DATA 6610
Seth Stock – a02365428 Final Project – Six Pager

Identifying Words from Images Using a Character Sequence Prediction Vision Transformer to Enable the Transcription of Highly Degraded and Handwritten Text

Introduction

Classifying words and characters from images is a major problem within the data science space, with there being many business and general uses for these types of models. There are multiple challenges to tackle, with many off-the-shelf models being capable of general word or general character classification. This is limiting, especially regarding words not contained within the training data. Models with this sort of architecture often struggle with detecting "new" words or characters, resulting in massive and computationally expensive models being required to achieve appropriate accuracy.

There is an opportunity in text transcription from images, particularly regarding low-quality images that contain handwritten, sometimes cursive text, with strange characters, symbols, etc., often from multiple languages. There are troves of valuable historical documents for businesses and more generally that have not been properly digitized. To do this manually, it takes absurd amounts of labor and is not financially feasible for most businesses. However, there is substantial value in maintaining and analyzing these records, and a solution could be highly valuable to historians, governments, and business owners.

This project aims to discover if it is possible to build a model that can successfully identify a word and the associated comprising characters, that is able to identify words not contained in its training set, between multiple languages, character and symbol types, handwritten or not, with varying image qualities.

From initial results, it is indeed possible to build a model that successfully transcribes words from low quality images from numerous languages, with sometimes handwritten or cursive text. Using a hybrid Convolutional Neural Network and Transformer-based sequence decoder, we can capture the features of the strokes and shapes of characters and words with the convolutional encoder, before decoding those features character by character with an attention-based decoder. This allows incredible versatility with word detection, allowing words to be inferred despite unreadable or even missing characters, classification of "new" or novel words, etc. Further training with more compute resources/time may be required to achieve models with higher accuracy, possibly by increasing transformer embedding dimensions above 256.

Approach: Data

The data used for this project is mostly from the TextOCR – Text Extraction from Images Dataset on Kaggle. To augment the dataset, I added a selection of several thousand images of handwritten text, gathered online and from images of my own notebooks, etc. There is a substantial amount of cursive text. Some examples of images include family census records from the late 19th century German Empire, and cursive journal entries. I utilized ChatGPT to help generate bounding boxes and labels for the words in my images, which were then placed appropriately inside the dataset. (JSON) This data is highly suitable for my task, as there are over one million images with an absurd range of variation, combined with my specific handwritten

examples. The largest limitation of the dataset is its size, and subsequent sampling requirement, which made it difficult to ensure that there were enough instances of words and characters in the data for a model to properly train on. Another challenge is addressing the class imbalance of handwritten to printed text, and a further class imbalance for English, Chinese, and Japanese text in the data.

Due to this, the data required significant preprocessing and sampling. First is pulling the proper bounding boxes for words and indexing them to images. Without this, the data is not labeled. Second, to limit RAM usage to less than 16gb, we had to sample just one thousand images from the dataset. After the subset, letters were segmented from the word bounding boxes using CV2. After the dataset of letters had been generated, it was augmented with rotations, zooms, and shifts, so that no letter had less than two hundred corresponding images. This was necessary to address the fact that there are thousands of characters that only appear once in the image subset. This same process was performed for the words dataset, without splitting the words into characters, and with fifty augmentations per word, as there were over 27,000 words. The words and characters datasets were then encoded and split into train, validation, and test sets. In addition to the previous steps, the images were padded to sixty-four pixels in height, and a width of 176 pixels. This was to ensure uniformity across images as well as high enough resolution to detect words and characters, while still being low enough to address the problem set.

Approach: Methodology

To gain a feel for the data and complete the modeling task, I implemented three separate model architectures. The first was a sequential Convolutional Neural Network for character-only detection, the second being a simple character classifying transformer, and the third being the convolutional encoding, multi-head decoding model to predict words and character sequences. The Convolutional model was intended to gain a baseline on how well a simple model could fit to the data before moving on to more advanced and computationally expensive methods. The character classifying transformer was implemented to get over the limitations of the CNN. Finally, the dual-output model is designed to complete the target task, using some of the lessons learned trying to fit the previous character classification models. This provided invaluable insight into the distribution of data and optimal model parameters for the more computationally intensive dual model, drastically cutting down my training time. The dual model is the appropriate choice in this context, as it will allow the generation of words not in the training set, it will be able to capture complex contours and relationships between letters, rather than simply scanning for the presence of those letters, significantly enhancing both its detection and prediction abilities.

The dual model uses convolutional layers that extract the features from the input image. It used a kernel size of six-by-six, embedding each patch into a vector with positional encodings. These embeddings are then sized to a dimension of 512 and passed through eight Transformer encoder blocks with eight attention heads each. The encoder output is a sequence of features representing the image and a CLS token output summarizing the whole word. This is then passed to the decoder, which has the same depth as the encoder, but with sixteen attention heads. The decoder reverts the sequence of features from each step, generating a character sequence and a

word classification in parallel. The forward feeding function means that during inference, it feeds its own last character output through as the input for its next step.

The model training regime takes the dataset of labeled words, with each training sample consisting of a word image, a target word label, and a target character sequence. Batch size was set to 32, with 25 epochs. The training optimizer was the AdamW optimizer for the dual model, as the inclusion of weight decay assists with convergence between word classification and sequence prediction. Learning rate was set to 1e-4 with a scheduler that reduces the rate if validation loss plateaus for more than two epochs. Gradient clipping with norm 1.0 to stabilize training was implemented, as we are combining losses from two outputs. The losses were also weighted, with alpha = 3 for word loss and beta = 0.5 for character loss. The loss function is cross-entropy.

Evaluation metrics that were utilized include Exact Match Accuracy, Character Error Rate, and Word Error Rate. These are computed between the Levenshtein edit distance between predicted text and the true value. This allowed the output of the best and worst ten predications based on Levenshtein similarity scores. Further, confusion matrices and example transcriptions were displayed for qualitative analysis.

Analysis & Results

The dual-output transformer performed surprisingly well for its small model parameters. In the iteration with an embedded dimension of 512 and sixteen attention heads, with eight blocks, I achieved a near perfect character loss of just .008 after 25 epochs. This training regime had an alpha value of 3 and a beta of 0.5 and resulted in a model that detects characters with near-perfect accuracy. Word loss was around 0.98, which indicates an accuracy of around 75-85%, confirmed by the average Levenshtein similarity of 80.48%, which shows that outputs are 80% accurate on average. However, the exact sequence match accuracy was just 19.8%, indicating that the model only got the full sequence of letters right in 20% of predictions. This is quite poor and indicates that the model needs more capacity, or another method to more accurately capture the proper sequences of words. Unfortunately, due to compute limitations, I have been unable to exceed the results described above. With more time, a model with an image sample between ten and one hundred thousand, a kernel size between six and eight, 256 to 1024 embedded dimensions, eight to 64 attention heads, and six to twelve layers would likely perform much better at this task. Further, an alpha value between two and five, with beta frozen at one or lower should be in future training iterations.

As shown below, the model particularly struggled with n's, t's, m's, s', e's, ones, and l's.

```
Most Frequent False Positive Pairs:
True: '1' → Predicted: 'S' | 18 times
True: 'T' → Predicted: 'S' | 16 times
True: 'I' → Predicted: 'T' | 11 times
True: 'S' → Predicted: 'Y' | 10 times
True: '0' → Predicted: '1' | 10 times
True: 'E' → Predicted: 'T' | 10 times
True: 'P' → Predicted: 'E' | 10 times
True: 'H' → Predicted: 'P' | 10 times
True: 'E' → Predicted: 'H' | 10 times
True: 'N' → Predicted: 'E' | 10 times
True: 'S' → Predicted: 'N' | 10 times
True: 'O' → Predicted: 'S' | 10 times
True: 'N' → Predicted: 'O' | 10 times
True: '2' → Predicted: '6' | 10 times
True: 'M' → Predicted: 'S' | 10 times
```

However, from a qualitative analysis, a graph of predicted word vs. true word is quite interesting and further guides my recommendation for a model with more capacity. Shown below, you can see that it does quite well with longer sequences of letters, but "does not know when to stop." From this, I gather that the kernel size should be increased to eight-by-eight, so that the convolutional encoding layer can pick up more general context trends. I think another factor primarily limiting the word prediction capabilities is the small image subset that was required to fit within computer system specifications

Discussion and Interpretation

The central research question for this project asked whether a single vision-transformer architecture—augmented with a convolutional feature stem—could reliably transcribe words from images featuring text that is hand-written, degraded, or entirely absent from the training vocabulary. The results produced by the dual-output model strongly suggest that yes, the

approach is viable, but that data volume and model capacity are now the binding constraints rather than model architecture.

The model classifies characters nearly perfectly, but is very sensitive to over-generation in sequence prediction, where it adds or repeats letters when it loses confidence in where to stop, as discussed above. This data, model, and scope are all appropriate for the task at hand, I just didn't get far enough into testing the model to improve accuracy above this point. From this point forward, it can be done on free cloud resources, but each time you update the training parameters, it takes well over 3-4 hours to train. However, querying the model is quite light on resources.

Several limitations plagued this training run. Particularly, the sample of one thousand images, though substantially augmented, cannot capture the massive breadth of text within the scope of this project. Also, compute restrictions limited embedding size to 512 and attention heads to sixteen, capping sequence-level performance. Since the evaluation set is also extremely limited due to the sample size, true out of domain prediction, like on a degraded Guttenberg bible, remains untested.

Conclusion & Next Steps

This model successfully demonstrates that convolutional encoders paired with transformer decoders can form a single, highly robust, and relatively lightweight model for image to text transcription. While the current iteration struggles with full sequence accuracy, it validates the model architecture and pinpoints next steps to fully address the task of flexible and highly accurate image transcription.

To expand upon and improve model performance, which should be done, several simple steps can be taken. In summary – further dataset scaling is needed, if possible, to upwards of one hundred thousand images. Model capacity should be expanded to 256-1024 dimensional embeddings, eight to 64 attention heads, and larger kernels (eight-by-eight or greater). To do so utilizing free cloud resources, twelve-hour runtimes are required, and converting the model to utilize tensor core processing would significantly decrease training times and allow for a much larger model. Addressing these limitations would move this from a proof of concept to a cost-saving platform for the preservation of a wide array of records, and for novel uses outside this case. For example, scanning license plates at car washes. For our purposes, it could save organizations like banks, governments, libraries and museums, and a range of business hundreds of thousands of dollars on record digitization initiatives.