Converting Image Subtitles to Text using OCR via CRNN

Chris Larson

1. Introduction

This project aims to build an optical character recognition (OCR) engine to convert BluRay movie subtitle tracks from their default image encoding to plain text. We use a Convolutional Recurrent Neural Network (CRNN) based on the paper "An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition" by Shi, Bai, and Yao [1].

2. Background and Motivation

2.1. Subtitling

Subtitles are text representations of audio used in film, TV, and music. BluRay movie discs contain subtitle tracks encoded using the Presentation Graphic Stream (PGS) standard. The PGS standard defines a compression algorithm to describe an image, its placement, and its time-to-display. PGS has the benefit of being able to encode stylized subtitles, as they are compressed images. However, many lower-end Smart TVs and Smart TV Boxes have difficulty decoding PGS and function better with subtitles represented as plain text in the SubRip Subtitle format (SRT).

2.2. Optical Character Recognition

Optical Character Recognition (OCR) is a technique to convert image text to machine-encoded text. Traditional OCR methods use classic Computer Vision methods such as Pattern Recognition and Feature Analysis but commonly encounter errors classifying similarly shaped characters (for example: "1" vs. "l") due to their lack of contextual information. In recent years, OCR techniques with deep learning have gained popularity, as they can contextualize text and can, therefore, minimize common errors found in traditional OCR.

2.3. Convolutional Recurrent Neural Networks

The CRNN algorithm combines a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN). It contains three kinds of layers: convolutional layers, recurrent layers, and a transcription layer. The convolution layers extract features from the input image, the recurrent layers predict labels for each frame, and the transcription layer translates the per-frame predictions into the final label [1]. The algorithm's ability to contextualize text is due to the RNN layers and can help minimize common errors found in traditional OCR.

3. Methods


```
12
00:02:19,849 --> 00:02:22,226
WADE: <i>I was born in 2027.</i>
```

Figure 1: Experimental data samples. *(left)* PGS subtitle, *(right)* SRT subtitle.

.

3.1. Dataset

We chose five movies from our collection to extract subtitles from. The movies spanned five primary film genres to account for different subtitle styling: action, comedy, drama, horror, and science fiction. In addition to extracting the scene-based subtitle images, which often span two lines, we used computer vision bounding-box techniques to save an image of each line and each word in the movie's subtitles. Figure 1 shows raw examples, and Table 1 shows the data distribution of each movie's subtitles.

Table 1: Experimental dataset overview. Overall subtitle images, line images, and word images.

Genre	Title	Subtitle Images	Subtitle Images, Line Split	Subtitle Images, Word Split
Action	John Wick	556	1223	2312
Comedy	Ace Ventura	1051	2469	4389
Drama	The Social Network	2318	5890	11255
Horrow	Scream	1464	2458	3474
Sci-Fi	Ready Player One	1840	4337	7870
Total		**7229**	**17393**	**28284**

3.1.1. Labeling

We acquired open-source SRT (text) files after extracting each movie's primary PGS (image) subtitles. Open-source subtitle files are peer-reviewed, which gave us confidence in using them as a baseline for data labeling.

3.1.2. Cleaning

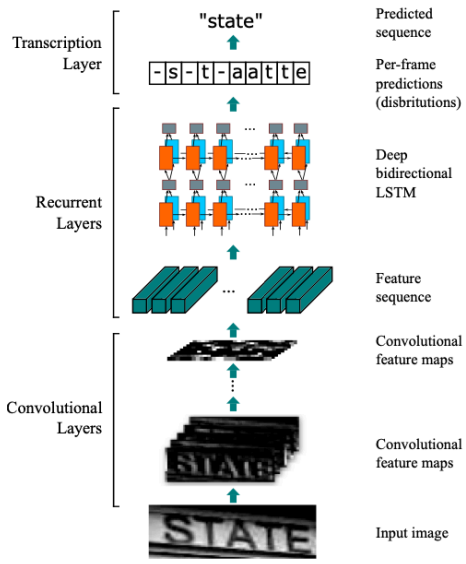
We used pysrt, an open-source Python SRT library, to explore the SRT text files. We performed a variety of checks on each movie's text labels: spelling, grammar, invalid characters, and word counts. Using this method, we identified about 100 text lines that required correction across our entire dataset. Upon making these corrections, we ran our data_processing.py program to create annotation files that associated images with their corresponding scene text, lines, and words.

3.1.3. Preprocessing

To perform a proper OCR, we normalized each image by inverting it to black and white and resizing it. At this point, our data was ready for use in modeling.

3.2. Model

We implemented the 7-layer CRNN precisely as described in SBY's paper. Per the paper, we used a CTC loss function. Figure 2 shows the network architecture and configuration. Figure 3 shows our Tensorflow implementation.



Type	Configurations
Transcription	-
Bidirectional-LSTM	#hidden units:256
Bidirectional-LSTM	#hidden units:256
Map-to-Sequence	-
Convolution	#maps:512, k:2 × 2, s:1, p:0
MaxPooling	Window:1 × 2, s:2
BatchNormalization	-
Convolution	#maps:512, k:3 × 3, s:1, p:1
BatchNormalization	-
Convolution	#maps:512, k:3 × 3, s:1, p:1
MaxPooling	Window:1 × 2, s:2
Convolution	#maps:256, k:3 × 3, s:1, p:1
Convolution	#maps:256, k:3 × 3, s:1, p:1
MaxPooling	Window:2 × 2, s:2
Convolution	#maps:128, k:3 × 3, s:1, p:1
MaxPooling	Window:2 × 2, s:2
Convolution	#maps:64, k:3 × 3, s:1, p:1
Input	$W \times 32$ gray-scale image

Figure 2: Shi, Bai, and Yao's proposed CRNN.
(left) Network architecture, *(right)* Network configuration.

```python
def crnn(input_dim, output_dim, activation="relu", dropout=0.2) -> Functional | Any:
    inputs: Any | list = layers.Input(shape=input_dim, name="input")

    # Layer 1
    conv_1: Any | None = Conv2D(64, (3, 3), activation=activation, padding="same")(inputs)
    pool_1: Any | None = MaxPool2D(pool_size=(2, 2), strides=2)(conv_1)

    # Layer 2
    conv_2: Any | None = Conv2D(128, (3, 3), activation=activation, padding="same")(pool_1)
    pool_2: Any | None = MaxPool2D(pool_size=(2, 2), strides=2)(conv_2)

    # Layer 3
    conv_3: Any | None = Conv2D(256, (3, 3), activation=activation, padding="same")(pool_2)

    # Layer 4
    conv_4: Any | None = Conv2D(256, (3, 3), activation=activation, padding="same")(conv_3)
    pool_4: Any | None = MaxPool2D(pool_size=(2, 1))(conv_4)

    # Layer 5
    conv_5: Any | None = Conv2D(512, (3, 3), activation=activation, padding="same")(pool_4)
    batch_norm_5: Any | None = BatchNormalization()(conv_5)

    # Layer 6
    conv_6: Any | None = Conv2D(512, (3, 3), activation=activation, padding="same")(batch_norm_5)
    batch_norm_6: Any | None = BatchNormalization()(conv_6)
    pool_6: Any | None = MaxPool2D(pool_size=(2, 1))(batch_norm_6)

    # Layer 7
    conv_7: Any | None = Conv2D(512, (2, 2), activation=activation)(pool_6)
    squeezed: Any | None = Lambda(lambda x: K.squeeze(x, 1))(conv_7)

    blstm_1: Any | None = Bidirectional(LSTM(128, return_sequences=True, dropout=dropout))(squeezed)
    blstm_2: Any | None = Bidirectional(LSTM(128, return_sequences=True, dropout=dropout))(blstm_1)

    outputs: Any | None = Dense(output_dim + 1, activation="softmax")(blstm_2)
    model = Model(inputs=inputs, outputs=outputs)

    return model
```

Figure 3: Our Tensorflow implementation of the CRNN.

3.2.1. Training

We used a MacBook Pro to train the model and relied on the tensorflow-metal package to enable GPU acceleration. We manually set the model's initial parameters but could not obtain an acceptable result, so we wrote a simple program to train the model under various epochs and batch sizes to reach our final result.

4. Results

We achieved excellent results under the limitations in our computing power and time frame for model development. Table 2 shows the model performance, and Figure 4 shows inference examples at the word and line level.

4.1. Examples

Figure 4: Inference results in title bar. *(top)* Word model, *(bottom)* Line model.

Table 2: CRNN Model performance. Word and line models.

CRNN Model	Epochs	Batch Size	Character Error Rate	Word Error Rate
Words	20	3	2.8%	6.5%
Lines	10	3	1.9%	12%

4.2. Discussion and Limitations

We were surprised to discover how long it takes to train neural networks. Our initial trials used batch sizes of 64, but our models could not converge at that size, and the training time was significant. We found through experimentation that on our machine, a batch size of 3, with 10-20 epochs and a learning rate of 1e-4 produced the best results. We will experiment with running long-duration training after the semester to see what results we can achieve.

Unsurprisingly, both versions of our CRNN have lower character error rates than word error rates. The likelihood of getting one character incorrect in a sequence of them is higher.

As seen in Figure 4, our line-level model cannot insert spaces between words yet. The ability to insert spaces during line-level predictions is an area of work we will pursue in the future.

5. Future Work

Our goal is to ultimately create an open-source CLI application for robust subtitle transcoding via OCR. As such, we have several future works planned.

5.1. Expand the experimental dataset

Through this experiment, we found that PGS subtitles vary significantly in their styling. Horror movies, like “Scream,” are styled more ominously and have less easily detectable edges. Sci-fi movies like Ready Player One are styled with crisp edges but often use italic fonts. Future work includes incrementally increasing our experimental dataset to ensure our model is generalizable across movie genres and eras.

5.2. Add support for extended ASCII characters

Many films have subtitles that contain accented characters in the extended ASCII domain. Due to time limitations, we could not build support for these characters into our model. Future work will include encoding and decoding extended ASCII characters for robust English language support and possibly even foreign languages.

5.3. Improve data pre-processing

We intended to explore how fundamental Computer Vision techniques like morphological transformations and boundary detection can improve our model’s performance. Specifically, we want to attempt to create a filter kernel that can detect and un-italicize image text before model training.

5.4. Perform additional model tuning

Finally, due to time constraints, we relied heavily on “out-of-the-box” parameters for a generalized text detection model. Given the narrow domain of subtitle images, we feel there are many tuning optimizations we can make to improve the model’s performance.

6. Conclusion

We successfully implemented a CRNN-based OCR engine based on the paper by Bai, Shi, and Yao [1]. We used our movie collection with open-source subtitles to build a well-annotated experimental data set. In the future, we will attempt to optimize the model for subtitle OCR, add extended-ASCII support, and increase the size of our experimental dataset.

References

[1] B. Shi, X. Bai, and C. 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 & Machine Intelligence*, vol. 39, no. 11, pp. 2298–2304, Nov. 2017, doi: 10.1109/TPAMI.2016.2646371.