האוניברסיטה העברית בירושלים
THE HEBREW UNIVERSITY OF JERUSALEM

Department of Computer Science

BSc Computer Science and Statistics with Emphasis on Data Science - Final Project

Academic Year 2020 - 2021

Explainability of Adversarial Generative Networks

Author: Roei Brudo

Advisors: Dr. Matan Gavish

A report submitted in partial fulfilment of
the requirements for the degree of Bachelor of Science

The Hebrew University of Jerusalem
Department of Computer Science

Abstract

In recent years, wherever you may look, you will come across a machine learning model. It significantly becomes an integral part of various industries revolving all parts of life. The catch is - machine learning models became much more complex and less intuitive to understand.

State-of-the-art models contain billions of parameters. A lot of real word integration of these models face major trust issues – how can one know that the model performs in the right way? How can we ensure that the model, for example, classify based of the features that 'really' matter and not on some bias within the data? (Amazon, for example, shut down a risk prediction model that is discovered to be biased towards woman). Explainable AI (xAI) is a relatively new area that try to answer exactly these questions, and to extract insights that are understandable to humans.

In this project, I have implemented a few xAI methods for GAN – machine learning framework that produce generative model. It was trained for the task of handwriting generation. I will try to examine (qualitatively) the kind of results from each one of the explainability methods by using a dummy dataset (MNIST). Next, I will explore these methods on a ScrabbleGAN model, which is able to generate handwritten text in various lengths and styles.

Contents

List of Figures

Chapter 1

Introduction

Explainable artificial intelligence (xAI) is a relatively new field exploring methods that allow humans to get a better understanding of the results of an AI system.

There are two main approaches to achieve this kind of transparency. First, Building models with an interpretability aspect within their structure. Second, Post-Hoc methods that are agnostic to the model, and explains it by exploring its behavior.

This work is an implementation of these ideas for Generative Adversarial Networks architecture, trained to generate handwritten text. I will try to explore a few aspects of these machines - what are the different styles that it can produce? What are the characteristics of the style that can be disentangled?

1.1 Aims and Objectives

The main goal of this work is to review a few of the latest works in the GAN explainability field. They will be implemented on a ScrabbleGAN model - a method that generates handwritten text with variate lengths and styles.

The methods that are implemented in this project are:

1. Post-Hoc methods - Unsupervised Discovery of Interpretable Directions in the GAN Latent Space.

2. GAN structures that gain explainability - GAN with attention Blocks and InfoGAN.

These tools will allow to generate handwritten texts with the following insights:

1. Latent space exploration

2. Explore the attention that is given to features while generating each pixel.

Chapter 2

Related work

2.1 Explainable AI (xAI)

Explainable AI is a growing field that comes with the need of humans to understand, or trust the models used. There are two main ways to get this goal:

1. Agnostic methods, that query the model (potentially an exponential number of times) and draw conclusions from the results (methods as SHAP, LIME)

2. Interpretable models - models with an interpretable process that can be explored in order to obtain further insights. For example, linear regression is defined by its coefficients. Observing these coefficients reflect the effect of the input features. Markov Chains models, as HMM, model an inner process that could later be explored. Usually, there is a trade-off between the model's complexity and their interpretablity.In recent years, the state of the art models have grown far more complex, making them uninterpretable.

in the context of generative models, xAI contribution might have two aspects - both explore the abilities of the model and the generation process.

2.1.1 Attention Models

The attention mechanism was initially introduced on a machine translation task [**7**]. The idea was to give the model the ability to filter, by itself, solely the relevant parts of the input at any point in time. In the original paper, it was done by adding another layer to the recurrent network. After the forward backward passes, the model weights all time steps outputs while generating every word. This is called the additive attention scheme.

The idea developed with the emergence of the transformer model [**8**] which uses the same key idea - give the model the ability to filter the relevant data needed to generate the output. This time, the authors use the "Scaled Dot-Product Attention", which can be implemented much more efficiently, and is able to capture a more complex dependencies in the data.

This is the same type of attention that will be used in the ScrabbleGAN model and will be elaborated on chapter 5.

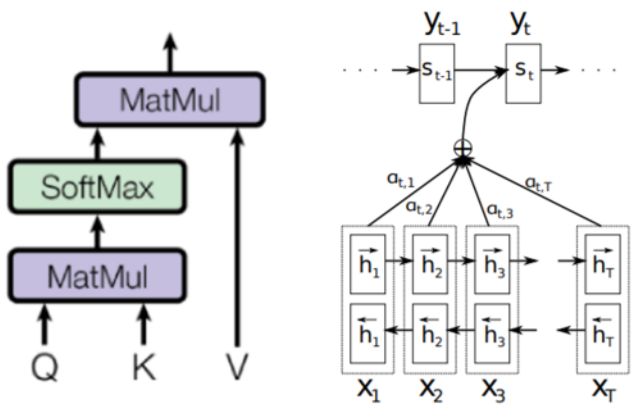

Figure 2.1: Right:Additive attention head Left:Scaled dot-product attention head

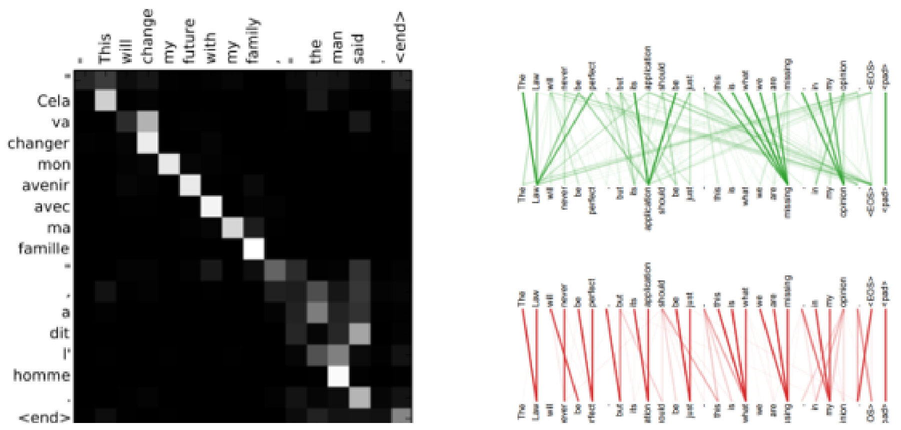

Figure 2.2: Example of features maps produced by exploring the attention mechanisms. Right: maps from additive attention in machine translation. Left: maps from two different scaled dot-product attention heads. Its clear hat parts of the input effects every output component.

2.2 GAN models

GAN model, first proposed in 2014 [**9**], is a generative model framework that consists of two networks that compete with each other - the Discriminator network tries to separate real images from fake ones, while the Generator network tries to generate real look-alike images. GAN produces SOTA results in many fields, and it comes as no surprise that it has been adapted to solve many problems - from generating images (and its derivatives as super resolution tasks) to simulating complex physics systems.

BiGAN is an extension of GAN, that adds another component to the classic GAN framework - Encoder. We train it to predict the random noise that was used to generate the image. The generator needs to maximize information that flows from the structure of the latent space to the image. We try to decode the latent space from the generated image, and by doing that we 'enforce' the generator to learn a better representation of the structure. This process is supposed to make the latent space more interpretable - a meaningful representation means a successful encoder.

Another way to extract insights from the latent space is to enforce the latent

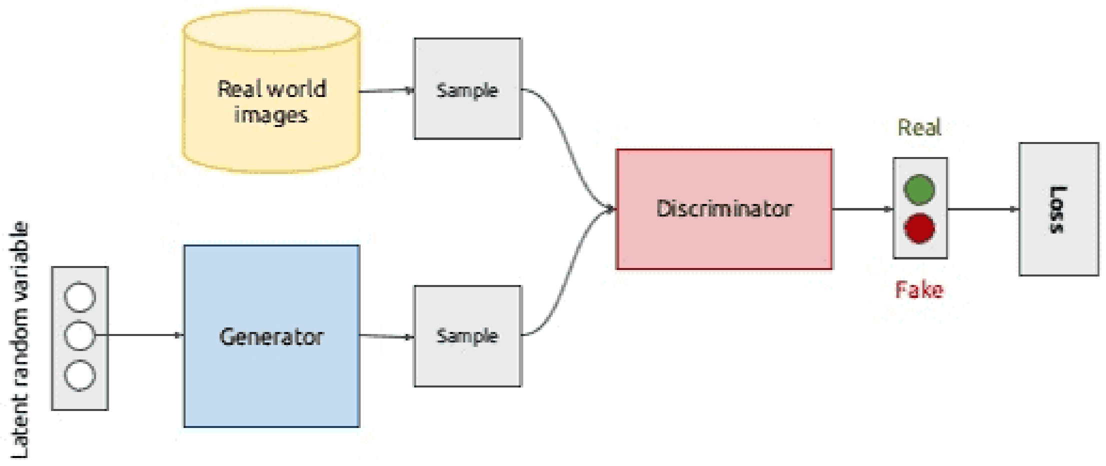

Figure 2.3: GAN model scheme [12]

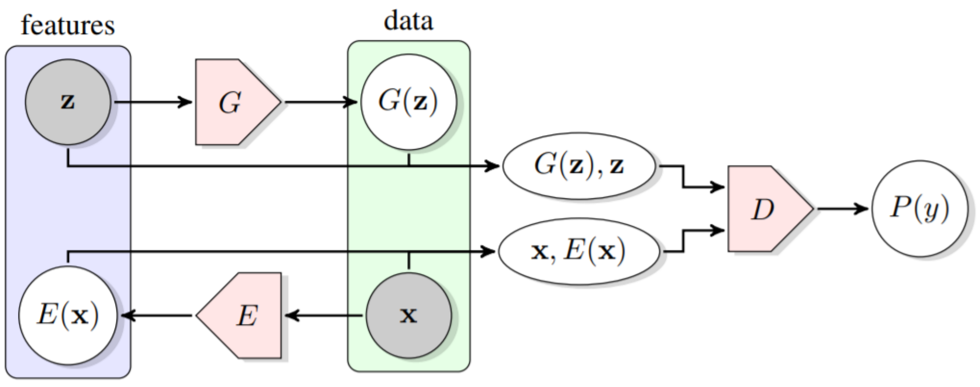

Figure 1: The structure of Bidirectional Generative Adversarial Networks (BiGAN).

Figure 2.4: BiGAN model scheme [12]

space to a certain meaningful structure (as one with discrete structure). InfoGAN [12] is an example of merging these two ideas, which will be elaborated on chapter 5. Unsupervised Discovery of Interpretable Directions in the GAN Latent Space is also based on the same idea. This method is agnostic. The algorithm tries to find directions in the latent space that have meaningful effect in the output space.

2.3 Generating Handwritten Text

Handwriting generation is a relatively new field. It was first introduced by Graves [3], who synthesized online handwriting data (coordinates in 2D space) with RNN. Other works, as DeepWriting [4] extended the same idea, adding more components to the model such as latent space sampling.

The output of these models is a list of probabilities. For each time step, the model generates the distribution of the next point's coordinates.

The main contribution of Graves paper is the use of CTC - Connectionist Temporal Classification [5] measurement as a loss function that can be used to train generative models.

CTC measures the probability of text given the whole sequence generated.
Y is the text, X is the image, and A is the group of all legal alignments between X and Y.

The loss is alignment free, which means we don't need a segmented sequence as an input. Segmenting handwritten data is not trivial due to areas in the input data which don't belong to a single character. Therefore, finding a loss term that can deal with that characteristic of the data is crucial.

ScrabbleGAN [**6**] is a handwriting generation model, that in contrast to Graves work, uses offline data (images of handwritten texts) instead of online data. It also uses the same CTC loss technique, this time over slices of the image instead of coordinates.

Chapter 3

Data

3.1 MNIST

MNIST data contains 70,000 samples of handwritten labeled digits [**1**]

3.1.1 Pre-processing

MNIST images have fixed dimensions (28 X 28 X 1), and therefore the only preprocessing is normalizing the range of pixels values from [0, 255] to [-1, 1].

3.1.2 Visualisation

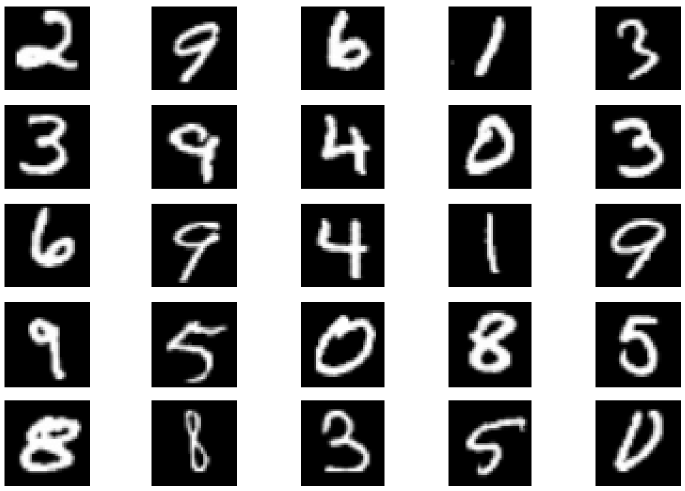

Figure 3.1: MNIST random sample
The digits variate in style - we can see different widths, orientations and curvatures for every image.

3.2 IAM

IAM data contains approximately 115,000 images of label handwritten words. [**2**]

3.2.1 Pre-processing

Words images were re-sized to the size of (16n, 32), where n is the length of the word (number of letters in the label). The idea here is to artificially make each character have an approximate dimensions of (16, 32) in the generated image. Outliers (images with much longer or shorter width of 16n) were removed, as they are not the typical handwriting that we want to examine in the scope of this project. Also, as stated before, the pixels values was normalized to a range of [-1, 1].

3.2.2 Visualisation

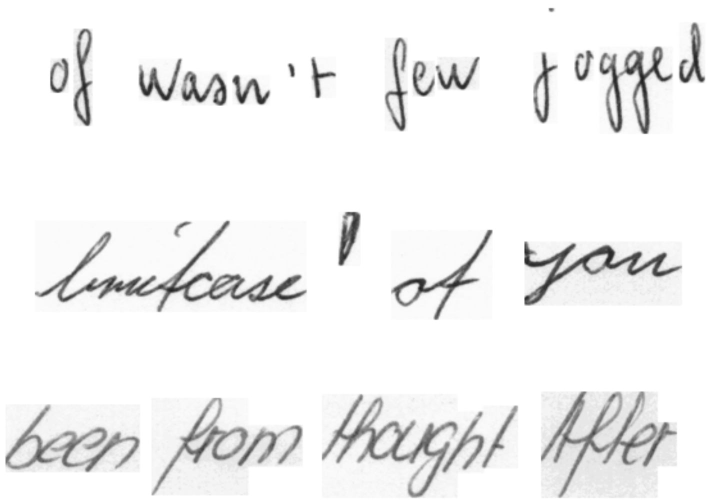

Figure 3.2: IAM random sample
Each row is a sample from a different writer. Notice the inter-group variance vs the intra-group variance - the way the style differs from one writer to another, and the difference between the same letters by the same writer. We want to capture both of these variances in our modeling process.

Chapter 4

Approach

4.1 Project structure

The main goal is to implement the methods mentioned above and to explore their results on MNIST data set, which is a simpler version of handwritten text data set. The idea is to see the qualitative results on a simple use case. I would expect this part to work flawlessly - the results should give a se....

Next, ScrabbleGAN model will be implemented, and the methods will be adjusted to fit this use case. The results of these methods will be explored on the handwriting generation task. Will the results in this case also be meaningful? If not, why is that and what can be improved? Are the reasons for failure inherent to the method, or is it the implementation details that should be taken more carefully?

4.2 Performances and Measurements

The tricky part of explainability is the struggle to define a good explanation. It depends on the use of the explanation. Does it have to be human interpertable, or should it focus on certain aspects of the generative process?

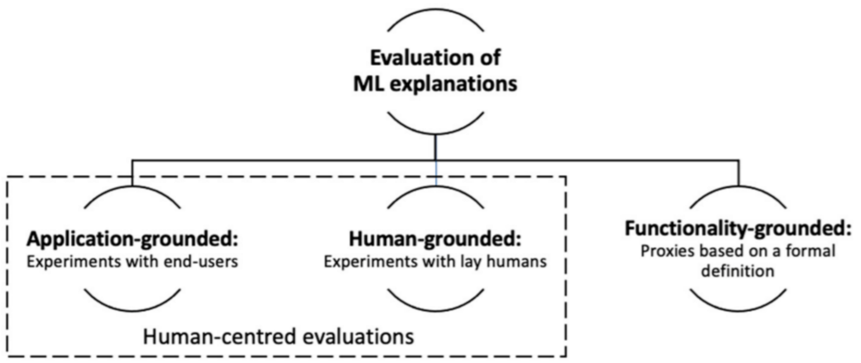
Evaluation of ML explanations
Application-grounded: Experiments with end-users
Human-grounded: Experiments with lay humans
Functionality-grounded: Proxies based on a formal definition
Human-centred evaluations

Figure 4.1: groups of explanations [**13**]

As the task explored is Handwritten text generation (a task well understood by humans), I chose to focus on the human-grounded methods - methods that are measured by their quality and their human interpertablity, in contrast to methods measured by the scores of a specific matrix.

Chapter 5

Design and Implementation

5.1 Hnadwriting Generation

5.1.1 ScrabbleGAN

I decided to implement the ScrabbleGAN [**ScrabbleGAN**] architecture, which is able to generate handwriting text from script with various lengths and styles. The fundamental choice was either to work with online or offline data. I chose to focus on the latter.

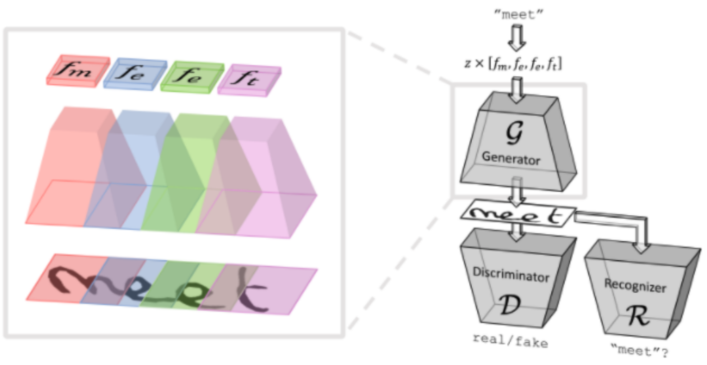

Figure 5.1: ScrabbleGAN structure

There are a few modifications from the original GAN structure, that make this model a suitable solution to generate handwriting in different styles and lengths.

1. Add a recognizer component - that encourage the network to generate meaningful and readable texts.

2. Local embedding of each character with overlapping. The final image is generated by convolutions that allow overlapping between the characters initial embeddings, so the model learns how to generate transitions between characters.

The model contains around 50 million parameters, including attention blocks, residual blocks and conditional batch normalization layers, as purposed in the original paper.

5.2 Explainability

5.2.1 Extract latent space structure from a trained GAN

An agnostic method to discover latent space linear directions is presented in the paper - in Unsupervised Discovery of Interpretable Directions in the GAN Latent Space paper - [**10**].

Main idea

The idea is to build a model on top of the generator, that will approximate meaningful directions (linear vectors). It contains two trained sub-models - A and R. The A sub-model produces the meaningful linear directions in latent space. From these directions, we produce shifted noise vectors. The generator generates images from the original noise vectors and from the shifted ones. The R sub-model reconstructs the directions from the two generated images.

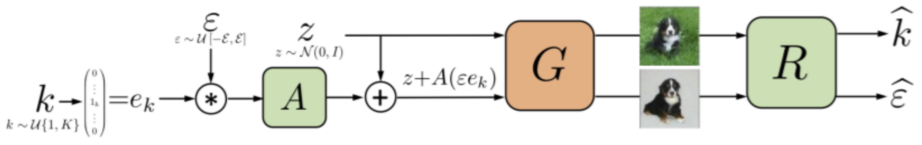
ε
$\varepsilon \sim \mathcal{U}[-\mathcal{E},\mathcal{E}]$
z
$z \sim \mathcal{N}(0,I)$
k
$k \sim \mathcal{U}\{1,K\}$
$=e_k$
A
$z+A(\varepsilon e_k)$
G
R
$\widehat{k}$
$\widehat{\varepsilon}$

Figure 5.2: unsupervised discovery of latent space direction scheme as proposed in the original paper

Algorithmic Flow

The columns of A are the shifts in the latent space, so by choosing two variables - column k and size ϵ, we define a transformation from vectors in latent space to another vector in latent space, shifted by ϵ times the $k'th$ column of A.

1. Generate random k, ϵ
2. Generate z, $z_{(k,\epsilon)}$
3. Generator generate two images, G_z, $G_{z(k,\epsilon)}$
4. R tries to recover k, ϵ from the two generated images.

A tries to find directions such that R will be able to reconstruct, based on the different characteristics of the two images. I've modified the original paper structure of R, for both MNIST GAN and ScrabbleGAN:

5.2.2 Attention Mechanism

In GAN models

In GAN models, it was introduced in SAGAN paper [**SAGAN**] the implementation of this idea as layers of the discriminator and generator:

Self-Attention Generative Adversarial Networks is the implementation of attention mechanism in GAN architecture.

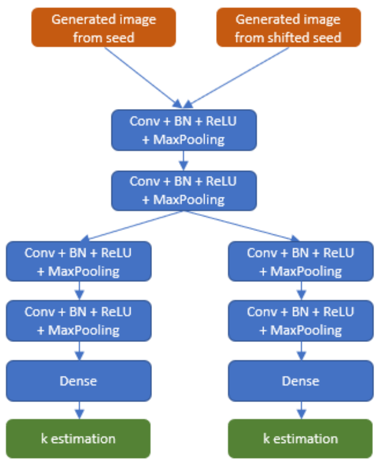
Generated image from seed
Generated image from shifted seed
Conv + BN + ReLU + MaxPooling
Conv + BN + ReLU + MaxPooling
Conv + BN + ReLU + MaxPooling
Conv + BN + ReLU + MaxPooling
Conv + BN + ReLU + MaxPooling
Conv + BN + ReLU + MaxPooling
Dense
Dense
k estimation
k estimation

Figure 5.3: Modified R network for MNIST

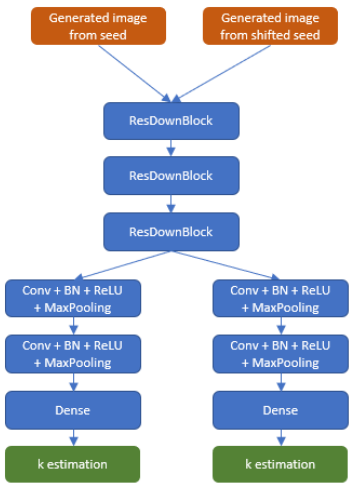
Generated image from seed
Generated image from shifted seed
ResDownBlock
ResDownBlock
ResDownBlock
Conv + BN + ReLU + MaxPooling
Conv + BN + ReLU + MaxPooling
Conv + BN + ReLU + MaxPooling
Conv + BN + ReLU + MaxPooling
Dense
Dense
k estimation
k estimation

Figure 5.4: Modified R network for SrabbleGAN

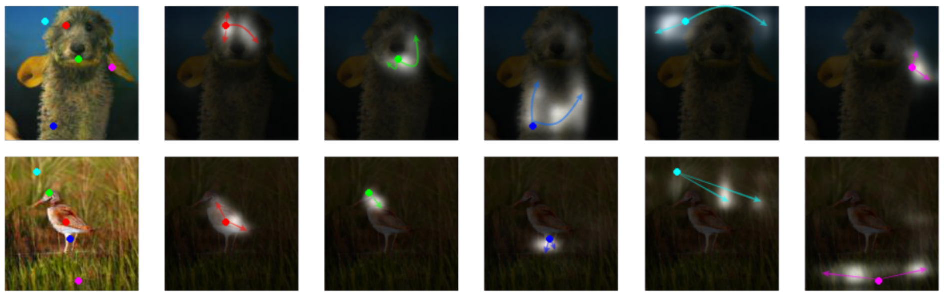

Figure 5.5: Self Attention maps as presented in SAGAN paper

Algorithmic Flow

The implementation is pretty straight forward from the the plot at Fig 5.6.

All three parts - K, Q and V are derived by separate convolutions of the inputs.

The keys and queries create (by matrix multiplication and soft-max activation) the attention maps. For each generated output of the model, there is a correlated attention map at the size of the layer input. It is activated where the model should put attention to, and zeros areas that are needed to be ignored.

This map is multiplied by the Values and creates a generated output. Each part of the output (pixels in our case) has access to information from all across the input thanks to the attention mechanism. This is why it sometimes called Non Local Blocks. With traditional architectures, each pixel won't have this various access to all parts of the input, but only to the local features in its area.

5.2.3 InfoGAN

InfoGAN is an extension of GAN, that by modifying both the latent space and the objective goal is able to learn distangled representations.

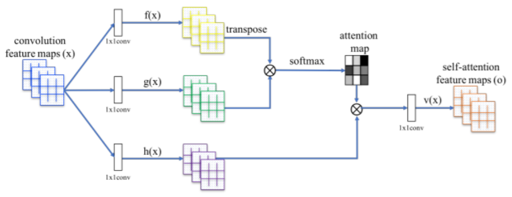

Figure 5.6: Self Attention layer as proposed in SAGAN paper

Main idea

InfoGAN adds a categorical feature c and countinous features c_i to the latent space. It tries to recover only them and not the whole latent space. One can think of this framework as a relaxed version of the BiGAN - we are enforcing some restrictions on the generator, forcing him to preserve the latent space structure.

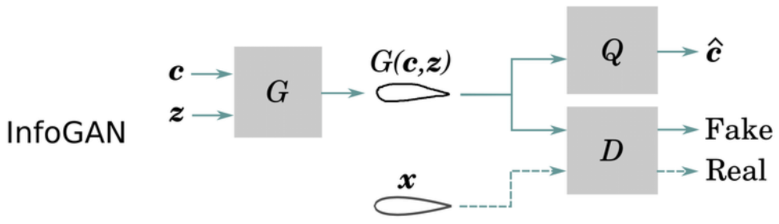

Figure 5.7: InfoGAN structure [14]

Implementation details

We draw the codes from a known distribution, and the encoder Q tries to recover the distribution they were drawn from.

the new loss term is:

$\underset{G}{min}\underset{D}{max}V(D,G) - \lambda I(c, G(z,c))$ where V(G, D) is the vanilla GAN loss term, and I is the information between the codes and the image generated using them as part of the seed.

The loss term is calculated by a probability function:

$$L_I(c, c_i, \varphi_c, \varphi_{c_i}) = -log(P(c|U_{\varphi_c})) - log(P(c_i|N_{\varphi_{c_i}}))$$

φ_c are the parameters to fix a discrete distribution (softmax activations), and φ_{c_i} are the parameters to fix a normal distribution (mean and variance), and these are the Q network outputs. We measure I by Q network, that predicts the distribution of these additional variable.

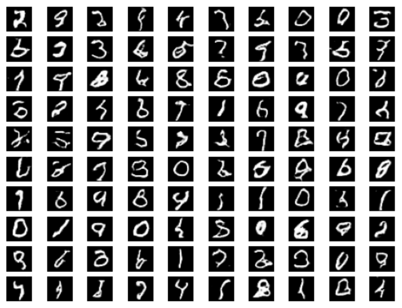

Figure 6.1: Generated MNIST data by GAN

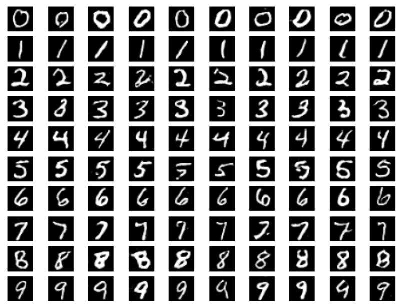

Figure 6.2: Generated MNIST data by CGAN

Chapter 6

Experiments and Results

6.1 MNIST results

6.1.1 GAN vs CGAN

It is important to notice that CGAN architecture works much better than the original GAN architecture on MNIST (Fig 6.1 and 6.2)

This phenomena can be understood just by thinking on the process of handwriting:

Conditioning the 'main attributes' of the generated handwriting should be based on the text. This how humans write. The content and the style are separated. Giving the model this ability produces much better results, and it also can be though as a tool to extract meaningful insights (though not implemented in this project).

Disentanglement of style and content

The CGAN model consists an additional layer to the generator structure - an embedding layer from the labels space to the pixels space. The embedded layer learns the "content", and the latent space variations control the different style characteristics of the digit.

We can find a confirmation to that claim be looking at figure 6.3 - the same noise seed generates all digits on the same row, and all of these samples seem to share the same style (width and orientation are the most obvious ones from the 1st and 9th rows).

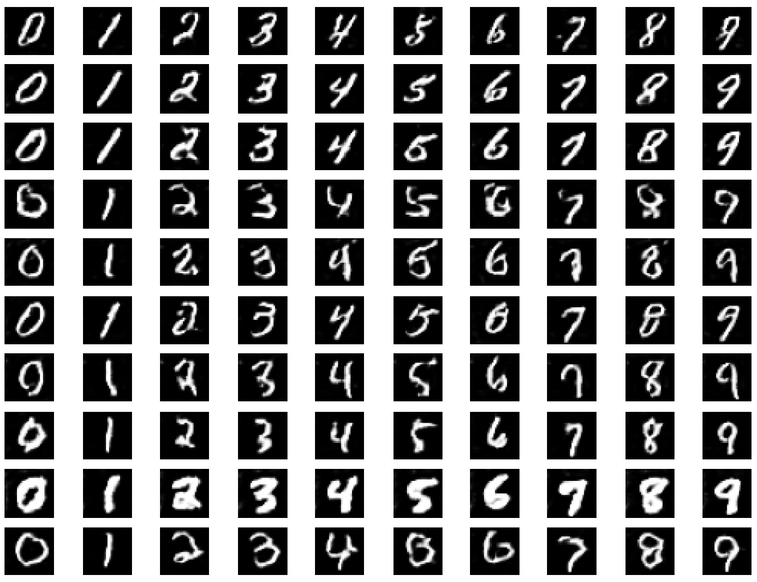

Figure 6.3: Generated MNIST data by CGAN, each row is generated from the same seed in latent space

Notice that the same qualitative insight can be inferred from Graves work (Generating Sequences With RNNs). The conditioned samples (those who are generated from text, and not just by sampling from the model) look much more realistic, beside the fact that they actually form meaningful words. Note that there seems to be real letters in the unconditional sample, but they are not accurate as in the conditional one.

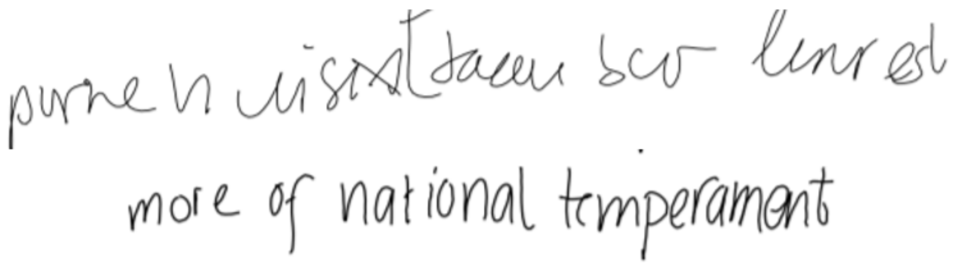

Figure 6.4: unconditional (first row) vs conditional sampling (second row) in Graves work on IAM online Data

Directions in latent space

By exploring the directions in the latent space, we can visualize the changes in style versus the changes in the latent space (linearly). In the vanilla GAN, the changes in the latent space seems to modify the shapes of the output digits, and not only what we refer to as their 'styles'.

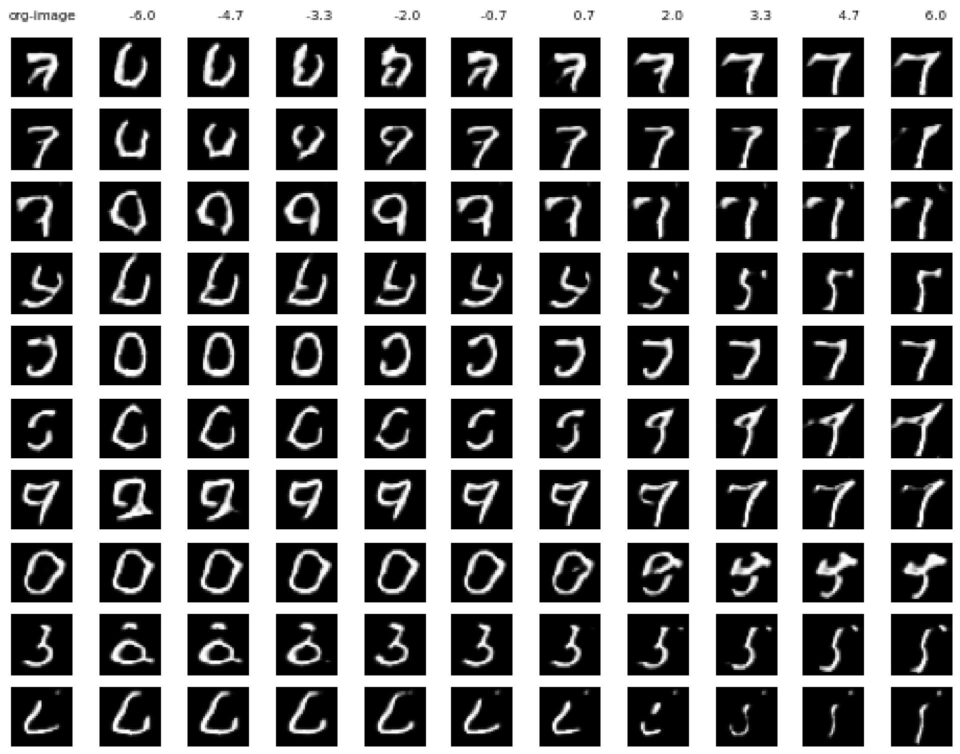

Figure 6.5: Discovered direction of GAN on MNIST

The latent space of CGAN seems to be much more 'organized' in the sense that it controls only the style of the digits and not their core characteristics. More over, the concepts of width and orientation, and even the size of the digit is reflected in the CGAN latent space.

6.1.2 Attention

Note that generally, the attention maps have the same general shape as the image generated. That means, that each pixel 'pays attention' to the whole generated image, and is activated based on that, and not just based on the features from the previous layer. Another important insight so notice is that when the pixel we queried is silent, the maps seems to tend toward zero, while when the pixel is drawn, the maps reveal the main shape of the digit we draw.

6.1.3 InfoGAN

Disentanglement representation

Note that the InfoGAN generate different digits just by adjusting the input categorical code, but without any other enforcement as in CGAN (the embedding layer).

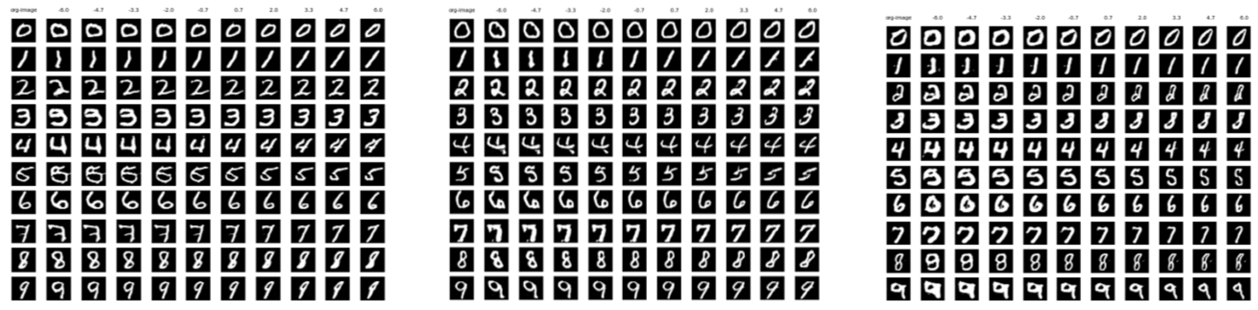

Figure 6.6: Discovered direction of CGAN on MNIST
each bloch is the same direction, each row is a different seed generating all labels. Note how the latent space exclusively on the style - we can see changes in orientations and width.

The model is not aware to the different classes in the data, but still tend to do the 'right' separation in latent space, just by minimizing the information from that latent code to the images the generator produces. in the original paper it is mentioned that this classification by the categorical codes gets up to 95% accuracy. The continuous codes seem to have less effect in my implementation, though that in the paper the effect of both codes is clear and interpretable.

6.2 ScrabbleGAN results

6.2.1 Directions in latent space

Though the algorithms was able to find direction that are meaningful to humans (Fig 6.9 and 6.10), it's still not as expressive as the one found in the MNIST GAN. That might relate to both the higher complexity of the problem (the change in R architecture might not be as optimal as possible), and also to the quality of the ScrabbleGAN generations.

6.2.2 Attention

Though we can find the same qualitative charactaristics in the attention maps of ScrabbleGAN output, they are much less informative and interpretable than in the MNIST case. Note that as in MNIST data, queries where there is no writing seems to be silent (zero), and the one with writing are more activated. Note the pattern that emerge in the map - it seems to be split by the number of characters, which obviously relates to the generation process. I would expect the maps to be more relevant, as in MNIST - to see shapes of the characters.

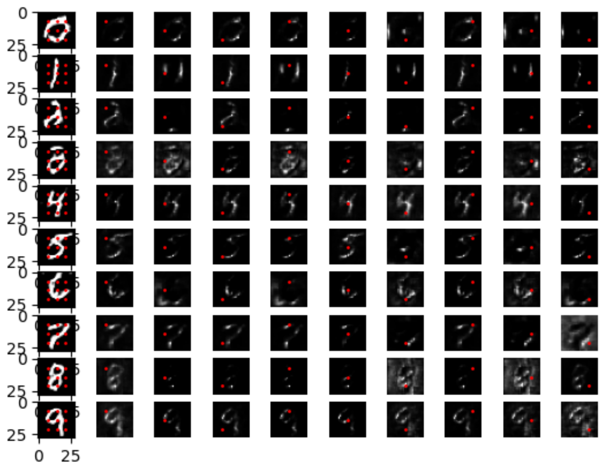

Figure 6.7: attention maps of the generator model
Leftmost image - the output image, with the location queries of the attention.
each one of the images to the right is the attention map for that specific query - what areas of the previous layer the model pay attention to.

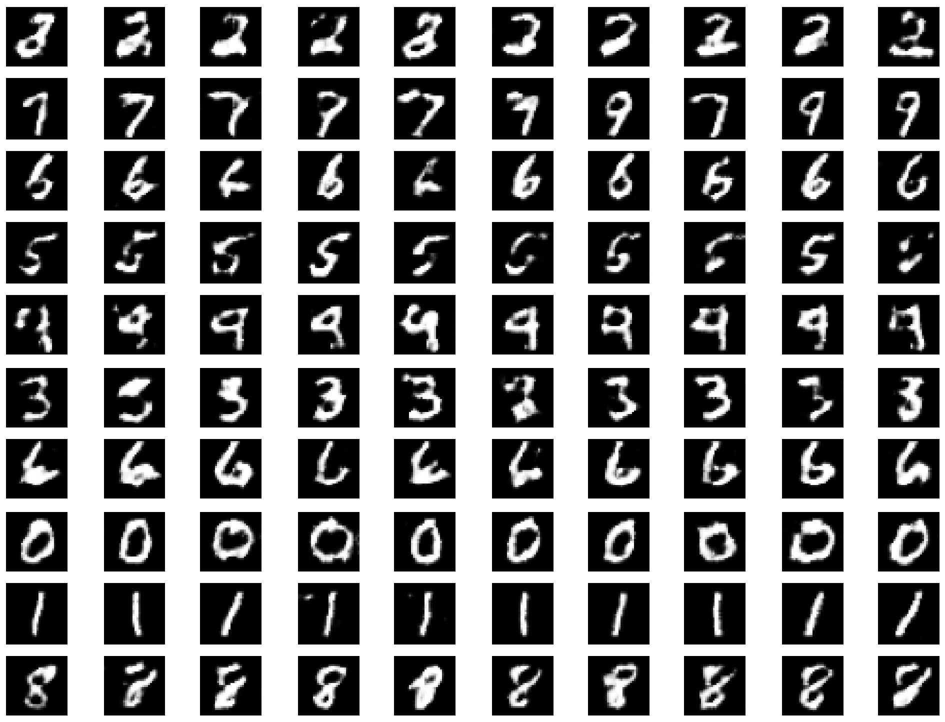

Figure 6.8: Random sample from InfoGAN

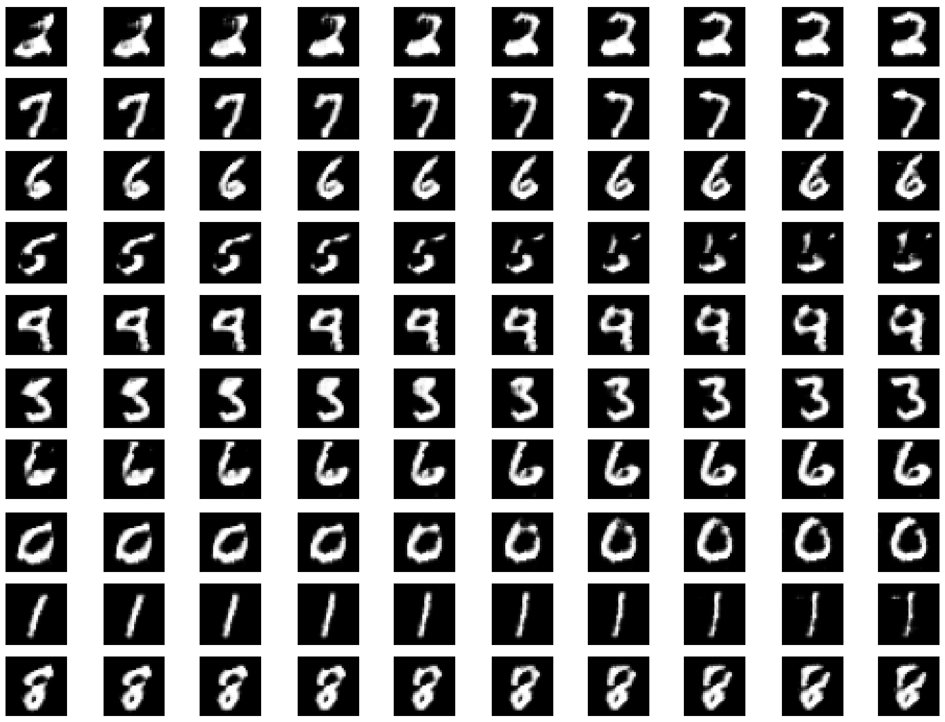

Figure 6.9: Same noise seed with different codes. Note how the codes define the digit, though the model wasn't train with any labels

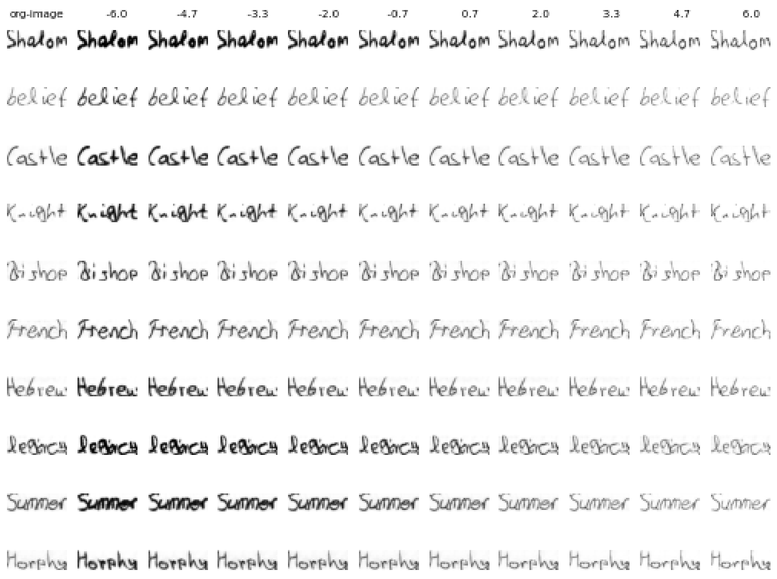

Figure 6.10: Detected direction in ScrabbleGAN that effect the width of the writing

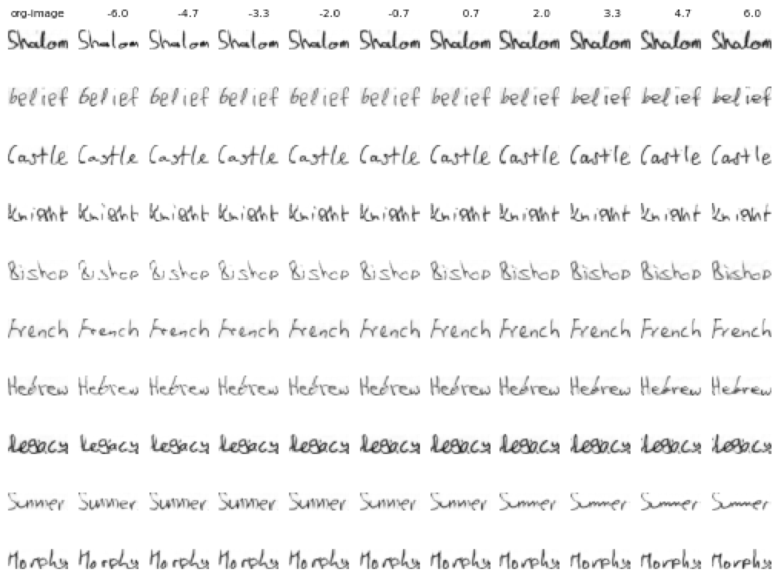

Figure 6.11: Detected direction in ScrabbleGAN that effect the cursivity of the writing

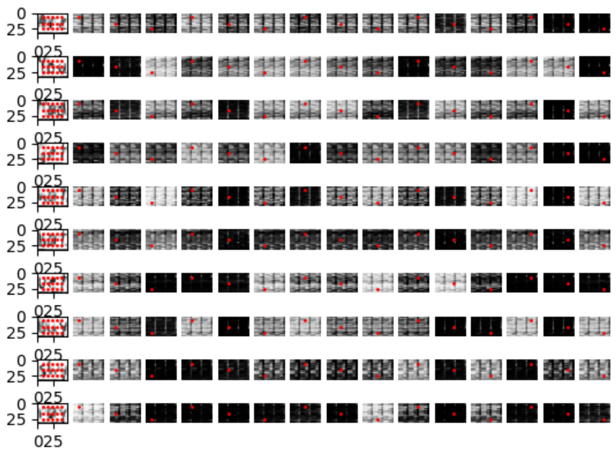

Figure 6.12: ScrabbleGAN attention maps

Chapter 7

Discussion

I will split the discussion of the results to two parts - the qualitative results of the explainability methods (as explored on the MNIST base data set), and the results on the ScrabbleGAN use-case.

7.0.1 The benefits of explainability

On the MNIST dataset, the methods seems to perform well and extracted some meaningful results.

Both InfoGAN and the direction discovery algorithms seems to produce meaningful latent space explorations. It reveals the power of the generative model on this simple dataset - the range of different characteristics is revealed by these explorations.

The attention maps also seems to be useful - it shows that the model captured the basics features of the output space, by showing that each pixel is 'aware' to all of the feature space and not just to its own area.

7.0.2 What is still missing

The implementation on ScrabbleGAN was partially successful - The latent space explorations seems to generate some interesting directions.

The results of the latent exploration could be improved with a better generative models - with more variation of style. ScrabbleGAN has a training trick involves gradient balancing that is supposed to increase the style variance, but I left this part out of the scope of this project. It will be interesting to see the results this exploration with a stronger generative model.

The attention maps doest seems to work on ScrabbleGAN - the maps doesn't look like anything that resemble any realistic process of writing. Beside looking for bugs in the implementation (the model seems to perform well), maybe some alternative visualizations is needed. This reminds the optimization tasks of kernel visualization in classical CNN, as in [**15**].

7.1 Future Work

Unfortunatly, I wasn't able to complete the implementation InfoGAN combined with ScrabbleGAN or CGAN. This is the obvious thing to explore - what kind of "classes" will emerge from the model, while we generate conditional samples (we condition the content, so I would expect kind of raw 'style' groups)

Another interesting continuation of this work is to Implement this methods on online writing methods. What adjustment in the architectures of the methods should be considered to make the methods feasible to generate meaningful insights?

Appendix A

Personal Reflection

A.1 Reflection on Project

As my first big machine learning project, I definitely underestimated the amount of work that was needed for the project. A note for the next time - got an interesting paper that looks relatively simple? It might take more than just a couple of hours to make it work. I guess that has a lot to do with my lack of experience of implementing large models with many parameters and configurations.

The computing power that was needed here is substantial, though my machine did a great job for the most part, and deploying the models to external machines is another skill I got thank to that project.

A.2 Personal Reflection

This project was a wild ride for me. From moments of total desperation to moments of sending images to my family to see how cool what I am doing - I had it all. I found this topic so interesting, that I just couldn't face the idea that I will submit a project without any new insights. Even if its not going to change the xAI world, at least its something worth reading, and not just a completely recycled versions of others papers. I was stuck for weeks over the implementation of ScrabbleGAN that will give me the flexibility to explore the topics I wanted (the list was much much longer...).

For me, the key lesson from this project was time management and estimation - I had to bound myself to sensible goals within the given scope (and general situation, in personal life and other outer components), and I feel that I didn't managed this that way at all.

Appendix B

Appendices

B.1 Some implementation notes

All of the implementation can be found here:

$https : //github.com/RoeiBrudo/Explainability_GAN$

the data could be found here:

$https : //drive.google.com/file/d/1rBQXD5mZv4ed1 - -XsmMig36oh0 - 70Kp2/view?usp = sharing$

All that is needed is to extract the data folder to the explainabilityMain folder, and to run *main.py* script.

The script arguments:

function	GAN or CGAN or InfoGAN
data	MNIST or IAM
action	train or infer
explore latent	flag
explore attention	flag

The only exception is the discovery of latent space direction on ScrabbleGAN, which gave the best results when used in an off the shelve solution by Nikolai10:

$https : //github.com/Nikolai10/scrabble - gan$

$https : //towardsdatascience.com/scrabblegan - adversarial - generation - of - handwritten - text - images - 628f8edcfeed$

The solution for that algorithm is ScrabbleDirections folder. all is needed it to train the model first *main.py* and then run the $run_explorer.py$ script.