HANDWRITTEN PHRASE SYNTHESIS USING GENERATIVE ADVERSARIAL NEURAL NETWORKS

Aniruddh Aragola **Nabeth Ghazi** **Ran (Andy) Gong** **Zerui (Kevin) Wang**

ABSTRACT

This report outlines an implementation of a Generative Adversarial Network that attempts to generate realistic-looking images of handwritten text from ASCII text input along with a text Recognizer model. We present architecture, training, data, and a discussion of our model's qualitative and quantitative results on different datasets. The IAM handwriting Database is processed for training and validation, while the model is tested on a set of randomly generated and handwritten Shakespeare text. We also prepare baseline models for the recognizer and generator to gauge our primary model's performance. We also utilize Fréchet Inception Distance and Character Error Rate metrics to evaluate different sub-networks within the GAN. The final result of the project is a Generator that does not perform as expected, and a successful Recognizer. Potential ethical considerations for this project are also addressed.

—-Total Pages: 9

1 INTRODUCTION

Computer-generated content has become common in our daily lives. From passages generated from ChatGPT, to artwork produced by Midjourney. Computers are producing contents closer and closer to authentic human-produced works. For this project, our team has chosen to add to this and produce a network to synthesize images of realistic handwriting corresponding to a text prompt input.

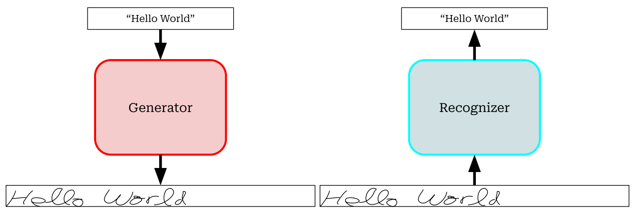

Figure 1: Overview of the goal of the project. Generator to convert strings to handwriting images, Recognizer to convert handwriting images to strings.

Our end goal is to make and train a Generative Adversarial Network (GAN) that take a string of ASCII text (letters, numbers, and some common punctuation) and output images of that text being "hand-written". To reach this goal, one sub-task is to train a text Recognizer that can decode an image of handwriting back into ASCII text to ensure the model is producing matching images to input text. Both goals are shown in Figure 1. This project can help people who are losing capability to manually write texts and help them produce more synthetic copies of their handwriting on new text contents. The Recognizer itself can also be used to decode handwritten mails/letters to help computers catalogue them.

Deep learning is a good approach to this problem. Due to the existence of multiple handwriting styles and that they can vary even within the same text, it is difficult to "hard-code" any specific rules or algorithms that can reliably produce authentic outputs. This is true for both handwriting generation and recognition. Therefore, a deep learning approach should be used to "learn" the handwriting styles to produce good outputs.

This project faces many challenges, such as GANs being complicated and computationally expensive models, requiring much longer time to train than a conventional neural network and can have trouble converging. Furthermore, the sub-task of Recognizer itself is a difficult project to complete.

2 Background & Related Work

Adversarial Generation of Handwritten Text Images Conditioned on Sequences — (Alonso et al., 2019) presents a GAN capable of producing images of singular French or Arabic words. The modified GAN used contains an Encoder-Generator-Discriminator-Recognizer architecture while also using many commonly-used GAN sub-networks such as ResBlocks and Conditional Batch Normalization. They employ CTC loss to train text Recognizer and uses Geometry Score and Fréchet Inception Distance to evaluate the model performance. The result of this work is producing legible images of French and Arabic words corresponding to their text labels.

Generating Sequences with Recurrent Neural Networks — (Graves, 2014) presents a text generation LSTM and adapted it to synthesize complex sequences via data point prediction. They use the IAM-OnDB, an online handwriting database (IAM Group, 2005) consisting of samples from 221 writers where data is represented by a ordered sequence of xy-coordinates from tracking pen tips. The results from various regularization and normalization techniques have shown to improve the network to the point of indistinguishability.

Realistic Handwriting Generation Using Recurrent Neural Networks and Long Short-Term Networks — (Bodapati et al., 2020) presents various deep learning techniques to generate realistic human-like handwriting images. The model they use is reversing the network of a handwriting detection system. They used RNNs and networks that takes in one-hot encoding of letters and outputs an sequence of pen movements which “draws” the letters. To do this, they used LSTM cells, an attention mechanism and Mixed Density Networks. The network is trained using the IAM dataset, and the end result is capable of “writing” readable sentences with few mistakes.

Generative Adversarial Network for Handwritten Text — (Ji & Chen, 2019) presents another GAN based approach for handwriting generation. They used a Discriminator consisting CNN and LSTM reading in handwriting data, and a Generator that produces sequential handwritten data through recurrent networks. This model treats handwriting information as series of “strokes“, which are straight lines connecting points of the image. The network starts at a point and continuously predicts the next “stroke” point. It produced two networks, one to generate random texts, and one to generate texts corresponding to an input. The result is a GAN synthesizing handwriting via “digital ink” that outperforms previous works.

Results of a PyTorch implementation of an Handwritten Text Recognition Framework — (Barrere et al.) presents a PyTorch based handwritten text recognition model. They uses a CRNN architecture containing convolution layers, pooling, dropout, and stacked Bidirectional LSTM. The model is trained using CTC loss and evaluated using Character Error Rate and Word Error Rate through the IAM dataset and ICFHR 2018 READ dataset. Despite not accomplishing a perfect result, they demonstrated that their model can still achieve good performance in text recognition.

3 Data Processing

The data we used come from the IAM handwriting database (Marti & Bunke, 1999) created by people hand writing phrases from a Corpus (Stig et al., 1978). The database cropped the handwritten texts letter by letter into labeled PNGs of words, lines, and sentences (Marti & Bunke, 2002).

We use the lines dataset containing lines of various lengths. First, we removed all the samples labeled `"err"` which denotes some errors when processing images. We also removed data containing non-alphanumeric characters other than `.,!?:'-"` and space to maintain data’s simplicity.

We will be downsizing the images to a size of 32×512 (to reduce memory usage), so we remove images with wider aspect ratio of 1:16 since we cannot easily produce matching text labels for cropped images.

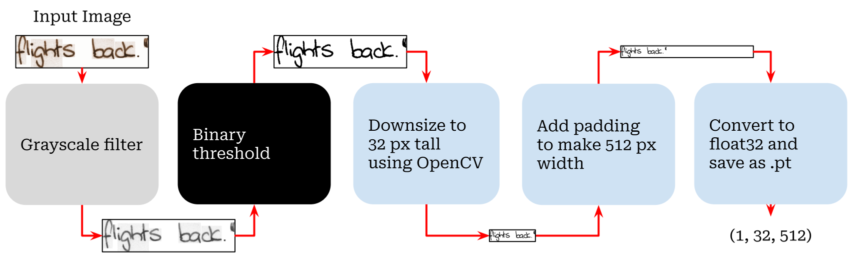
Input Image
Grayscale filter
Binary threshold
Downsize to 32 px tall using OpenCV
Add padding to make 512 px width
Convert to float32 and save as .pt
(1, 32, 512)

Figure 2: High level overview of the image processing applied to all the selected PNGs.

We grayscale and binarize the PNGs according to the IAM database's information and downscale them to a height of 32 using OpenCV. We then add white padding to the right side of the image to produce $1\times32\times512$ grayscale images.

The text labels are converted to an array of character indices, where we add `0` to the end of texts as padding to the length of the longest valid text which has 82 characters.

In total, we have 7 135 valid data of matching texts and images. Out of which, we used 2 500 training and 500 validation data for training of the main GAN model. We used less sample than available due to computer hardware and training time constraints.

For Recognizer training, we use the full dataset due to it being a smaller model and can thus learn faster. We applied an 80%-20% split between the training and validation dataset. For training sample, we try to improve the resilience of the network by data augmentation. For texts ≥ 30 characters, we horizontally compress them by a random factor between 0% and 50%. For texts < 30 characters, we horizontally stretch them by the same random factor. Each sample has 3 augmented copies, creating a total 22 832 training samples, along with 1 427 validation samples. We choose to stretch short text and compress long text to prevent them from exceeding the 32×512 size.

4 Architecture

The primary model for handwriting synthesis uses a modified Generative Adversarial Network (GAN) comprised of an Encoder, Generator, and Discriminator. Along with it we developed a Recognizer network trained prior to the primary model to detect text contents in a handwriting image. Encoder converts a string text into an embedding, Generator uses embedding and a noise to produce handwriting images, Discriminator classifies between real and fake (generated) images to encourage realistic image production, and Recognizer detects text in generated images to encourage images to match the intended text. Encoder, Generator and Discriminator, and Recognizer are shown in Figures 6, 7, and 8, respectively, with overview shown in Figure 3.

The primary model and its sub-networks takes inspiration from previous works: (Alonso et al., 2019) (Brock et al., 2019) (Zhang et al., 2019) (de Vries et al., 2017) (Bluche & Messina, 2017) (Barrere et al.).

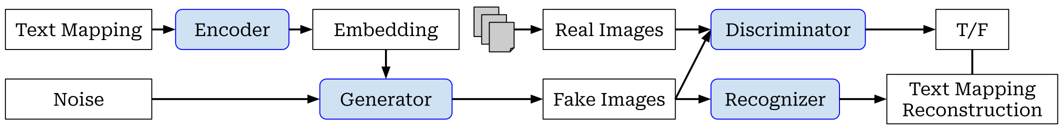

Figure 3: High-level overview of how different components interact in our network to form the full architecture

4.1 Sub-Networks

We also implemented some sub-networks as a part of the main networks: `ConditionalBatchNorm2d`, `ResBlockUp`, `ResBlockDown`, `ResBlock`, and `SelfAttention`.

`ConditionalBatchNorm2d`: This network takes in an input and condition tensor. It applies regular batch normalization with `affine=False` to input. It produces one tensor (with length twice the number of input's channel) by passing condition tensor through linear layer with hidden layer size of 512. This tensor is split into bias and scale for each input channel, and used to scale and shift the batch-normalized input. Architecture is shown in Figure 4.

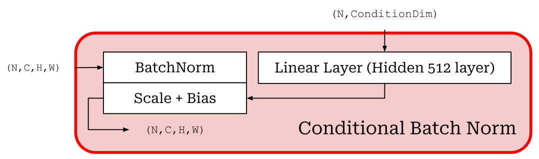

Figure 4: Architecture of Conditional Batch Normalization network

`ResBlocks`: There are `ResBlockUp`, `ResBlockDown`, and `ResBlock` shown in Figure 5. `ResBlockUp` applies skip connection via 1×1 convolution and "nearest" $2\times$ upsampling. The input is also passed through Conditional Batch Normalization (CBN) with a condition tensor, "nearest" $2\times$ upsampling, 3×3 convolution with padding 1, another CBN, ReLU, and 3×3 convolution with padding 1. This output is summed with the output of the skip connection to produce the final output. `ResBlockDown` replaces CBN with regular batch normalization, upsampling with 2×2 average pool, and ReLU with leaky ReLU using leak 0.01. `ResBlock` is same as `ResBlockDown` without pooling.

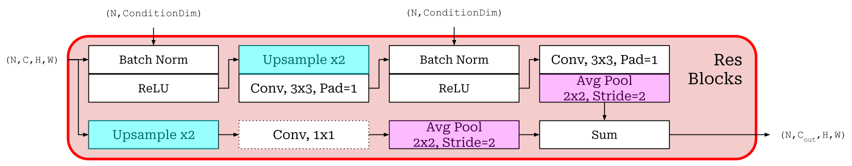

Figure 5: Architecture of the Res Blocks. ResBlockUp replaces Batch Norm with Conditional Batch Norm, and does not use pooling. ResBlockDown replaces ReLU with Leaky ReLU and does not use upsample. ResBlock does not use any upsample or pooling, and uses Leaky ReLU. Regular Batch Norm layers does not require condition tensor.

`SelfAttention`: This layer is taken directly from (Zhang et al., 2019), which can allow the model to pay more "attention" to features that are distant from each other in an image.

4.2 Encoder

The Encoder receives input in `(N, 82)` with each value representing index of a character in the 82-character long text. It uses `nn.Embedding` to encode each character to a 128-long tensor, and passes this through a bidirectional LSTM with hidden-dimension 256 and 6 layers. Outputs a `(N, 512)` tensor after max pooling.

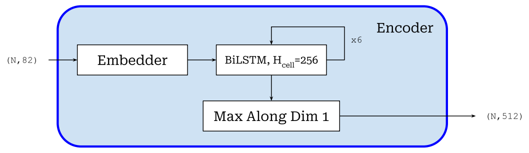

Figure 6: Architecture of the Encoder

4.3 GENERATOR

The Generator receives input noise in `(N, 96)` and embedding in `(N, 512)`. The noise is split into 6 chunks of `(N, 16)`. It applies one linear layer to convert one noise to a tensor reshaped into `(N, 256, 1, 16)`. This 1×16 image is upsampled by 5 `ResBlockUp` using one new noise chunk for each `ResBlockUp` concatenated with embedding as the condition tensor. Between 3rd and 4th `ResBlockUp`, we apply `SelfAttention`. The `ResBlockUp` has output channel of 256, 128, 64, 32, 16. The output of last `ResBlockUp` is passed through batch normalization, ReLU, 3×3 convolution with padding 1 with output channel 1. The final output is passed through sigmoid to produce an image.

4.4 DISCRIMINATOR

The Discriminator receives input a `(N, 1, 32, 512)` image. It applies 5 `ResBlockDown` to the image with output channels 16, 32, 64, 128, 256, applying `SelfAttention` between layer 2 and 3. It then applies one `ResBlock` before summing values along dimension 2 and 3, passing it through a fully connected layer into 1 value, and passing the sigmoid function to produce probability the image is real.

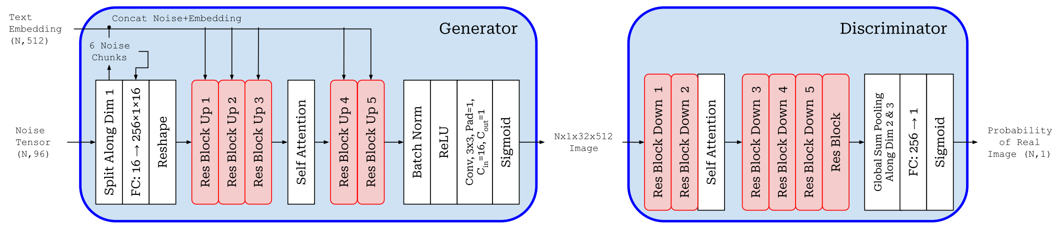

Figure 7: Architecture of the Generator and Discriminator

4.5 RECOGNIZER

Inspired by (Barrere et al.), The recognizer receives a input image of size `(N, 1, 32, 512)`. It applies three 3x3 convolutional layers, then applies a `(4, 2)` convolutional layer. Each convolution layer has output channels 8, 16, 32, and 64. We apply 20% dropout to 3rd and 4th layer. The output tensor goes through a 4-layer bidirectional LSTM with hidden size 128, which outputs a `(N, 256, 505)` tensor. This output is sent through a 256 to 73 and a 505 to 82 linear layer to reshape it to `(N, 73, 82)`. After applying log-softmax to the tensor, the output represents the probability of a possible character (out of 73 options) being at each of the 82 positions in the text.

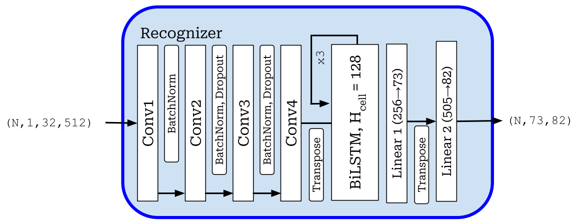

Figure 8: Architecture of the text image Recognizer

4.6 TRAINING HYPER-PARAMETERS

We trained the Recognizer separately prior to the training of the Encoder, Generator, and Discriminator. For Recognizer training, we used the Adam optimizer with learning rate 5e-4, batch-size 8, and betas (0, 0.999). The loss function we tried to minimize is `nn.NLLLoss`. We trained for a total of 50 epochs before stopping the training after noticing increasing validation errors past epoch 47.

To train the Encoder, Generator, and Discriminator, we used the Adam optimizer with learning rate 2e-4 for Encoder and Generator, and 1e-5 for Discriminator, batch-size 32, and betas (0, 0.999). We tried to minimize `nn.BCELoss` between Discriminator prediction of fake/real images with fake/real image labels. We also minimize `nn.BCELoss` for the Encoder and Generator between the Discriminator's output of generated images versus real image labels. The lower learning rate for Discriminator prevents it from improving too fast and "overpowering" the Generator. We trained the model for 100 epochs and chose epoch 34 as the final model because it has the lowest validation Fréchet Inception Distance (see Section 6) after the model trained for over 20 epochs.

5 Baseline Model

For image generation, the baseline model is a hand-coded heuristic model where we hand write individual characters of the alphabet, crop the images of each character, and concatenate the images to match a line of text. We also add between 1 to 2 pixels of white space between characters, and between 12 to 15 pixels of white space for spaces.

For the Recognizer baseline model, we use Tesseract, which is an Optical Character Recognition (OCR) that first locates lines of text, chops the image into letters and classifies characters by detecting features (Smith, 2007)(Smith, 2009)(Unnikrishnan & Smith, 2009)(Smith et al., 2009)(Shafait & Smith, 2010). We used Pytesseract which is a wrapper for Tesseract Engine in python. It includes additional features by using LSTM, but mostly still uses OCR.

We included some outputs of the baseline models in Figure 9, which will be compared with the main model in section 9.

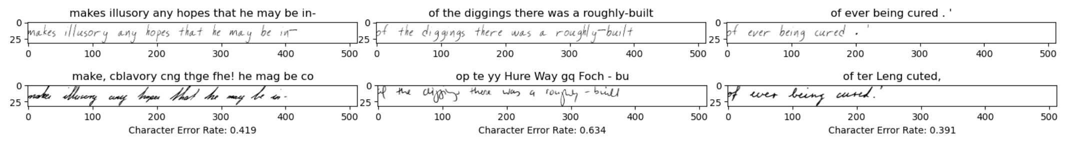

Figure 9: Qualitative results of the baseline models using validation data. Top row consists the text input to the Baseline Generator as title and generated image from the Baseline Generator. Bottom row consists recognized text content from the Baseline Recognizer as title, input image, with Character Error Rate below the input image.

6 Quantitative Results

To quantitatively evaluate the model, we used different measurements for different networks.

For Generator, we use Fréchet Inception Distance (FID), a popular metric for evaluating GANs. It compares the distribution of characteristics between generated and real images and calculates a "distance" between such distributions (Obukhov & Krasnyanskiy, 2020). The lower the distance, the better the generated images. Torchmetrics FID with `feature=2048` is used to compute this. Our chosen Generator has validation FID: **347.1**

For Discriminator, we use a simple accuracy measurement which describes the percentage of predictions between real and fake is correct. The Discriminator corresponding to the chosen Generator has validation accuracy: **1.000**

For Recognizer, we use Character Error Rate (CER), calculating the percentage of characters that were predicted incorrectly. The Recognizer used to train the Generator has an validation error of **0.3763**.

We show the training and validation error/accuracy graphs in Figure 10.

For comparison, our baseline Generator has FID of **273.9** on the images they generated, and the baseline Recognizer error rate of **0.4887**.

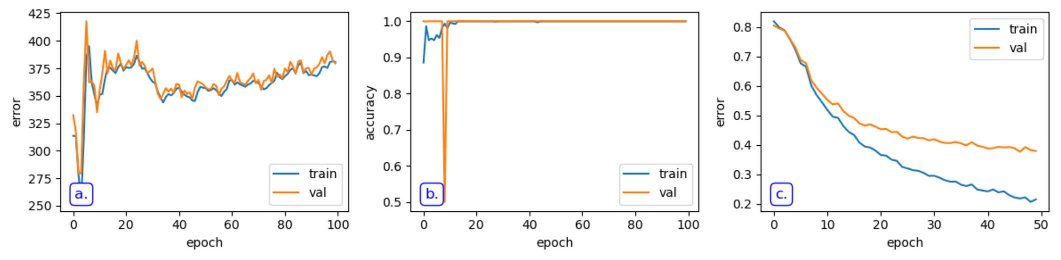

Figure 10: Training and validation error / accuracy graphs during training process of each network. **a.** FID of the Generator. **b.** Accuracy of the Discriminator. **c.** CER of the Recognizer.

7 Qualitative Results

To show Generator qualitative results, we randomly selected 3 images generated by the best chosen model with input text from all validation samples. This is a straightforward way to directly show if the model is producing images we expect, and why the model is having an FID of **347.1**.

For the Recognizer, we selected the same 3 data as the ones we used for Generator qualitative results. The only difference is that we use the image validation samples instead of text validation samples. This helps conceptualize the quantitative CER of the model and how "close" it is reading from the texts.

Qualitative results from both networks are shown in Figure 11.

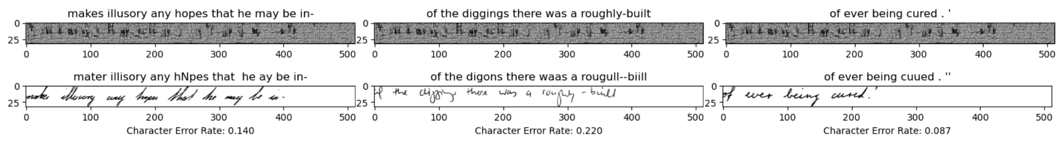

Figure 11: Qualitative results of the final network using validation data. Top row consists the text input to the Generator as title and generated image from the Generator. Bottom row consists recognized text content from the Recognizer as title, input image, with CER below the input image.

The baseline model's qualitative results are shown in Figure 9, showing the baseline's output on the same input data for comparison.

8 Evaluate Model on New Data

To demonstrate the model's performance on never-seen-before data, we sampled texts from Shakespeare, which the model has never seen since the IAM database contains British texts produced in 1961 (Marti & Bunke, 1999) (Stig et al., 1978). We input the text into Encoder and Generator, while we evaluate the Recognizer by using images of our team member's own handwriting of the Shakespeare texts, which went through similar pre-processing as the dataset used before.

The result of passing new data through the network yielded Generator FID **347.8** and Recognizer error **0.3968**. Randomly selected generated images and text recognition are shown in Figure 12.

9 Discussion

The validation Generator FID of **347.1** shows that the Generator model is performing poorly. Generally, an FID of higher than 200 indicates that the generated images have significant distortion and that they are very far from the expected "real" images. Since our model's validation FID is **347.1**, it shows that the generated images are not close to what we want the Generator to produce. The testing FID is **347.8**, which is only slightly higher than the validation FID, showing that the model is not overfitting, but very likely underfitting.

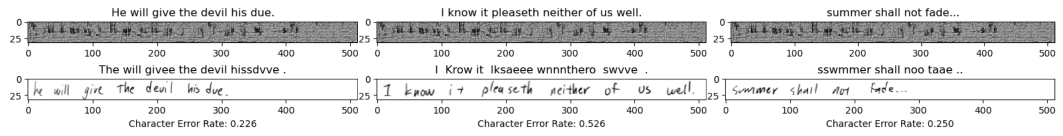
He will give the devil his due.
I know it pleaseth neither of us well.
summer shall not fade...
The will givee the devil hissdvve .
I Krow it Iksaeee wnnnthero swvve .
sswmmer shall noo taae ..
Character Error Rate: 0.226
Character Error Rate: 0.526
Character Error Rate: 0.250

Figure 12: Qualitative results of the final networks on Shakespeare texts. Top row consists the text input to the Generator as title and generated image from the Generator. Bottom row consists recognized text content from the Recognizer as title, input image, with CER below the input image.

Looking at the qualitative results produced by the model for validation and testing from Figures 11 and 12, we can see visual confirmations that the Generator does not produce recognizable texts, which explains the high Generator FID. Furthermore, since FID compares the feature distribution of images, the fact that all the produced images look virtually identical regardless of input prompt can further contribute to the high FID score, since FID values diversity in the sample images.

Comparing our Generator's results with the baseline Generator shows that our Generator is underperforming compared to baseline. This can be seen through the baseline FID of **273.9** being significantly lower than our Generator's FID of **347.1**. Qualitative results also supports this since, even though images produced by the baseline generator has repeated characters, it is clearly legible and will not be seen as a computer-generated image at first glance.

It is interesting to note that the model output, despite not containing fully legible characters, forms some uniformly distributed black "blobs" or partial strokes at locations where characters commonly appears. This could be an indication that the model is only memorizing some features of characters, but not capable of replicating legible texts. A possible reason could be that these black "blobs" are where the "average" characters appears so the Generator was producing them as an attempt to "trick" the Discriminator. Another interesting phenomenon to discuss is the identical-looking generated images. This could indicate that the Generator does not allow the "condition" tensors (that is the noise and text embedding) to influence the model output sufficiently.

A potential reason for the Generator to not perform well is the fact that the Discriminator, despite having its learning rate tuned down, is still "overpowering" the Generator network. This can be seen from the discriminator accuracy graph in Figure 10b, where the discriminator quickly reaches an accuracy of 1.0 and never goes down from there. When this happens, it is possible that the Discriminator captures features that benefits it, but in consequence causes these features, useless for the Generator, to be backpropagated anyway. This could lead the Generator to try converging in an non-ideal manner. We have noticed this during training, where the image background in generated images are turning more and more grey, causing less distinct textual features and lead to the rising FID for Generator as can be seen in Figure 10a.

For the Recognizer, the validation CER of **0.3763** shows that only 37.63% of the characters are predicted incorrectly, indicating that the Recognizer is correct on most portions of the input. Furthermore, our Recognizer model had a CER of **0.3968** on images it has never seen before, with both the content (the model has never read Shakespeare texts) and writing styles (the model has never been trained on our group member's handwriting before) being brand-new to the network. Even though the test error is slightly higher than the validation error, it is to be expected since the model was never tuned for this data, and the fact that the model is still capable of recognizing most of the input image's texts shows that the model is capable of generalizing. When compared to the baseline Recognizer error of **0.4887**, we can see that our model outperforms the baseline.

Reading what our model is outputting for validation, we can see that the CER of **0.3763** for validation slightly understates the accuracy of the network. We can see that sometimes the model predicts characters correctly, but they are shifted by one space, or that sometimes the model predicts individual characters incorrectly but instead predicts some characters extremely close to the expected character. In Figure 11, the Recognizer predicted "rougull–billl" while the real text was "roughly-built", which is extremely close, but the CER would still increase the error rate as if it predicted some completely irrelevant text. Looking at the test qualitative results in Figure 12, we notice another phenomenon that the model seems to be recognizing mostly the earlier portion of the text correctly, while failing to recognize later portions of the text. This could be due to the fact that our data set always contains characters in the earlier portions of the text, but not all data contains text in the later parts of the text, causing the network to not be "trained enough" for longer texts. This is supported by our line dataset having a mean label length of **41.2** characters with standard deviation of **10.8** characters. One more interesting result of the Recognizer worth discussing is that in the test qualitative results in Figure 12, there are a lot of repeated characters being predicted. One possible reason is that the image dataset has writing styles with various sizes, so the model might be attempting to decode multiple characters from the same pixel groups, causing the same character to be repeatedly recognized.

Overall, we found this project to be quite challenging, and more difficult than expected. One of the main challenges was the sheer size of the GAN and the difficulties of tuning the hyperparameters to balance all three of the main sub-networks. While the generator, discriminator, and recognizer architectures were all functional on their own, balancing the discriminator and recognizer to produce a useful loss function and selecting the right learning rates proved to be extremely difficult. Since the network is so sensitive to the these hyperparameters, it was very challenging to tune each sub-network to make them compatible with each other. The size also made the network very time-consuming to train - even using features such as CUDA, the full GAN often took tens of hours to train, which limited the range of hyperparameters we were able to try.

Despite these challenges, we were able to produce a working model for each sub-network, and our recognizer outperforms the Pytesseract baseline model.

10 Ethical Considerations

Some ethical issues that may arise from the use of this model includes:

Forgery: People could be using a handwriting synthesis model to replicate handwriting contents of others and fake documents (signatures, letters, journals, etc.). Even though the current model does not produce perfect images, it is still a possibility with an improved model.

Representation Bias: The training data presents ethical concerns, as the data we used (IAM dataset) contains only British texts in 1961, which may not accurately represent the vast diversity of what people write around the world. Furthermore, the data is handwritten by a few select individuals, likely not originated from diverse cultural / educational backgrounds, meaning the model may fail to produce / recognize the diversity of handwriting styles of all populations. It is essential to recognize the potential bias originating from the training data.

Privacy Concerns: When training the model to produce or to recognize handwriting images, the model "memorizes" the input training data. Which could potentially lead to the model "leaking" characteristics of the providers of the handwritings, such as their mental health, physical strength, education level, etc. Which could be possible to decode by some other tools. Also, there is another possibility of the model outputting near-exact content of the training data, hence leaking the original handwriting / text contents used to train the model.

REFERENCES

Eloi Alonso, Bastien Moysset, and Ronaldo O. Messina. Adversarial generation of handwritten text images conditioned on sequences. *CoRR*, abs/1903.00277, 2019. URL `http://arxiv.org/abs/1903.00277`.

Killian Barrere, Aurélie Lemaitre, and Bertrand Coüasnon. URL `https://perso.eleves.ens-rennes.fr/people/killian.barrere/papers/Results_of_a_PyTorch_implementation_of_an_Handwritten_Text_Recognition_Framework.pdf`.

Théodore Bluche and Ronaldo O. Messina. Gated convolutional recurrent neural networks for multilingual handwriting recognition. In *14th IAPR International Conference on Document Analysis and Recognition, ICDAR 2017, Kyoto, Japan, November 9-15, 2017*, pp. 646–651. IEEE, 2017. ISBN 978-1-5386-3586-5. doi: 10.1109/ICDAR.2017.111. URL `https://doi.org/10.1109/ICDAR.2017.111`.

Suraj Bodapati, Sneha Reddy, and Sugamya Katta. *Realistic Handwriting Generation Using Recurrent Neural Networks and Long Short-Term Networks*, pp. 651–661. 03 2020. ISBN 978-981-15-1479-1. doi: 10.1007/978-981-15-1480-7_55. URL `https://link.springer.com/chapter/10.1007/978-981-15-1480-7_55`.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis, 2019.

Harm de Vries, Florian Strub, Jeremie Mary, Hugo Larochelle, Olivier Pietquin, and Aaron C Courville. Modulating early visual processing by language. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/6fab6e3aa34248ec1e34a4aeedecddc8-Paper.pdf`.

Alex Graves. Generating sequences with recurrent neural networks, 2014. URL `https://arxiv.org/pdf/1308.0850.pdf`.

IAM Group. Iam handwriting database, 2005. URL `https://fki.tic.heia-fr.ch/databases/iam-on-line-handwriting-database`.

Bo Ji and Tianyi Chen. Generative adversarial network for handwritten text. *CoRR*, abs/1907.11845, 2019. URL `http://arxiv.org/abs/1907.11845`.

Urs-Viktor Marti and H. Bunke. The iam-database: An english sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, 5:39–46, 11 2002. doi: 10.1007/s100320200071. URL `https://www.researchgate.net/publication/226662568_The_IAM-database_An_English_sentence_database_for_offline_handwriting_recognition`.

Urs-Viktor Marti and Horst Bunke. A full english sentence database for off-line handwriting recognition. *Proceedings of the Fifth International Conference on Document Analysis and Recognition. ICDAR '99 (Cat. No.PR00318)*, pp. 705–708, 1999. URL `https://www.researchgate.net/publication/2461096_A_full_English_sentence_database_for_off-line_handwriting_recognition`.

Artem Obukhov and Mikhail Krasnyanskiy. Quality assessment method for gan based on modified metrics inception score and fréchet inception distance. In Radek Silhavy, Petr Silhavy, and Zdenka Prokopova (eds.), *Software Engineering Perspectives in Intelligent Systems*, pp. 102–114, Cham, 2020. Springer International Publishing. ISBN 978-3-030-63322-6.

Faisal Shafait and Ray Smith. Table detection in heterogeneous documents. In David S. Doermann, Venu Govindaraju, Daniel P. Lopresti, and Premkumar Natarajan (eds.), *Document Analysis Systems*, ACM International Conference Proceeding Series, pp. 65–72. ACM, 2010. ISBN 978-1-60558-773-8. URL `http://dblp.uni-trier.de/db/conf/das/das2010.html#ShafaitS10`.

Ray Smith. An overview of the tesseract ocr engine. In *ICDAR '07: Proceedings of the Ninth International Conference on Document Analysis and Recognition*, pp. 629–633, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2822-8. URL `http://www.google.de/research/pubs/archive/33418.pdf`.

Ray Smith. Hybrid page layout analysis via tab-stop detection. In *ICDAR '09: Proceedings of the 2009 10th International Conference on Document Analysis and Recognition*, pp. 241–245, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3725-2. doi: http://dx.doi.org/10.1109/ICDAR.2009.257. URL `http://www.google.de/research/pubs/archive/35094.pdf`.

Ray Smith, Daria Antonova, and Dar-Shyang Lee. Adapting the tesseract open source ocr engine for multilingual ocr. In Venu Govindaraju, Premkumar Natarajan, Santanu Chaudhury, and Daniel P. Lopresti (eds.), *MOCR '09: Proceedings of the International Workshop on Multilingual OCR*, ACM International Conference Proceeding Series, pp. 1–8. ACM, 2009. ISBN 978-1-60558-698-4. doi: http://doi.acm.org/10/1145/1577802.1577804. URL `http://www.google.de/research/pubs/archive/35248.pdf`.

Johansson. Stig, Geoffrey Leech, and Helen Goodluck. Manual of information to accompany the lancaster-oslo : Bergen corpus of british english, for use with digital computers. 1978. URL `http://korpus.uib.no/icame/manuals/LOB/INDEX.HTM`.

Ranjith Unnikrishnan and Ray Smith. Combined orientation and script detection using the tesseract ocr engine. In Venu Govindaraju, Premkumar Natarajan, Santanu Chaudhury, and Daniel P. Lopresti (eds.), *MOCR '09: Proceedings of the International Workshop on Multilingual OCR*, pp. 1–7, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-698-4. doi: http://doi.acm.org/10.1145/1577802.1577809. URL `http://www.google.de/research/pubs/archive/35506.pdf`.

Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks, 2019.