BertonGan: a conditional GAN for performing various tasks

Aaron Schindler, Herbert Wright

December 16, 2022

1 Introduction

1.1 Problem Statement

Given data $\mathcal{D} = (X, y)$ drawn from some unknown distribution such that $X \in \mathbb{R}^{n_x \times d}$, and $y_i \in y$ is an "identity" or "style class" of example x_i, we wish to define a function $\hat{f}$ such that we can take in a "content" image and a "style" image and generate an image that is close to the "content" in distance, but shares the same "style class" as the style image.

This is called style transfer and one common application is face swapping. In this instance, our style classes are people's identities, and we are trying to, given one image, project another image's person's face onto it.

1.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) were introduced by Ian Goodfellow [1], and are a common way to perform style transfer. One such example is StyleGAN [2]. An example of a GAN being used for face swapping specifically can be found in [3]. Unfortunately, these GANs tend to require a lot of data and computation to train.

2 Methods

2.1 BertonGan structure / Netorks

Our Approach is two learn a metric of face/style classes via an encoder network, then learn to encode image content with a seperate encoder, and then a decoder network which operates on both inputs. We also define two different discriminator outputs; the first determines if the image is fake, the second is conditioned on a latent vector from our face/style classes encoded space and determines if the image belongs to that style class. In short we have 5 different networks:

1. Face encoder network: $f_F : \mathbb{R}^{n \times W \times H} \to \mathbb{R}^{h_f}$
   (a) Input: n images of the same subjects face
   (b) Output: A latent representation of the subjects face
2. Image encoder network: $f_I : \mathbb{R}^{W \times H} \to \mathbb{R}^{h_I}$
   (a) Input: An image of a subjects face
   (b) Output: A latent representation of the image
3. Image decoder network: $f_G : \mathbb{R}^{h_F + h_I} \to \mathbb{R}^{W \times H}$
   (a) Input: Latent representations of a face and image
   (b) Output: A reconstructed image decoded from the latent features
4. 2 Discriminator networks: $f_D : \mathbb{R}^{W \times H} \times \mathbb{R}^{h_F} \to [0, 1]^2$

(a) Input: An image and a latent representation of a face

(b) Output: Two probabilities (each of these probabilities has it's own corresponding network in practice)

   i. Probability of being a fake image

   ii. Probability of being a different person than the faces encoded into the latent vector

We generate our images by encoding our style image (F_A) and our content image (I_A or I_B depending on its style class) with our two different encoders, then feeding the output into our image decoder network. Our discriminators use this as input. A visual of the flow of the network is shown in figure 1 (circles denote networks and boxes are tensor inputs or outputs):

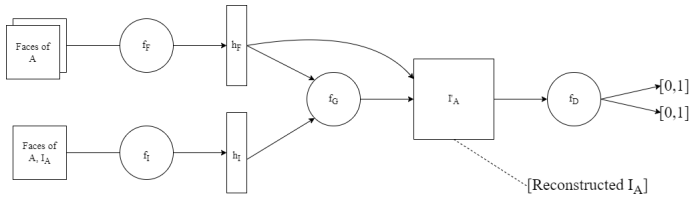

Figure 1: Figure 1 is a visual representation of our total network, comprising each of the four components above

2.2 Training Procedure

We propose using two datasets to train/test. The first is the built in MNIST dataset from the torchvision package [4]. This dataset will allow us to experiment with our networks on a smaller scale, as the MNIST images are 28×28 images of numbers rather than people. The next step is to use the celebA dataset which is also built in to the torchvision package. Our original plan to use the MS-Celeb-1M dataset [5] has been changed because the dataset is no longer publicly available. The celebA dataset comprises over $200,000$ images of faces, that are 178×218 in size.

Given a batch $\beta = (F_A, I_A, I_B)$, where $F_A = n$ faces of person A, $I_A = N$ other faces of person A, and $I_B = N$ faces of people other than person A, we will compute the following quantities:

1. $h_F = f_F(F_A) \in \mathbb{R}^{h_F}$
2. $h_I = f_I(F_B) \in \mathbb{R}^{N \times h_I}$
3. $h_B = f_I(F_B) \in \mathbb{R}^{N \times h_I}$
4. $I'_A = f_G(h_F, h_I) \in \mathbb{R}^{N \times W \times H}$
5. $I'_B = f_G(h_F, h_B) \in \mathbb{R}^{N \times W \times H}$

6. $(R_{A'}, C_{A'}) = f_D(I'_A, h_f) \in [0,1]^{N\times 2}$
7. $(R_A, C_A) = f_D(I_A, h_F) \in [0,1]^{N\times 2}$
8. $(R_{B'}, C_{B'}) = f_D(I'_B, h_F) \in [0,1]^{N\times 2}$
9. $(R_B, C_B) = f_D(I_B, h_f) \in [0,1]^{N\times 2}$
10. $D_A = \|I_A - I'_A\|$
11. $D_B = \|I_B - I'_B\|$

We will optimize f_D by maximizing R_A, R_B, C_A and minimizing $R_{A'}, R_{B'}, C_B$. We also optimize f_F by maximizing $C_A, C_{A'}, C_{B'}$ and minimizing C_B, D_A. Additionally, we optimize f_G, f_I by maximizing $R_{A'}, R_{B'}, C_{A'}, C_{B'}$ and minimizing D_A, D_B. For stability purposes, we use a least-squares loss as proposed for GANs in [6] All parameters are optimized by using stochastic gradient descent.

After training, we will be able to use $f_F, f_I,$ and f_G to perform face swaps or style transfer. We will then use f_D to identify fake/real images generated using face encoding h_F. Additionally, we will use f_F and f_G to generate new images of an already learned face.

3 Experiments

Code for all experiments can be found in our GitHub repo here:
https://github.com/Herb-Wright/berton-gan

3.1 MNIST Experiments

We first trained on the MNIST dataset with defined networks that were not too far from the GANs we used in the notebook in class. In the first BertonGan we trained ended up with saturated gradients in the discriminator1 network (the one that predicts whether or not the image is fake). After this, the generator was able to easily fool this, leading to fuzzy output images. We trained this network for 50 epochs. We show two different graphics in figures 2 and 3; the first is performing our version of style transfer where the number of the image is the style class it belongs to, and just some generated images from picking random content and style images. All of the style and content images are from the test set and thus not seen before by the network.

Note that in figure 2, columns 1 and 4 are content, 2 and 5 are style (digit number), and 3 and 6 are the generated image. The goal is for the third column's image to be "close" to the first column, with the same number as the second.

The gradients were being saturated because we had a sigmoid in the last layer of the network to output a probability. We removed this activation, added two more layers to the network and retrained. We used a slightly different procedure this time around; we first trained the image encoder and decoder as an autoencoder for 5 epochs, then trained the whole network. Figures 5, 6, 7

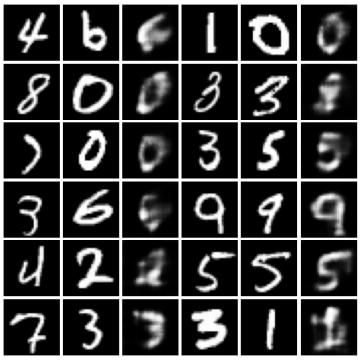

Figure 2: (a)
Style transfer

Figure 3: (b)
Generated images

and 8 show this network at various epochs. Figures 9 and 10 show, after 50 epochs, our version of style transfer and a sample of generated images similar to figures 2 and 3. Each image used for content and style is from the test set as before

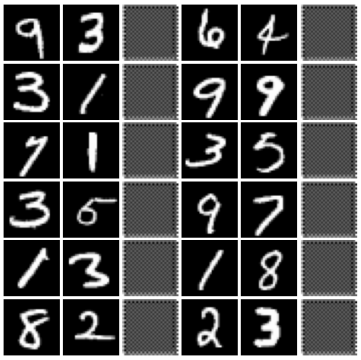

Figure 4: (a)
1st epoch (random noise)

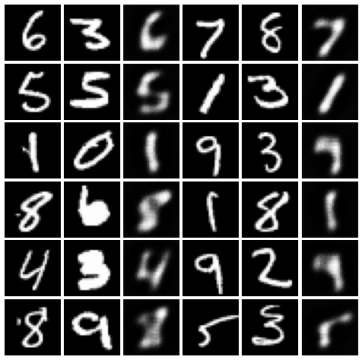

Figure 5: (b)
5th epoch (autoencoder)

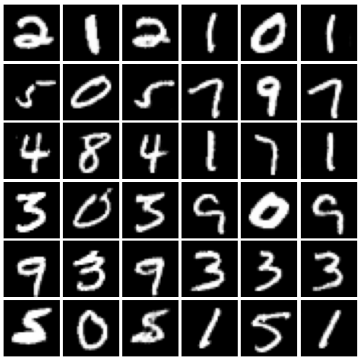

Figure 6: (a)
15th epoch (no style transfer)

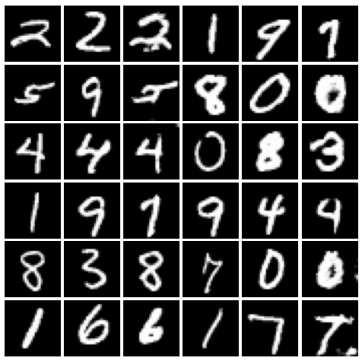

Figure 7: (b)
30th epoch

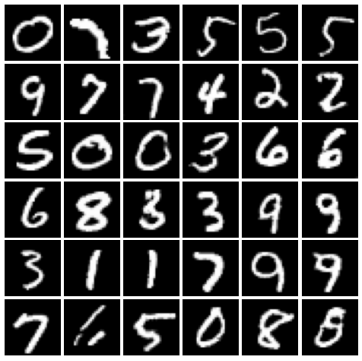

Figure 8: (a)
Style transfer (50th epoch)

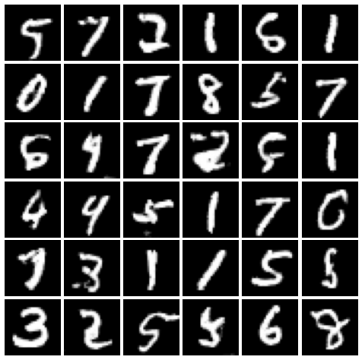

Figure 9: (b)
36 generated images

We can evaluate our face/style metric space by performing classification on the MNIST test data in the following way: first run through the training dataset and use the face encoder network to encode each digit, then average the latent vectors for each style class (digits 0, 1, ..., 9). Then, for each image in the test dataset, find the class whose averaged latent vector acheives the highest score from the second discriminator network (the one that predicts if the image and latent vector belong to the same class). Performing this procedure on the MNIST test images achieves 97% accuracy, suggesting an informative metric was learned.

Because we use latent dimension of 2 for our face/style encodings (encoding of the digit shown), we can display a grid of how this value changes as you move throughout the latent space for a given image. In figure 10, we encode the same image with the image encoder network then vary the latent face/style encoding vector. The output is the similar images but with the number changing as you move in the latent space.

Figure 10: visualization of latent space

3.2 CelebA Experiments

We attempted to train on the CelebA dataset, but it took too long and Colab kicked us out before we even got to epoch 4. Because of this the only model we have on the dataset is not very good. Our approach was to resize the images to 128 by 128 and then use deeper networks as part of the BertonGan. Figures

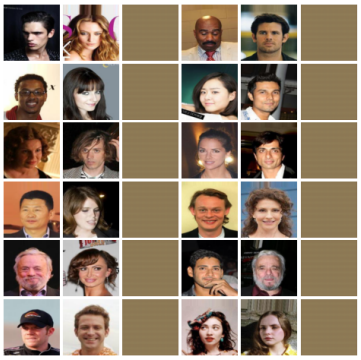

Figure 11: (a)
Style transfer

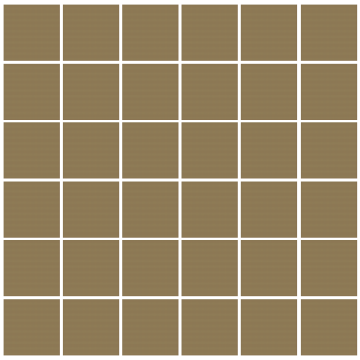

Figure 12: (b)
Generated images

4 Conclusion

In conclusion, it is very clear that when generating deepfakes it is possible to perform style transfer via metric learning using GANs. One downside to performing this operation was the amount of time and compute resources it takes to train the network. It takes roughly 10 minutes to train using cloud computing resources such as google colab for just the MNIST dataset. It took over 2 minutes for one epoch when trained on the CPU. Because of this, the celebA dataset presented an entirely new challenge since the dataset size is much larger than MNIST. When we tried to train on the CelebA dataset, it took roughly 27 minutes to complete one epoch. It would be nice to see if there was some way to alter the structure of BertonGan or training procedure so that it didn't take quite as long. Despite this, our MNIST experiments clearly show potential in style transfer via learning a metric.

Future Work. In another direction, diffusion models have grown in popularity for image generation [7]. It would be interesting to see if ideas from BertonGan could be translated into the diffusion model setting, where you still learn metric and condition your diffusion model on the metric space.

References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[2] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4401–4410, 2019.

[3] B.-S. Lin, D.-W. Hsu, C.-H. Shen, and H.-F. Hsiao, "Using fully connected and convolutional net for gan-based face swapping," in *2020 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pp. 185–188, IEEE, 2020.

[4] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE signal processing magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[5] Y. Guo, L. Zhang, Y. Hu, X. He, and J. Gao, "Ms-celeb-1m: A dataset and benchmark for large-scale face recognition," in *European conference on computer vision*, pp. 87–102, Springer, 2016.

[6] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley, "Least squares generative adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 2794–2802, 2017.

[7] F.-A. Croitoru, V. Hondru, R. T. Ionescu, and M. Shah, "Diffusion models in vision: A survey," *arXiv preprint arXiv:2209.04747*, 2022.