Using Neural Speech Recognition Models For Optical Character Recognition of Cursive Script

Jon Braatz
jfbraatz@stanford.edu

Abstract

While the tasks of automatic speech recognition (ASR) and optical character recognition (OCR) have historically been approached from entirely different angles using brittle heuristics and hand-crafted features specific to the task at hand, in recent years the most successful models for each task have converged on approximately the same neural architectures. However, new models used in production like Baidu's Deep Speech haven't yet made a mark on the OCR literature, despite the fact that the overall structure of models used in both fields are so consistently similar. We give heuristic reasons for why the same type of end-to-end architecture should give state-of-the-art results for both tasks, and we show that Baidu's Deep Speech system can also learn to transcribe written text with reasonable accuracy despite not being designed with that task in mind.

1 Introduction

State-of-the-art Optical Character Recognition (OCR) systems and Automatic Speech Recognition (ASR) systems have recently been converging on the same kind of neural architectures based on convoluational layers and recursive neural networks. Both types of systems seek to find a monotonic mapping from one sequence to another: in OCR, a sequence of vertical image slices from left to right are mapped to a sequence of characters with the same left-to-right alignment, and in ASR a time series of frequency intensities in the form of a spectrogram is mapped to a sequence of phonemes or, in the case of end-to-end systems, a sequence of characters.

Due to the natural variations in pronunciation and intonation between natural language speakers, ASR must be robust to a wide range of contexts that a spoken word might appear in. This is in contrast to most applications of OCR, which are usually optimized for character-based languages in which each character is easily separated from surrounding ones. Cursive scripts on the other hand are heavily context dependent and hence the task of transcribing cursive handwriting or printed cursive text is closer to the difficulty of ASR than OCR of normal character-segmented text. We've chosen Arabic as an intrinsically cursive script to investigate the efficacy of a state of the art ASR system, namely Baidu's DeepSpeech2, when applied to the OCR problem. Arabic was chosen due to the heavy context-dependence of the visual appearance of its characters and the multitude of cursive printed fonts that are available for it.

2 Related Work

Two approaches are currently used for ASR and OCR tasks, both relying on neural architectures that have achieved state-of-the-art results in other fields in recent years. The RNN encoder-decoder paradigm, as used in Google's NMT system for example, uses an encoder RNN to map the variable length input sequence to a fixed length vector, and a decoder network to generate a sequence of output predictions from the fixed length vector output from the encoder. These systems are usually trained

Preprint. Work in progress.

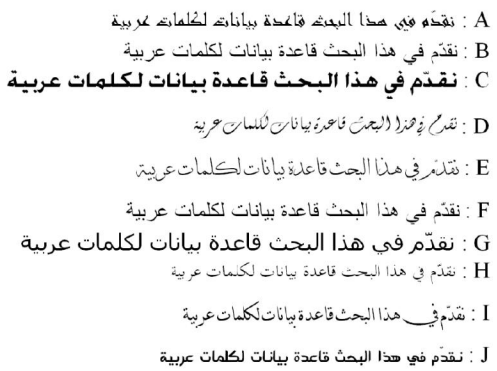

Figure 1: Fonts used to generate the APTI Database: (A) Andalus, (B) Arabic Transparent, (C) AdvertisingBold, (D) Diwani Letter, (E) DecoType Thuluth, (F) Simplified Arabic, (G) Tahoma, (H) Traditional Aatbic, (I) DecoType Naskh, (J) M Unicode Sara

using a cross-entropy loss. Adding an attentional mechanism can improve the performance of such systems, but at the cost of requiring the input sequence to be shorter than a fixed length.

The second technique technique that is most commonly used for mapping variable length input to variable length output is using an RNN to model temporal information and training using the Connectionist Temporal Classification loss. This approach has the advantage of not requiring any kind of frame-wise alignments of the input and not contraining the input to be less than a fixed size. This is the approach that both Google's open-source OCR system Tesseract [1] uses as well as the approach that we emulate from the DeepSpeech 2 architecture. To accelerate training, we also follow DeepSpeech2 in using sequence-wise batch normalization layers [6] in the RNN for each hidden unit.

3 Dataset

For our dataset of Arabic text images, we use 170,000 images of printed Arabic text lines from the Arabic Printed Text Image (APTI) database [5]. This dataset was synthetically generated using a lexicon of 113,284 words, 10 Arabic fonts, 10 font sizes, and 4 fonts styles. Images also feature ligatures and flourishes that are common in Arabic text and have no parallel in Latin-script based languages. This combination of fonts, styles, and sizes gives a representative sample of Arabic text that would be commonly encountered on a computer screen, newspapers, books, and many other documents.

We shuffle and split this dataset into train, dev, and test sets with 90/5/5 ratios respectively.

4 Methods

Our model follows the architecture outlined in the DeepSpeech 2 paper, using scaffolding code based on the SimpleHTR repository [4]. In particular, we use several layers of either 1D or 2D convolutional layers to extract features from the text image in question, a bidirectional RNN with either LSTM or GRU cells, and a final fully connected softmax layer that computes a probability distribution over characters. The model is trained using the CTC loss function.

4.1 Input

Whereas the inputs to DeepSpeech2 are audio clips that are converted to power-normalized spectrograms with unknown alignments between audio segments and the characters being pronounced during that segment, our inputs are grayscale images of images of printed Arabic text from the APTI database. To adapt the network to the format of the data we're using, we reimplement the DeepSpeech2 architecture from scratch to have it work directly on our textline images instead of including the audio preprocessing step.

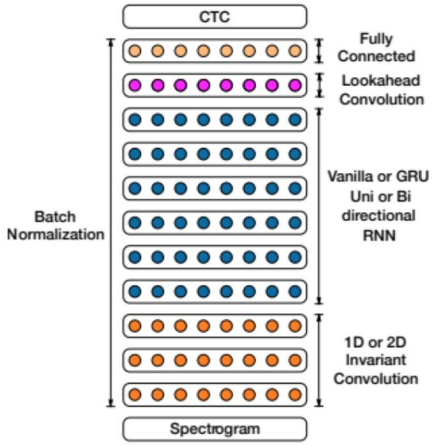

Figure 2: Architecture of the deep RNN used in Deep Speech 2

4.2 CNN

DeepSpeech2 experimented with adding between one and three layers of 1D temporal convolutions that perform the convolution operation only across the time domain, in addition to 2D convolutions that also convolve in the frequency domain, using a "same" padding in all cases. Using 2D convolutions across both time and frequency were shown to improve Word Error Rate by 23.9% on their development set, and we note that 2D convolutions are commonly used as a feature extraction layer of Google's Tesseract OCR system as well.

4.3 RNN

The feature vector outputs from the CNN are used as inputs to a bidirectional RNN, using GRU units in the original DeepSpeech2 network using the following computations:

$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\
\tilde{h}_t &= f(W_h x_t + r \circ U_h h_{t-1} + b_h) \\
h_t &= (1 - z_t) h_{t-1} + z_t \tilde{h}_t
\end{aligned}
$$

where $\sigma(\cdot)$ is the sigmoid function, z and r represent the "update" and "reset" gates respectively, $f(\cdot)$ represents an activation function (tanh in our case) and $W_\square$, $U_\square$, and $b_\square$ represent input-hidden weight matrices, recurrent weight matrices, and bias terms respectively. These GRU units were found to be more performant than either LSTMs, which took longer to train and were more likely to diverge, and vanilla tanh RNN units, which achieved a consistently higher word error rate than the GRU units did.

4.4 Batch Norm

Deep neural networks such as DeepSpeech2 often suffer from optimization issues as additional layers are added. Sequence-wise normalization [37] was used in DeepSpeech2 to mitigate these issues and accererate training using the following transformations:

$$
\begin{aligned}
\mathcal{B}(x) &= \gamma \frac{x - \mathrm{E}[x]}{(\mathrm{Var}(x) + \epsilon)^{1/2}} + \beta \\
\tilde{h}_t &= f(\mathcal{B}(W_h x_t) + r \circ U_h h_{t-1} + b_h)
\end{aligned}
$$

4.5 CTC Loss

The Connectionist Temporal Classification (CTC) loss was proposed by Graves et. al in 2006 for the purposes of labelling unsegmented sequence data with RNNs. It is particularly useful when:

- both input and output sequences can vary in length
- the ratio of the input and output sequence lengths can vary
- we don't have accurate alignment between the input and output sequence.

This loss is useful in ASR contexts when multiple time slices can correspond to a single phoneme or in OCR contexts when multiple vertical image slices can correspond to the same character. The outputs of the network include can include blank outputs and repetitions of the same output character many times, and output sequences are considered equivalent if they differ only in alignment, ignoring blanks. For example, equivalent alignments of the word `HELLO` include:

- `-HEEE-LL-LLOOOOOOO-`
- `HE-L-LOOO--`
- `HHHEEEEE-LLL-LLLO--`

There is an efficient forward-backward algorithm for summing over all possible ways of aligning the output sequence, and we train by maximizing the probability of the true labels after integrating over all possible alignments consistent with the true label.

Concretely, if a is a sequence of characters output by the RNN on input x, we model the time-level labels at each time step to be conditionally independent so that the probability of the output sequence given the input sequence is:

$$\Pr(a|x) = \prod_{t=1}^{T} \Pr(a_t|x).$$

Denoting by $\kappa(y)$ the set of time-level output alignments consistent with the ground truth label y, the probability of the output label y given the input x is given by summing the probabilities of all possible time-level alignments consistent with the output label:

$$\Pr(y|x) = \sum_{a \in \kappa(y)} \Pr(a|x)$$

Our final objective maximizes the probability of the true labels:

$$\mathcal{L}_{CTC} = -\log(\Pr(y^*|x))$$

5 Experiments

5.1 Evaluation Metric

While we train using the CTC loss, our objective is to minimize the average edit distance between our transcriptions and the ground truth labels in the test set. We measure both the character error rate using edit distance and the word accuracy rate as the fraction of entirely correctly transcribed words.

5.1.1 Optimization

While DeepSpeech2 introduced an novel optimization method that they call "SortaGrad", we found that using RMSProp with a learning rate of .01 for the first 10 batches and then .001 for the rest yielded good enough results that we didn't need to explore further optimizations. The motivation for the development of SortaGrad was also due to the far larger dataset that Baidu trained with, and our dataset didn't require it.

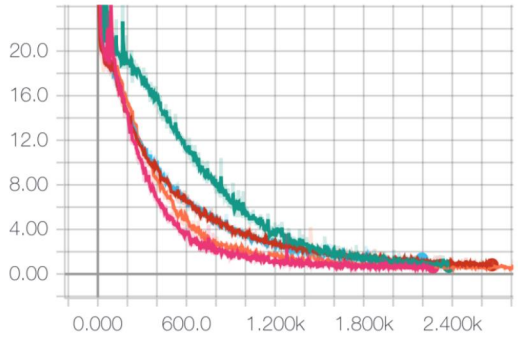

Figure 3: Training losses for each experiment. Colors correspond to successive rows in Table 1 as follows: red-brown, light blue, pink, green, orange.

5.2 Hyperparameters

We ran the following experiments to explore the effect to which the convolutional filter size, BatchNorm, RNN hidden units, and RNN layers had on performance and training time. All except Exp. 5 use 3 5×5 convolutions followed by 2 3×3 convolutions, with strides of 2×2 for the first 2 layers and 2×1 for the rest.

Table 1: Experimental Results

Experiment	Character Error Rate (%)	Word Accuracy (%)
1 Layer RNN, 50 hidden units, w/ BatchNorm	4.50	78.90
4 Layer RNN, 50 hidden units, w/ BatchNorm	5.50	76.08
2 Layer RNN, 200 hidden units, w/ BatchNorm	3.36	82.64
2 Layer RNN, 200 hidden units, no BatchNorm	4.43	79.29
8 Layer RNN, 512 hidden units, w/ BatchNorm, doubled kernel sizes	3.72	80.09

6 Conclusions and Future Work

We ultimately found that our best performaning model contained 2 layers in the RNN, 200 hidden units, kernel sizes of size 5×5 repeated three times and 3×3 repeated twice, and BatchNorm enabled. We noticed that extending the number of layers in the RNN past 2 rarely affected performance or training time, and hidden layer size seemed to matter more than number of layers in the network did. We also noticed that enlaring the receptive fields of inputs to the RNN by doubling kernel sizes to 10×10 and 6×6 did not have much of an effect. We also found that BatchNorm did speed up learning, as the authors in the original paper found as well.

Looking at examples of incorrectly transcribed examples, we noticed that the most common mistakes were omitting diacritics or using the wrong ones. Adding max pooling layers to the CNN could cause these small parts of the images to be emphasized and learned easier.

Ultimately we found that while DeepSpeech 2 had more parameters than were required for our smaller dataset, the architecture was able to recognize printed Arabic script with a character error rate of less than 4%, despite it not being designed with this task in mind.

تقطر <- فطر
يفتنن <- يفتنن
أدوا <- أدوإ
فيخشي <- فيخشى
مستفتلتين <- مستفتلتين
الشآان <- الشأن

Figure 4: Incorrectly transcribed examples, with correct answer on left and model outputs on right

Additional Information

Code for this project is available at https://github.com/jfbraatz/CS-234-Final.

References

[1] Tesseract OCR (https://github.com/tesseract-ocr/tesseract)

[2] Hannun, Awni, Case, Carl, Casper, Jared, Catanzaro, Bryan, Diamos, Greg, Elsen, Erich, Prenger, Ryan, Satheesh, Sanjeev, Sengupta, Shubho, Coates, Adam, and Ng, Andrew Y. Deep speech: Scaling up end-to-end speech recognition. 1412.5567, 2014a. http://arxiv.org/abs/1412.5567.

[3] Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In ICML, pp. 369– 376. ACM, 2006.

[4] SimpleHTR (https://github.com/githubharald/SimpleHTR)

[5] Arabic Printed Text Image (APTI) Database (https://diuf.unifr.ch/main/diva/APTI/)

[6] Amodei, Dario et al. "Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin." ICML (2016).