WRITETHROUGH: HANDWRITING RECOGNITION AND GENERATION WITH TR-OCRS AND CGANS

Ali Zia and Nathan Demssie

ABSTRACT

The following research initiative is an attempt to formalize the handwriting recognition and recommendation process through the chaining of optical character recognition (OCR) models, word embeddings, and conditional general adversarial networks (cGANs). The report will outline a proposed pipeline for converting images of handwriting to handwritten recommendations of the next word in the sentence, tackle strengths and limitations of the proposed pipeline, while also discussing published research initiatives tackling similar issues. Through the chaining of a Transformer-based OCR model, a tri-gram word embedding, and a conditional GAN for generating handwritten letters, our team was able to build a pipeline that successfully provided pseudo-handwritten next-word recommendation for a handwritten sentence.

1 CURRENT LITERATURE

There exists a plethora of documented research on the conversion of handwriting into text, with the most notable example in machine learning being Yann LeCun's work on document recognition and handwritten number parsing using the MNIST dataset (LeCun et al., 1998). However, the opposite conversion (text to handwriting) has only recently increased in popularity with the advent of diffusion models and generative AI tools to generate human-like handwriting outputs. With the applications of cGANs to handwriting generation, new approaches have began to take shape.

1.1 EXISTING OCR AND TR-OCR MODELS

Our team originally opted to utilize OCRs (Optical Character Recognition) in order to convert images of text into machine-readable formats by analyzing visual patterns to identify characters and text structures. OCRs are typically powered by deep learning models like CNNs or transformers (Vaswani et al., 2023), and achieve high accuracy by recognizing complex patterns in fonts and handwriting. We originally looked into leveraging Easyocr and fine-tuning it on different handwritten data such as the IAM Handwriting Database or possibly DeepWriting Database.

However, after various tests, we noted the limited capabilities of OCRs in that they are very good at translating images into text but perform very poorly when trying to translate handwritten images. Specifically when testing the base OCR model it performed well with identifying text within images of signs and license plates but performed very poorly with our handwritten images and other handwritten images. After further research, we came across (Li et al., 2022), which introduces TrOCR, a Transformer-based Optical Character Recognition (OCR) model that uses pre-trained image and text Transformers for end-to-end text recognition.Typically before we mentioned how most OCR methods rely on convolutional neural networks (CNNs) for image understanding and recurrent neural networks (RNNs) for character-level text generation.

This paper, however introduces a way to replace CNNs with a vision Transformer as the encoder and uses a text Transformer as the decoder, simplifying the architecture. Through eliminating the convolutional backbones and the integration of pre-trained models for both vision and language tasks, the model is able to generate word piece units directly, in turn avoiding the need for separate language models.

When looking further into the architecture, the TrOCR employs a Transformer-based encoder to process image patches and a Transformer-based decoder for language modeling. The model first is

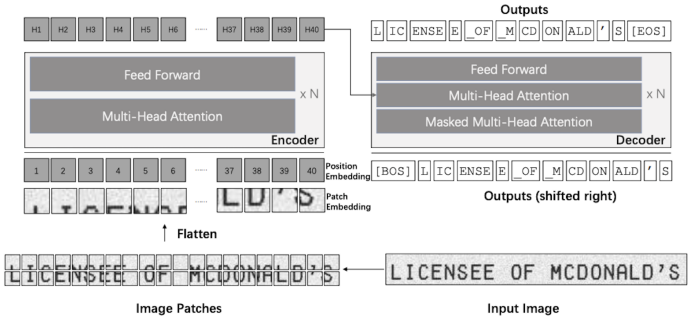

Figure 1: The architecture of TrOCR, where an encoder-decoder model is designed with a pre-trained image Transformer as the encoder and a pre-trained text Transformer as the decoder.

pre-trained within two stages using synthetic datasets for printed and handwritten text recognition, then is further fine-tuned on task-specific datasets. The input images are divided into patches (e.g., 16x16 pixels) and encoded into a sequence of embeddings, enabling the model to focus on specific visual and linguistic features without bias from CNN architectures (Dosovitskiy et al., 2021; Touvron et al., 2021; Bao et al., 2022). When looking at its performance, the model outperforms OCRs through various benchmarks such as printed, handwritten, and scene text datasets while also having higher CER scores compared to traditional and hybrid models. As a result of this our team chose to go ahead with fine-tuning a base TrOCR model.

1.2 Existing text-to-handwriting models

Our team pinpointed two approaches to converting typed text into handwriting: diffusion and generative adversarial networks (GANs).

Diffusion models, which have shown remarkable success in generating high-quality images, are now being applied to generate handwriting. These models work by gradually reversing a process of noise addition, allowing them to synthesize realistic handwritten text from a given typed input. Unlike earlier approaches that relied on simple generative models or rule-based systems, diffusion models capture the subtle variations and personal touch inherent in handwriting, such as slant, stroke thickness, and letter spacing.

One recent publication on diffusion models is DiffusionPen, a few-shot model designed to learn a writer's unique handwriting style from as few as five reference samples (Konnik et al., 2024). DiffusionPen combines a style extraction module with metric learning and classification techniques, enabling it to capture both seen and unseen handwriting styles with minimal examples. This flexibility allows the model to generate new text that closely mimics the input style. Evaluations on handwriting datasets, such as the IAM dataset (later used in our own TrOCR model) and GNHK, show that DiffusionPen can produce diverse and realistic handwriting, matching real-world handwriting distributions (Konnik et al., 2024). Additionally, when used for training Handwriting Text Recognition (HTR) systems, the generated data was shown to improve recognition performance.

GAN-based models have also proved to be effective in generating handwriting from text based on samples, and generally tend to be less computationally expensive than diffusion-based models. A GAN is comprised of two neural networks: a generator and a discriminator. The generator creates synthetic data, while the discriminator evaluates whether the data is real or fake. The two networks are trained simultaneously, with the generator improving its outputs to fool the discriminator, and the discriminator getting better at distinguishing real from fake data (Konnik et al., 2024).

The paper "GANwriting: Content-Conditioned Generation of Styled Handwritten Word Images" introduces a novel approach to generating realistic and diverse handwritten text using a GAN-based framework (Kang et al., 2020). The proposed model produces handwritten word images conditioned on both the textual content and the writer's specific calligraphic style, allowing it to replicate varied handwriting styles even from a few reference samples. The generator is guided by three objectives: creating realistic images, mimicking a specific handwriting style, and conveying the intended text. This mix of style and conveying the meaning of text was essential for pushing our team to inves-

tigate handwriting and typed text not only through GANs and cGANs, but also through OCR and embedding.

After consulting the existing work on text-to-handwriting, our team chose to explore the process through conditional GANs (cGANs). A cGAN adds an additional input condition, such as labels or other data features, to both the generator and discriminator discussed previously.

These GAN models proved to be more lightweight and easier to train on large datasets with less formatting requirements. In addition, the use of GANs to generate words letter-by-letter allowed for more granular control on each letter, which was helpful for our use case of generating a single word rather than sentences or paragraphs.

2 PIPELINE

2.1 FINE TUNED TRANSFORMER-BASED OCR MODEL

Given the TrOCR framework, our team's goal was to adapt one of the earlier stage models in order to recognize handwritten text using the IAM dataset. We chose to use the microsoft/trocr-base-stage1 as it had not yet been trained on the IAM dataset and as such would be the perfect model to start with.In order to handle the data we wrote a custom dataset class, IAM dataset, in order to take care of loading the images and text samples, filtering out any problematic images using the PIL library. Moreover, we preprocessed the data by standardizing the inputs specifically by extracting image features and tokening the text using a TroCRProcessor(which combines a ViTFeatureExtractor to resize and normalize and a RobertaTokenizer (Liu et al., 2019) for encoding/decoding), for each sample the dataset outputs the processed image as a tensor (pixel-values) and the encoded text (labels).

The model we used, VisionEncoderDecoderModel, combines a vision encoder (for feature extraction from images) and a text decoder (to generate the corresponding text).The decoder employs attention masking during training to ensure each position only attends to previous positions, aligning with prediction behavior.The output of the decoder will right shift one place from the input of the decoder, the attention mask needs to ensure the output for the position i can only pay attention to the previous output, which is the input on the positions less than i (Li et al., 2022).

$$h_i = \text{Proj}(\text{Emb}(\text{Token}_i))$$

$$\sigma(h_{ij}) = \frac{e^{h_{ij}}}{\sum_{k=1}^{V} e^{h_{ik}}} \quad \text{for } j = 1, 2, \ldots, V$$

Final outputs are obtained using beam search, with hidden states projected to vocabulary size and probabilities calculated via a softmax function. Specifically, we chose to set up special tokens like decoder-start-token-id, pad-token-id, and eos-token-id to match the tokenizer. We also during inference added beam search to make the model's predictions more accurate. In order to accomplish this we reconfigured parameters like the maximum sequence length, the number of beams, length penalties, and constraints to avoid repeating phrases. Next with training we used Seq2SeqTrainingArguments and implemented different parameters like batch size evaluation strategy, mixed-precision training, and checkpointing to save the model at intervals.

In order to measure the performance of the fine-tuned model we chose to measure the Character Error Rate (CER), which compares the model's predicted text with the actual translation. In order to get this calculation we decoded the model's outputs and compared them to the actual labels. Lastly, we utilized Seq2SeqTrainer in order to train our model such that it would handle everything from the forward and backward passes to evaluation and saving the model, while also managing padding for variable-length inputs using a data collator.

The fine-tuning process adjusted the pre-trained weights of the model to better fit the specifics of handwritten text in the IAM dataset and through minimizing the CER, our team optimized the model's ability to accurately recognize text. The use of beam search during testing helped ensure the text predictions were not only accurate but also high quality. In the end, a successful fine-tuning process led to a lower CER, which showed that the model became much better at understanding and decoding handwritten text.

2.2 WORD EMBEDDINGS LAYER

As far as generating the next word, our team sought a simple yet effective solution for quickly providing a recommended next word based on the content of handwriting samples used to test the TrOCR. Initially, we decided to use bigram models to generate predictions.

A bigram model is a type of n-gram model where predictions are based on the pair of consecutive words in a sequence. It calculates the probability of a word given the preceding word, allowing us to predict the next word by observing the frequency of word pairs in a corpus. The model relies on the assumption that the next word is dependent only on the immediate previous word, which makes it a simple yet effective model for basic text prediction.

In our project, we applied this concept by using the Reuters dataset, which contains a vast amount of text data from the Reuters corpus. We tokenized the text output from the TrOCR into words and then used the Natural Language Toolkit library to generate bigrams from these tokens (Bird & Loper, 2001). This dataset was chosen because of its large size and diverse vocabulary, which helped create a broad model capable of predicting a wide range of next words. Once we generated the bigrams, we counted the frequency of each pair, allowing us to predict the next word in a sequence based on the most frequently occurring bigram that starts with the last word of the input.

Initially, we used the bigram model to predict the next word in a sentence after converting handwriting to text. The model works by examining the last word in the input sentence and looking up the most common word pair that begins with that word. For example, if the last word in the input is "the," the model would suggest the most likely word that commonly follows "the," based on the dataset's frequency of word pairs.

However, we observed that the bigram model provided poor accuracy because it only considers the immediate previous word, which does not always provide enough context for meaningful predictions, especially in longer or more complex sentences.

To address this limitation, we expanded to trigrams, which use the previous two words for prediction. This extended context helped improve the accuracy significantly, making the predictions more contextually appropriate. Trigrams allowed the model to consider a broader context, capturing more nuanced relationships between words, leading to better next-word predictions.

This approach formed a useful middle step in our pipeline. After converting handwritten text to digital text, we used the bigram and trigram models to predict the next word in the sequence. These predictions were then used to generate a coherent word or phrase, which could later be converted back into handwriting. This process helped ensure that the generated text was contextually relevant, enhancing the overall quality and coherence of the handwriting generation.

2.3 CUSTOM CGAN TRAINED ON EMNIST DATASET

Conditional Generative Adversarial Networks (cGANs) extend the functionality of traditional GANs by incorporating auxiliary information, such as class labels, into both the generator and discriminator networks. This conditioning mechanism enables the model to learn not just the distribution of the data but also how the distribution varies across different conditions.

Formally, a cGAN modifies the standard GAN objective by introducing a conditional variable y. The generator $G(z \mid y)$, parameterized by noise z and condition y, aims to synthesize samples x that match the distribution of real data conditioned on y. Simultaneously, the discriminator $D(x \mid y)$ attempts to distinguish real samples x paired with y from fake samples $G(z \mid y)$ (Kang et al., 2020).

The optimization process for cGANs involves solving the following min-max problem:

$$\min_G \max_D \mathbb{E}_{x,y\sim p_{\text{data}}}[\log D(x|y)] + \mathbb{E}_{z\sim p_z, y\sim p(y)}[\log(1 - D(G(z|y)|y))].$$

This conditional framework allows the generator to produce diverse outputs aligned with specific class labels, making it an effective approach for tasks like class-conditional image synthesis.

In our implementation, we applied the principles of cGANs to the problem of generating handwritten letters. Using the EMNIST dataset, which contains grayscale over 100,000 images of handwritten characters and their corresponding labels, we designed a cGAN to generate synthetic handwriting

samples conditioned on letter labels. The EMNIST dataset is an extension of the popular MNIST dataset which includes handwritten letters (Cohen et al., 2017).

The dataset's letters split consists of 27 classes (26 for the letters 'a' to 'z' and one for 'N/A'). To condition the cGAN on these labels, we embedded the label information directly into both the generator and the discriminator. This ensured that the generator learned to produce plausible handwritten letters associated with each label and that the discriminator evaluated the authenticity of the generated samples in the context of their corresponding labels.

The architecture of the generator was designed to map random noise vectors and label embeddings to realistic images. To achieve this, we employed a sequence of fully connected layers with batch normalization and Leaky ReLU activations. Batch normalization was crucial for stabilizing the training process by normalizing the inputs to each layer, while Leaky ReLU activations introduced non-linearity without completely suppressing negative values, enabling better gradient flow. The final layer reshaped the output into a 28×28 grayscale image, followed by a tanh activation to scale pixel intensities to the range $[-1, -1]$, aligning with the normalized dataset. The generator's forward pass is expressed as:

$$G(z, y) = \text{Reshape}(\text{Tanh}(W_k \cdot \cdots \cdot \sigma(W_1 \cdot [z, e(y)] + b_1) + b_k)$$

where z is the latent vector, $e(y)$ is the embedding of label y, W_i and b_i are the weights and biases of the i-th layer, and σ represents Leaky ReLU.

The discriminator's role was to distinguish between real and synthetic samples while incorporating label conditioning. To help the model achieve this, we concatenated the embedded labels with the image data flattened into a single vector. The network consisted of progressively smaller fully connected layers, with dropout layers to prevent overfitting, and Leaky ReLU activations to maintain gradient flow. The output layer produced a single scalar probability through a sigmoid activation, representing the likelihood that the input image-label pair was real. The discriminator's loss function was the binary cross-entropy (Kang et al., 2020):

$$\mathcal{L}_D = -\mathbb{E}_{x,y\sim p_{\text{data}}}[\log D(x|y)] - \mathbb{E}_{z\sim p_z, y\sim p(y)}[\log(1 - D(G(z|y)|y))].$$

Training involved alternating updates to the generator and discriminator using Adam optimizers. The generator focusing on fooling the discriminator into misclassifying fake samples as real. The discriminator, on the other hand, minimized its error in distinguishing between real and fake samples while considering label consistency.

After rotation and processing, the model produced a single letter corresponding to the typed letter, which was then strung together to represent the entire word.

In the process of fine-tuning our previously trained cGAN, we specifically aimed to enhance the model's ability to generate handwritten letter images by training it on a Kaggle dataset consisting of approximately 3,000 images (Dave, 2021). The data was comprised of handwriting samples similar to those in the EMNIST dataset, also in a different handwriting style. This fine-tuning approach is particularly useful when leveraging pre-trained models, allowing us to adjust only certain parameters, retaining the general features learned during the initial training while adapting to the new dataset.

We first identified the layers in both the generator and discriminator models that were crucial for fine-tuning. Instead of freezing many layers, we chose to freeze the first layer of both models, allowing the networks to retain their high-level feature learning while enabling deeper layers to adapt to the new dataset. This is based on the assumption that the early layers typically capture lower-level features like edges and textures, which are applicable across various handwriting styles, while the deeper layers are more specific to the data that we want to iterate on.

The fine-tuning process was carried out with a learning rate of 0.00005, a lower value compared to the initial training to avoid overfitting the small new dataset. We used label smoothing for both real and fake labels during training, setting the real labels to 0.9 and the fake labels to 0.1, to prevent the discriminator from becoming too confident and creating vanishing gradients. The loss functions for both the generator and discriminator were computed using the Adam optimizer with a learning rate of 0.00005 and betas of (0.5, 0.999), which is typical for stabilizing training in GANs.

3 RESULTS AND ANALYSIS

3.1 FINE TUNED TRANSFORMER-BASED OCR MODEL

After training for several epochs, the model achieved a progressively lower CER score, indicating an improved transcription accuracy for handwritten text recognition. While fine-tuning the model we kept checkpoints at every 200 iterations and found that the training loss continually decreased (signifying the models improving ability to fit to the training data), the validation loss also continued to decrease (signaling that the model is not just memorizing the training data but is actually able to make accurate predictions on new and previously unseen examples). When looking at Li et al. (2022) which also fine-tuned one of the earlier models upon the IAM dataset, we see that the TrOCR small had a CER of 4.22 percent, the TrOCR Base had a CER of 3.42 percent while the TrOCR Large had a CER of 2.89 percent. Our fine-tuned model achieved a CER of 5.1 percent.One thing to however note is that both the fine-tuned model and the TrOCR Base model struggled to accurately transcribe images containing multiple lines of handwritten text.This limitation likely stems from challenges in handling spatial dependencies across lines, as the models were primarily optimized for single-line inputs or lacked robust mechanisms for multi-line contextual understanding. Moreover further research and experimentation with line segmentation or architectural adjustments, such as incorporating hierarchical context modeling, may be helpful and necessary to improve performance on multi-line handwritten text.

Model	Architecture	Training Data	External LM	CER
(Bluche and Messina 2017)	GCRNN / CTC	Synthetic + IAM	Yes	3.2
(Michael et al. 2019)	LSTM/LSTM w/Attn	IAM	No	4.87
(Wang et al. 2020a)	FCN / GRU	IAM	No	6.4
(Kang et al. 2020)	Transformer w/ CNN	Synthetic + IAM	No	4.67
(Diaz et al. 2021)	S-Attn / CTC	Internal + IAM	No	3.53
(Diaz et al. 2021)	S-Attn / CTC	Internal + IAM	Yes	2.75
(Diaz et al. 2021)	Transformer w/ CNN	Internal + IAM	No	2.96
$\text{TrOCR}_{\text{SMALL}}$	Transformer	Synthetic + IAM	No	4.22
$\text{TrOCR}_{\text{BASE}}$	Transformer	Synthetic + IAM	No	3.42
$\text{TrOCR}_{\text{LARGE}}$	Transformer	Synthetic + IAM	No	2.89

Figure 1: Evaluation results (CER) on the IAM Handwriting Database.

After the final step of training, a checkpoint was created, which stored various deployment pieces essential for model inference and further fine-tuning. With these files we were able to upload and deploy the model's current state with all necessary components and build a Hugging Face Transformer model, which was then deployed on the website, enabling seamless integration of the trained OCR model for real-time transcription of handwritten text.

Figure 2: Various metrics used to evaluate the model

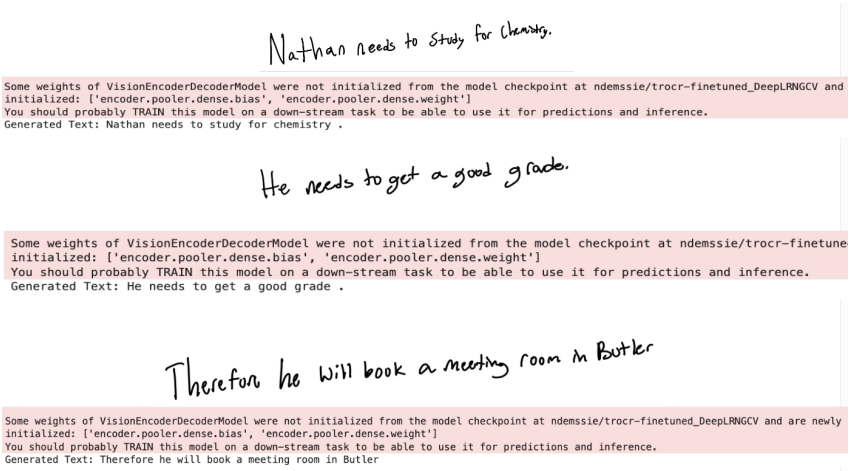

Figure 3: Examples used to show recognition functionality

3.2 WORD EMBEDDINGS LAYER

When looking back at the results of the model, we saw a 66% accuracy score on the 1,000 test sentences from the Reuters dataset, which in turn demonstrates the effectiveness of n-gram models in predicting the next word in a sequence. Moreover it suggests that the trigram-based prediction model, with a fallback to bigram predictions, captures meaningful context within short text sequences. However, we can also reflect and note that performance is not perfect, demonstrating the limitations of n-grams in handling more complex linguistic structures and nuances in natural language.

Several reasons could contribute to the observed accuracy. First, the Reuters dataset comprises well-structured and formal text, which favors n-gram models as they thrive on repeated patterns in structured language. Another reason could be the fallback mechanism ensures that predictions remain robust even when trigram data is unavailable. However, n-grams inherently struggle with sparsity issues, especially for less frequent word combinations, and lack the ability to generalize beyond seen sequences. These limitations, combined with the deterministic nature of choosing the most frequent continuation, may lead to errors in scenarios requiring nuanced understanding or context-sensitive interpretation.

Another perspective might see drawbacks in the dataset itself. The Reuters corpus is filled with academic/formal words and one improvement would be to use a more informal general data set to pair with the informal handwritten text. Moreover a more advanced embedding step would likely improve the applicability of the pipeline.

3.3 CUSTOM CGAN TRAINED ON EMNIST DATASET

After training the custom cGAN using the 27 classes of letters provided by the EMNIST dataset, the model exhibited differing performance based on the complexity of the strokes in the letters. The cGAN performed relatively well in generating images for simple letters, such as ”l” and ”o,” that involve minimal strokes. However, the model struggled to produce accurate representations for letters with more intricate strokes, suggesting that the cGAN's capacity to generate visually complex handwriting may be limited by the data/architecture.

A conditional GAN operates fundamentally differently from an image classifier, as it is tasked with generating images rather than categorizing them. The discriminator's role is to differentiate between real and generated images. As such, the discriminator became central to evaluating the model's performance. The accuracy of the model was therefore primarily assessed by examining the discriminator's ability to distinguish real images from fake ones. We first provided the discriminator with a set of fake images generated by the cGAN, followed by real images from the EMNIST dataset, and measured its accuracy in distinguishing between them.

The discriminator was able to classify real images with an accuracy of 74.76%, while its accuracy on fake images was slightly lower at 70.02%. The overall discriminator accuracy was around 72.39%. These results suggest that the cGAN was able to generate reasonably convincing fake images. There is still room for improvement, however, especially for generating high-quality handwriting for more complex letters. The discrepancy in performance between simple and more complex characters highlights a potential area for further model refinement and a more data-rich cGAN.

It is important to note that **the fluctuation in the cGAN generator's loss is likely a result of the adversarial setup**, where both the generator and discriminator continuously learn and adapt in response to each other's performance. When the generator output becomes more convincing, the discriminator task becomes more difficult, and it adjusts to improve at distinguishing real from fake images. This back-and-forth can cause the generator's loss to oscillate as it constantly adapts to the changing discriminator.

After fine-tuning using a Kaggle dataset with 3,000 images of hand-drawn letters, the model initially produced results that were highly noisy and lacked clarity (Dave, 2021). The relatively small size of the new dataset, compared to the much larger dataset the model was originally trained on, posed challenges in achieving meaningful results. The model also struggled to adapt to the distinct characteristics of the new images, leading to inconsistency in the outputs and sometimes unrecognizable characters. During fine-tuning, the discriminator's accuracy fluctuated significantly, ranging from as low as 20% to as high as 60%. The new dataset contained images with variations in style,

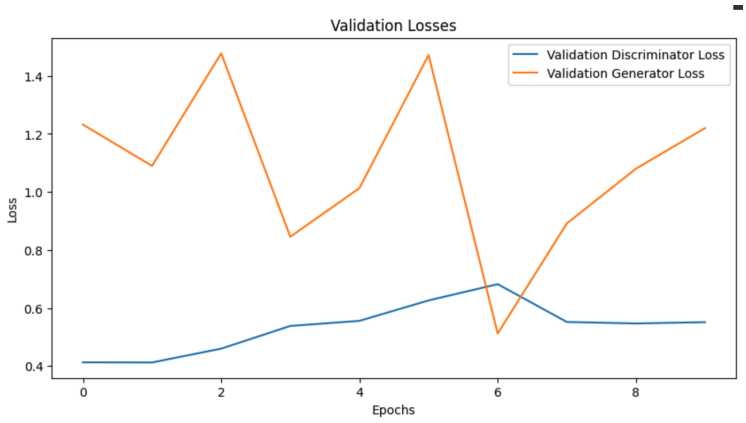

Figure 4: Discriminator and generator loss for cGAN

quality, and noise that were not present in the original EMNIST data, making it more difficult for the fine-tuned model to generate accurate representations of handwriting.

Figure 5: Sample image output for prompt ”words”

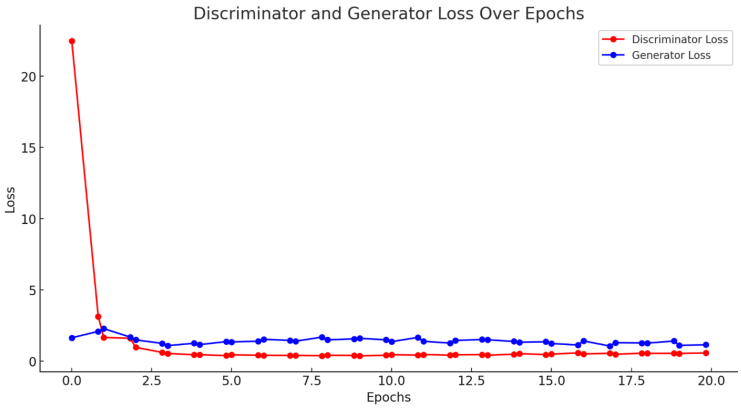

Figure 6: Discriminator and generator loss for fine-tuned cGAN

4 Conclusion

4.1 TrOCR

Looking back on our progress, our exploration of Optical Character Recognition (OCR) tools and techniques has culminated in a promising fine-tuning process of the TrOCR model for handwritten text recognition (Dosovitskiy et al., 2021). While traditional OCRs like EasyOCR showed significant limitations in handling handwritten text despite their proficiency with printed and scene text,

TrOCR offered an innovative approach that leverages the strengths of Transformer architectures in both vision and language tasks. By eliminating the convolutional backbones of traditional models and incorporating pre-trained Vision and Text Transformers, TrOCR demonstrated superior accuracy, precision, and efficiency across various text recognition benchmarks.

The decision to fine-tune the microsoft/trocr-base-stage1 model on the IAM dataset allowed our team to adapt TrOCR's powerful capabilities specifically for handwritten text recognition. Moreover, it allowed us to compare the results shown within Li et al. (2022) where they also finetune a pretrained TrOCR with the IAM Handwriting Database. Through custom dataset preparation and preprocessing, including the use of the TroCRProcessor for feature extraction and tokenization, we ensured high-quality input for the model. Moreover, by employing the VisionEncoderDecoderModel, we integrated a vision encoder for image feature extraction and a text decoder for sequence generation, while also employing several techniques such as attention masking, beam search, and strategic parameter tuning which in turn further refined the model's training and inference capabilities.

To test our fine-tuned model, we used the Character Error Rate (CER) as a performance metric, focusing on minimizing discrepancies between predicted and actual text. The implementation of the Seq2SeqTrainer streamlined the training process, managing crucial components like padding for variable-length inputs and checkpointing to save progress. These strategies enabled us to achieve CER of 5.1 percent which is competitive to other state-of-the-art models such as the TrOCR small which had a CER of 4.22 percent, the TrOCR Base which had a CER of 3.42 percent and lastly the TrOCR Large which had a CER of 2.89 percent.

In retrospect, this project highlights the potential of TrOCR and similar transformer-based architectures in overcoming the challenges posed by handwritten text recognition. The success of our fine-tuning process not only validated the efficacy of TrOCR but also underscored the importance of carefully crafted preprocessing, training, and evaluation pipelines in achieving state-of-the-art performance in OCR tasks. This work lays a solid foundation for future advancements in recognizing complex text patterns across diverse applications.

4.2 Word embeddings

With respect to the embedding model, we integrated the handwritten text recognition with the next-word prediction in order to form a coherent text pipeline, in turn highlighting the strengths and limitations of n-gram-based models in this context. By leveraging the TrOCR model for handwriting recognition and combining this with the trigram models for next-word prediction, we were able to effectively create this pipeline and coupled with the Reuters dataset we were able to build an efficient word-pair frequency model, which scored a 66 percent accuracy on 1000 test sentences.

While the fallback mechanism between trigram and bigram predictions ensured a degree of robustness, the inherent sparsity of n-grams and their reliance on observed sequences limited the model's ability to generalize effectively. This was specifically shown with the Reuters dataset which somewhat lacked the diversity needed to mirror the variability found in informal handwritten text.In order to enhance the pipeline's performance, we hope to incorporate more diverse and informal datasets that better align with the characteristics of handwritten input. Also we would like to replace or augment the n-gram approach with advanced word embeddings or transformer-based language models in order to improve the contextual relevance and generalization capabilities of the predictions. Embedding techniques like Word2Vec, GloVe, or even fine-tuned versions of GPT could bridge the gap between deterministic frequency-based predictions and nuanced natural language understanding.

4.3 Custom cGAN trained on EMNIST dataset

The results of our conditional GAN (cGAN) experiment demonstrated both the potential and the challenges of using generative models for handwriting synthesis. While our model was able to replicate simpler letters with reasonable fidelity and generated letters with a high accuracy, it struggled with complex characters and achieving stylistic consistency. Notably, fine-tuning our model with a small dataset presented further difficulties, particularly due to the increased noise and varying accuracy, emphasizing the need for high-quality, diverse data to improve our cGAN.

The work of Kang et al. (2020) provides compelling evidence of the strength of cGANs for granular handwriting generation. Their research, mentioned previously, focused on generating full words in varying handwriting styles, showcasing cGANs' ability to synthesize visually coherent and stylistically diverse outputs. By using datasets such as the IAM dataset and the Brown corpus, their team ensured that the model was trained on data with significant variability and complexity to adapt to different kinds of writing styles.

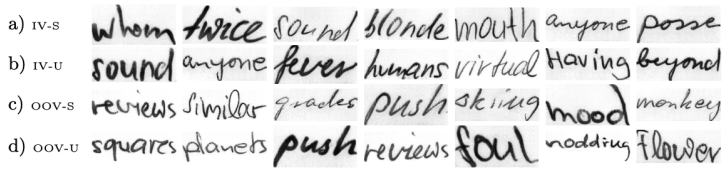

Figure 7: Generated words with varying styles from the GANwriting model (Kang et al., 2020)

These insights suggest promising avenues for improving our pipeline. The IAM data, for example, includes segmented word-level handwriting samples with different styles, while the Brown corpus offers a textual basis for word and sentence-level generation that is more integrated into the writing generation compared to having a separate embedding layer. Since the IAM dataset was also used in our TrOCR model, integrating this data to also generate handwriting would likely help the pipeline in being more consistent. Utilizing these kinds of datasets for fine-tuning and enhancing our embedding technology could help our model better capture the subtleties of character stroke formation and achieve greater stylistic range.

In addition, a GitHub repository that also showcased an attempt at applying similar GANs to words was able to manually string together letter outputs to include line breaks and spaces (Inamdar et al., 2024). This additional layer of processing on top of the cGAN outputs could make the results more readable and useful for the user. Our team hopes that further iterations on the model and data exploration with approaches like this could lead to a more human-friendly handwriting result.

AUTHOR CONTRIBUTIONS

Ali focused on the implementation of both the conditional GAN (data allocation, training, and fine-tune from scratch) as well as the bi-gram and tri-gram testing and evaluation.

Nathan focused on fine-tuning, and evaluating and packaging the TrOCR model while also building out the tri-gram/bi-gram model functionality.

REFERENCES

Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. Beit: Bert pre-training of image transformers. 2022. URL https://arxiv.org/abs/2106.08254.

Steven Bird and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2001. URL https://www.nltk.org/book/ch02.html.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre van Schaik. EMNIST: an extension of MNIST to handwritten letters. *arXiv*, 2017. URL http://arxiv.org/abs/1702.05373. arXiv:1702.05373.

Dhruvil Dave. English handwritten characters dataset, 2021. URL https://www.kaggle.com/datasets/dhruvildave/english-handwritten-characters-dataset. Accessed: 2024-12-08.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. 2021. URL https://arxiv.org/abs/2010.11929.

Amogh Shreedhar Inamdar, Anagha M. Rajeev, and Sindhuja V. Rai. Handwriting-gan: Letter lsgan, 2024. URL https://github.com/AmoghSInamdar/Handwriting-GAN/tree/master. Accessed: 2024-12-07.

Lei Kang, Pau Riba, Yaxing Wang, Marçal Rusiñol, Alicia Fornés, and Mauricio Villegas. Ganwriting: Content-conditioned generation of styled handwritten word images. *arXiv*, 2020. URL https://arxiv.org/abs/2003.02567. arXiv:2003.02567.

Vadim Konnik, Julien Bieder, Karol Kurach, Jiasen Deng, David Berthelot, and Antoine Parisot. Diffusionpen: Towards controlling the style of handwritten text generation. *arXiv*, 2024. URL https://arxiv.org/abs/2409.06065. ECCV 2024.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

Minghao Li, Tengchao Lv, Jingye Chen, Lei Cui, Yijuan Lu, Dinei Florencio, Cha Zhang, Zhoujun Li, and Furu Wei. Trocr: Transformer-based optical character recognition with pre-trained models. 2022. URL https://arxiv.org/abs/2109.10282.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. 2019. URL https://arxiv.org/abs/1907.11692.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. 2021. URL https://arxiv.org/abs/2012.12877.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2023. URL https://arxiv.org/abs/1706.03762.