it is very rare for them to end with the letter "a". While I did not implement this

in my algorithm as it can be treated just like the case of initial probabilities, it is worth noting.

4.3 Predictions

Given a sequence of probabilities of length k, we therefore make a prediction by maximizing the likelihood of observing a word, over all words of length k in a certain dictionary, as shown in equation (3). By doing so, we assume that all words we wish to recognize are valid English words (or at least, part of a dictionary we can use as reference).

$$P(X_0 = s) = P_{LM.initial}(s) * NN_0(s) \quad (1)$$

$$P(X_i = s|X_{i-1}) = P_{LM}(s) * NN_i(s) * P(X_i = s|X_{i-1} = s') \quad (2)$$

$$\max_{w \in D} l(w|LM, NN) = \max_{w=w_0 \cdots w_k \in D} P(X_0 = w_0, ..., X_k = w_k|LM, NN) \quad (3)$$

It is important to note that this method does not permit us to predict words with spelling mistakes. There are also obviously other ways of predicting (such as predicting using the language model by solely using the CNN outputs for which we are certain), which could potentially also be explored and present other advantages depending on the task desired.

5 Results

The following table presents the results we obtained from four different models.

Accuracies of various models					
Model	Training	Testing	Raw Handwr.	Word	Handwriting
Simple CNN no bias	86%	85%	23%	11%	35%
CNN with Bias	85%	84%	74%	71%	89%
CNN and BN with bias	85%	84%	74%	78%	91%
CNN and BN with bias & dict.	84%	74%	80%	98%	99.6%

The training and testing accuracies correspond to the generic dataset we trained our model with. The Raw Handwriting metric is the accuracy obtained on our handwriting dataset without considering our bayes network, by taking the max probability of the output vector of our CNN. The word accuracy corresponds to the percentage of correctly predicted words in our handwritten dataset. Finally, the handwriting accuracy is the percentage of correctly predicted characters as output of our Bayes Network.

The first *Simple CNN, no bias* model corresponds to a CNN trained without exposing it to the alphabet of our target handwriting style. One can quickly notice that this model performs quite poorly (23%) when predicting handwritten characters from our target handwriting style, and as a result does a poor job at word prediction (11%).

On the other hand, all our models trained with the target handwriting alphabet give very accurate predictions, with an 80% raw handwriting recognition accuracy. The third and last models use the Bayes network structure previously discussed, and slightly improve the accuracy of our final handwriting and word predictions.

In the third model, we maximize over all possible words in the English language, whereas in the last model we maximize over the possible choice of words in our handwritten dataset.

6 Conclusion

In this paper we presented an convolutional neural network architecture which, when combined with a simple Bayes Network, is able to achieve near perfect accuracies. The use of a really small sample dataset of a target handwriting has proven to be extremely beneficial to make comfortable predictions on this style. It is important to note that our architecture also leaves plenty of room for improvements, particularly as it relates to our Bayes Network. For example, we found that maximizing the likelihood over all words in the English language can lead to very obscure predictions, which could easily be leveraged by assigning probabilities to entire words as well. Nevertheless, the problem of Offline Handwriting Recognition is still far from being solved. We deal with pre-segmented data and predict over handwritten paint data, which is naturally free of noise, and thus makes our task easier. We are also only testing over a very small dataset of 50 words, and thus we could expect the accuracy of our model to vary greatly on different testing datasets.

References

Aiquan Yuan, Gang Bai, Lijing Jiao, Yajie Liu. 2012. *Offline Handwritten English Character Recognition based on Convolutional Neural Network.* 2012 10th IAPR International Workshop on Document Analysis Systems

Y. Le Cun et al. 1990. *Handwritten Digit Recognition with a Back-Propagation Network.* Advances in Neural Information Processing Systems.

Elie Krevat and Elliot Cuzzillo. 2010. *Improving Offline Handwritten Character Recognition with Hidden Markov Models.* IEEE Transactions on Pattern Analysis and Machine Intelligence.