CS230

Optical Character Recognition via Deep Learning

Connor Meany
Stanford University
cmeany@stanford.edu

Matias Arola
Stanford University
matiasa@stanford.edu

Abstract

Given the ease of handwriting and large quantity of existing handwritten text, converting handwriting to computerized text is a problem of great importance with many applications. We created both a character level and word level neural network to recognize handwriting. Our data came from the EMNIST dataset (characters) [17] and the IAM dataset (words) [15]. The character-level model utilizes a ResNet-50 structure and achieved 88% accuracy. The word-level model uses a 3-layer CNN which feeds into an LSTM layer; this achieved 77% accuracy at a character level. The main problems that we encountered were character segmentation and normalizing word length.

1 Introduction

1.1 Motivation

As the world becomes more and more digitalized, the incompatibility of handwritten text with computers becomes a greater problem. A great quantity of data is only saved in formats inscrutable to digital processing; this makes it difficult to access and also means it can't be easily searched, stored, shared, and analyzed. Handwriting is also commonly used in applications such as taking notes and filling in forms due to the fact it is often easier than dealing with technology. There is a growing divide between the increasing usefulness of digitalization and the plethora of undigitalized text. An accurate algorithm for converting from handwriting to computerized text would help make a whole new set of data accessible and have applications from analyzing historical texts to improving note-taking.

1.2 Outline of Model

Because text is made up of words, and words are made up of letters, the core of handwriting recognition is identifying handwritten characters. We initially assumed that after creating a model that could identify characters, we could simply apply it to every character on a page of text and process the entire page. Thus, the first model we created aimed to convert images of characters to their respective encoding. The input to our first algorithm was 28 by 28 pixel grayscale images of a single character. We then applied a ResNet to obtain a predicted character via a softmax vector. The labels are one-hot vectors which encode the character; i.e. the first entry of the vector corresponds to "a," the second to "b," and so on (though the actual vectors start with numeric characters).
After implementing the character recognition network, we turned to the process of identifying whole words. We tried using open source character segmentation software to separate characters from within the word and run them through our model, but we achieved very low accuracy because letters were often touching each other and thus they weren't able to be segmented. We decided to switch to a word-based neural network. This model takes grayscale images of handwritten words as input.

It runs them through a CNN and then an RNN with LSTM blocks to obtain a sequence of one-hot vectors similar to the character vectors.

2 Related work

2.1 Character Recognition

Handwriting recognition is a relatively well-studied field, so there are many other papers written on the subject. Identifying characters is a subject that was addressed even before the era of deep learning, but these algorithms had low accuracy or only worked on limited datasets as noted by Line Eikvil [8]. The first deep learning algorithms for optical character recognition (OCR) emerged when image classification techniques were refined. An incredibly popular machine learning task is classifying the MNIST dataset, which is similar to our dataset but only consists of numeric characters 0-9. *Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis* by Simard, Steinkraus, and Platt was a very helpful paper as we started trying to apply convolutional neural networks (CNNs) for image analysis on a dataset like this [19]. It includes tips for data processing and structuring CNNs. Another paper by Bottou et al. used a variety of CNN structures on the MNIST dataset and compared the performance of these different structures, finding that their "Boosted LeNet 4" was the most successful [4]. This model was a variation on a CNN. It was evaluated on only numeric characters, so it is hard to compare to our model, but it achieved an impressive 99.3% test accuracy. The drawback of their model was that it took 5 weeks to train, while our character model only took less than a day. The introduction of residual networks (ResNets) was a breakthrough for image classification, and so this method is very applicable to character recognition [12]. ResNets allow for much deeper CNNs, which is why we used a ResNet for our final character recognition model. An example of character recognition using ResNets was in a Stanford CS231N paper by Balci et al. [2]. Their approach was similar to ours, but we obtained higher accuracy—the difference could be because our network is deeper and we used dropout while training it.

2.2 Word Recognition

Our final model took entire word images as input, rather than just characters. One paper which did this was by Pham et al., which used a 2-layer CNN which fed into a bidirectional recurrent neural network (RNN) with LSTM cells [18]. This overarching structure is almost identical to ours, but they use a different number of layers, different activation functions, different hyperparameters, and a different dataset. Their approach is very successful at word-level recognition, which is why we implemented a model so similar to theirs. One of the current best models in the world at word recognition utilizes a similar structure, but with a multidimensional RNN structure, created by Graves and Schmidhuber [10]. This extends the idea of a bidirectional RNN to two dimensions, corresponding to the dimensions of the image. This model is very effective, and won an international handwriting recognition competition. We didn't attempt this technique due to time constraints, but it would be something we would try if we had more time.

3 Dataset and Features

3.1 Character recognition

For character level recognition, we used the EMNIST Balanced dataset [7]. The dataset is based on the NIST Special Database 19 [17], which contains handprinted sample forms from 3600 writers and 810000 character images isolated from their forms. Alternative smaller datasets have been derived from NIST for character classification purposes. The most widely known and used is the MNIST database of handwritten digits, which has been extended with letters to form the EMNIST dataset.
The EMNIST Balanced dataset is based on images from the NIST database. In this set, upper and lower case classes of certain letters are merged to prevent errors resulting merely from misclassification between upper and lower case letters. This reduces the number of classes from 62 to 47, with 10 classes for the different digits and 37 for letters. Examples of images from this dataset can be found in Figure 1(a).
The dataset contains a balanced subset of all these classes, divided into a training set of 112,800 images and a test set of 18,800 images. They have been preprocessed into high contrast black and

Table 1: Distribution of words under 6 characters

Word length	1	2	3	4	5	6	Total
Number of words	2,718	14,481	17,974	13,544	9,308	6,851	64,876

Table 2: Processed datasets

Dataset	Train	Test	Dev	Total
4-6 characters	26,732	1,486	1,485	29,703
1-6 characters	58,388	3,244	3,244	64,876

white images with the size 28×28 pixels. The size of the dataset was large enough for it to be used on the task without adding any additional samples using data augmentation, but to reduce bias further this step of preprocessing could be introduced.
For importing the images and the labels into our model we used python-mnist, a simple data parser for MNIST and EMNIST [14]. It allowed us to import the images and labels directly into NumPy arrays [21] of size (m, n_x) and $(m, 1)$, respectively, where m corresponds to the number of examples in the dataset and $n_x = 28 * 28$ to the number of pixels in an image. We then reshape the images into $(m, 28, 28, 1)$ for the CNN model and convert the list of label numbers into an array of one-hot vectors with a shape (m, n_y) where $n_y = 47$ is the number of classes. The last step of preprocessing was to divide the pixel values by 255 to have a distribution of values between 0 and 1.

3.2 Word recognition

For training a model to recognize complete words we used the IAM Handwriting Database [15]. The database contains handwriting examples from 657 writers, segmented into sentences, text lines and words. For our purposes we used the set of 115,320 segmented words as individual black and white PNG files of varying shapes and a text file containing the label information. Examples of sample images can be found in Figure 1(b).
Although the segmentation of words was generally good, 18,864 examples had been labeled as possibly having bad segmentation. Furthermore, there are many punctuation marks and other symbols labeled as their own words. We removed all these examples from our dataset to have a balanced dataset of only words consisting of letters. The size of this new, cleaned dataset was 82,250 words.
Since the word images have varying shapes, we had to reshape them to have the same dimensions in order to use them as input for our model. We added white padding to all the images until they were square-shaped, and then scaled them down (or up) to 64×64 pixels using the Python Imaging Library [6]. Later, when importing them to our model, we converted the pictures into a NumPy array of size $(m, 64, 64, 1)$. The pixel values were also divided by 255 to achieve input values between 0 and 1.
We extracted the labels from the text file and padded shorter words with artificial "end of word" characters to have a constant length. This allowed us to split the words into characters, which we finally represented as one-hot vectors. In the end, the shape of our label vector was $(n_c w, m, n_c c)$, where $n_c w$ and $n_c c$ correspond to the length of the longest word and the number of possible characters, respectively. To increase performance we decided to limit our dataset to shorter words, initially from 4 to 6, later 1-6 characters long. As our dataset sizes were between 10,000 and 100,000 samples, we used a 90-5-5 distribution between training, test and dev sets. The total sizes of these sets are in Table 2.

4 Methods

4.1 Models

We experimented with two different types of models for recognizing individual characters. For our baseline model we trained a CNN-model: 3 CNN-layers with ReLu-activation and max pooling, and a fully connected layer accompanied by a softmax activation. This general type of a network is

Table 3: Accuracies of the models

	1 char, CNN	1 c, ResNet	4-6 c, 1-layer CNN	1-6 c, 1-l. CNN	1-6 c, 3-l. CNN
Train	89.9%	90.6%	81.9%	88.5%	83.7%
Test	84.8%	88.5%	47.5%	71.6%	76.9%

widely used as a baseline for image classification, which also includes character recognition. The output of the softmax function corresponds to a probability distribution of all the different possible labels.
As an improvement to this baseline model we found using residual networks to be effective. We modified the ResNet-50 model [11] for our own use, changing the last layer to a fully connected layer with softmax activation. Thus the output of this model is similar to that of the CNN-model.

We also developed a separate model to classify word images. The word model feeds the input image into a CNN-model, which is used to encode the image before it is passed into an RNN-model. We experimented with several models for the CNN, with the two main versions being a 1-layer CNN and a 3-layer CNN. The deeper model was more accurate, but also needed more regularization to avoid overfitting. After extensive hyperparameter tuning we achieved the best results using filter sizes of 3×3 and layer sizes increasing from 16 and 32 to 64.
The CNN-layers consist of a convolutional layer, a batch normalization layer [13], a ReLu activation and a max pooling layer with 3×3 filters. Additionally, we used Dropout after each layer to prevent overfitting [20]. The CNN-layers are extracting features from the images, providing the next layers with information for detecting the characters.
The output from the CNN-layers is divided into 16 time steps, which are then fed into a layer of 64 Long Short Term Memory (LSTM) cells [9]. Each character of the word we're predicting has its own LSTM layer, so in total we'll have 6 layers of 64 LSTM cells. As we found experimentally, the model with 16 time steps has a better performance than models with just one time step, since it is able to take a sequence of encoded features from the CNN and use it to predict the character.
Each of the LSTM-layers has its output fed into a fully-connected layer with a softmax activation, resulting in a distribution of the character probabilities. This is then the output that is compared to the corresponding character of the training example.

4.2 Training

The loss function of both our models is categorical cross-entropy, which aims to maximize the softmax output of the correct label for each example: $L = -\sum_i y_i log(\hat{y_i})$. Since instead of having just one output the word model has one output for each character in the word, the total loss is simply the sum of these losses.
For training the models we used the Adam optimizer [1] with the default hyperparameters to update the network weights. With a large dataset and a multi-layer classification model it was important to use this optimizer speeding up the training. We also used minibatches of 32 samples to update parameters without having to feed an entire epoch through the model before seeing results.
For tweaking our word recognition model initially, we used subsets of the training set to improve our model until it achieved satisfactory results on these small sets, before moving on to the entire training set.

5 Experiments/Results/Discussion

We experimented with our baseline and ResNet models, and finally the best character recognition model achieved an accuracy of 89% [Table 3]. However, using this model to recognize entire words proved to be a much greater challenge, as even the best open source character segmentation models [3] ended up having an extremely low accuracy on the training data. This ultimately lead us to use a separate model for the word recognition problem.

Figure 1: Samples and predictions

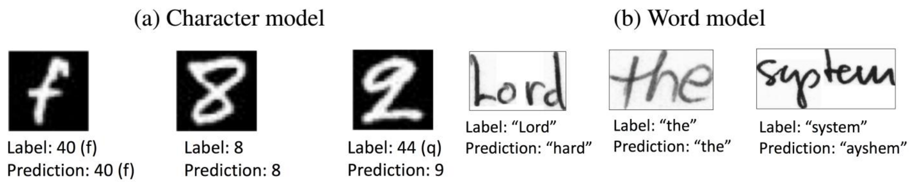
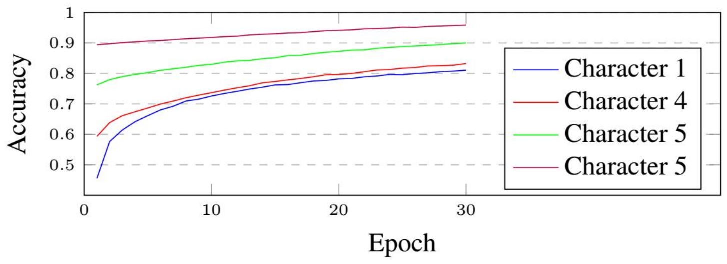


Figure 2: Train accuracy of the word model by character



In our initial experiments we found that using the entire dataset makes the model predict shorter words than expected, since the number of longer words is low compared to the short words. Therefore we started training our dataset only with words from 4 to 6 letters. However, with this dataset the model tended to overfit and the test accuracy remained below 50% [Table 3]. We achieved a better accuracy with a dataset that also included the words under 4 characters, getting 77% of the characters labeled correctly in the test set.
We kept track of the individual accuracies of the characters to gain insight into the way our model learned to predict words. It learned fairly quickly to recognize word lengths, and thus the accuracy of characters 5-6 started as significantly higher than that of the others [Figure 2]. Although training accuracy increased for a long time, the maximum test accuracy was achieved already early in the process, leaving bias as the main issue of the model.
We were able to find a clear trend in the characters the model labeled incorrectly. It learned to recognize the shapes of letters fairly well, but since handwriting varies a lot between different people, it did make mistakes whenever the characters were unclear due to imperfect or cursive handwriting. For example, a capital L with a slightly curved horizontal penstroke is easily confused with a lowercase h [Figure 1(b)]. Also, in contrast to the merged classes of the character dataset, the word model had to classify between upper and lower case characters for each letter in the alphabet. Taking these challenges into account, the final accuracy of the model was on a satisfactory level.

6 Conclusion/Future Work

We set out to address the challenge of recognizing English handwriting. We started by creating a character recognition CNN; our best performing model was a ResNet model which achieved 88% accuracy. We then pivoted to word-level handwriting recognition; we implemented a CNN to RNN model with LSTM cells. Our best model used a 3 layer CNN and achieved 77% character level accuracy. This model worked better than the 1 layer model because the 1 layer model overfit the training set; making the neural net deeper helped counteract this. Our next step if we had more time would be to improve the accuracy of our word-model by trying a multidimensional RNN like the one used by Graves and Schmidhuber [10]. We would also try to implement a post-processing neural network which analyzes the output of the word model and corrects it to the closest valid English word. We also wanted to adjust our model so that it can handle words longer than six letters. Finally, we would also try to use word segmentation software to separate words out from a larger image of text and then run them through our model, allowing us to run entire pages of text. From the core model we've built, there are many directions with which to explore and apply our work if we had more time.

7 Contributions

Both team members contributed equally and were involved with the creation of both models. However, Connor focused more on the character-level ResNet while Matias focused on the word-level CNN-RNN. Since the word-level model was more complicated, Connor also wrote the project milestone. The proposal, poster, and report were written together.

References

[1] Ba, J., Kingma, D. Adam: A Method for Stochastic Optimization. (2014) https://arxiv.org/abs/1412.6980

[2] Balci, Batuhan, Dan Saadati, and Dan Shiferaw. "Handwritten Text Recognition using Deep Learning." CS231n: Convolutional Neural Networks for Visual Recognition, Stanford University, Course Project Report, Spring (2017).

[3] Bansal, D. Segmenting Handwritten Paragraphs into Characters (2016) https://github.com/dishank-b/Character_Segmentation

[4] Bottou, Léon, et al. "Comparison of classifier methods: a case study in handwritten digit recognition." Pattern Recognition, 1994. Vol. 2-Conference B: Computer Vision & Image Processing., Proceedings of the 12th IAPR International. Conference on. Vol. 2. IEEE, 1994.

[5] Chollet, François et al. Keras. (2015) https://github.com/keras-team/keras

[6] Clark, A., et al. Pillow: the friendly PIL fork. http://doi.org/10.5281/zenodo.44297

[7] Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Retrieved from http://arxiv.org/abs/1702.05373

[8] Eikvil, Line. "Optical character recognition." citeseer. ist. psu. edu/142042. html (1993).

[9] Gers, F., Schmidhuber, J. and Cummins, F. 2000. Learning to Forget: Continual Prediction with LSTM. Neural Comput. 12, 10 (October 2000), 2451-2471. http://dx.doi.org/10.1162/089976600300015015

[10] Graves, Alex, and Jürgen Schmidhuber. "Offline handwriting recognition with multidimensional recurrent neural networks." Advances in neural information processing systems. 2009.

[11] He, K., Zhang, X., Ren, S., Sun, J. Deep Residual Learning for Image Recognition. (2015) https://arxiv.org/pdf/1512.03385.pdf

[12] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[13] Ioffe, S., Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. https://arxiv.org/abs/1502.03167

[14] Marko, R. python-mnist 0.5 https://github.com/sorki/python-mnist

[15] Marti, U., Bunke, H. The IAM-database: An English Sentence Database for Off-line Handwriting Recognition. Int. Journal on Document Analysis and Recognition, Volume 5, pages 39 - 46, 2002.

[16] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[17] NIST Special Database 19. NIST Handprinted Forms and Characters Database. http://doi.org/10.18434/T4H01C

[18] Pham, Vu, et al. "Dropout improves recurrent neural networks for handwriting recognition." Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on. IEEE, 2014.

[19] Simard, Patrice Y., David Steinkraus, and John C. Platt. "Best practices for convolutional neural networks applied to visual document analysis." ICDAR. Vol. 3. 2003.

[20] Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958

[21] Travis E, Oliphant. A guide to NumPy, USA: Trelgol Publishing, (2006).