



3A ENSAE
BAYESIAN STATISTICAL PROJECT

Article Summary : Bayesian GAN (NeurIPS¹ 2017)

Supervised by Anna SIMONI

Yao Pacome KOUAME, Lasme Ephrem Dominique ESSOH

15 janvier 2023

1. The Conference and Workshop on Neural Information Processing Systems (abbreviated as NeurIPS and formerly NIPS) is a machine learning and computational neuroscience conference held every December.

1 Introduction

The paper Bayesian GAN was published by two authors, Y. Saatchi and A.G. Wilson, at NeurIPS 2017. This paper introduces a new formulation for GANs (Generative Adversarial Networks) that applies Bayesian techniques, and outperforms some of the current state-of-the-art approaches. GAN (Goodfellow and al., 2014) is a generative model based on neural networks for estimating high-dimensional data distribution. Powered by Deep Neural Network (DNN) learning capabilities, GANs have been very useful to generate natural signals, such as images, videos and audios. However, GANs face some problems, in some cases their learning objective can lead to a mode collapse which is the fact that GAN's generator learns or memorizes single mode while data distribution is multimodal. Many authors have proposed solutions for helping alleviate these practical difficulties, including the Wasserstein GAN method (Arjovsky and al., 2017) and the variational divergence minimization (Nowozin and al., 2016). But all these solutions are complicated in practice due to difficulty to decide which divergence we wish to use for GAN training. To address mode collapse, the authors Y. Saatchi and A.G. Wilson propose a different approach based on Bayesian inference to solve the problems related to GANs : Bayesian GAN. This report concerns scientific aspects of Bayesian GAN paper (Y. Saatchi and A.G. Wilson, 2017). It is organized as follows : section 2 presents a well-understood summary of the model developed in the main paper we studied. Section 3 presents the results of our application of the Bayesian GANs and section 4 provides a conclusion of our work and how it could be continued.

2 Article summary of Bayesian GAN's paper

In this first section we present the abstract of the Bayesian GAN article. But before this presentation, we quickly recall the principle of classical GAN introduced by Ian Goodfellow and al. in 2014.

2.1 Generative adversarial networks (GAN)

Let's present the classical case of GAN. Generative Adversarial Network [Goodfellow et al. 2014] comprises of two models : a generative model G and a discriminative model D .

- **Generative model** : the generator G has the role to create new data from the input data distribution, therefore implicitly trying to approximate the true data distribution as best as possible ;
- **Discriminative model** : the discriminator D has the role to distinguish between true data in training set and generated data by G .

Therefore, generator G is trying to fool discriminator D by producing suitable samples, while D is trying to become better at identifying the artificially generated data.

How GAN is it trained ? Let G and D , the generator and discriminator models respectively. We suppose that two models are parametrized by vectors θ_G and θ_D respectively². We suppose also that we have data x coming from an original distribution $P_{data}(x)$. The goal of GAN is to approximate this latest distribution. Finally, we consider that "fake" data generated by G provide from an input noise z with $P_{noise}(z)$ distribution. In the original paper by Ian Goodfellow et al., the problem of GAN is defined as zero sum game based on binary classification for two models G and D . According to Goodfellow and al. if we consider the binary cross-entropy loss formula :

$$\mathcal{L}(\hat{y}, y) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}), \text{ where } y : \text{original data, } \hat{y} : \text{reconstructed data,}$$

the objective of the discriminator is maximize the loss function :

$$\mathcal{L}(D) = \max_D [\log(D(x)) + \log(1 - D(G(z)))] \quad (1)$$

and the generator is competing against discriminator. So, it will try to minimize the loss :

$$\mathcal{L}(G) = \min_G [\log(D(x)) + \log(1 - D(G(z)))] \quad (2)$$

where $D(x)$ and $1 - D(G(z))$ represent, respectively, the probability that the discriminator D categorizes true input x and generated or "faked" input $G(z)$ correctly.

We can combine (1) and (2) to obtain :

$$\mathbf{L} = \min_G \max_D [\log(D(x)) + \log(1 - D(G(z)))] \quad (3)$$

Remember that the above loss function is valid only for a single data point, to consider entire dataset we need to take the expectation of the above equation as

$$\min_G \max_D V(D, G) = \min_G \max_D (E_{x \sim P_{data}(x)} [\log(D(x))] + E_{z \sim P_z(z)} [\log(1 - D(G(z)))]) \quad (4)$$

This is the (4) loss which is used to train GAN by Empirical Risk Minimization principle.

What's possibly problems in GAN training? Training a GAN can be tricky and the process unstable, requiring manual intervention and careful parameter tuning. The most common problems is mode-collapse where the generator G finds some convenient way to fool D , and overly exploit this and "overfit" to a specific part of the distribution by transforming most random inputs z into very similar outputs, coming from the same part of the underlying data distribution. We will then have a generator that does not correctly approximate the distribution of the data, but has instead "collapsed" to one of its modes. One of the greatest contribution of the paper we will present here, Bayesian GAN is a fully Bayesian approach to solve this issue.

2. Generally in case of GAN, G and D are neural networks parametrized by vectors θ_G and θ_D

2.2 Bayesian GAN

What is it and why is it inovation? Bayesian GAN (BGAN) is a generalization of the traditional GAN based on bayesian analysis. The idea is to approximate the posterior distribution one the parameters of the generator θ_G and discriminator θ_D and use the full distribution to generate data instead of a pointwise estimation. This allows to generate more diverse samples, avoid mode collapse and better represent the underlying distribution. In a typical GAN, we learn one θ_G and one θ_D . Here, the goal is to learn a distribution over each of these parameters. By doing so, we are able to recover more diverse samples from the original data distribution, and are able to train this system more reliably with lesser chances of mode collapse. In Bayesian GAN paper, authors achieve their results by marginalizing the posterior over the weights of the generator and discriminator using Stochastic Gradient Hamiltonian Monte Carlo (SGHMC). The samples from the posterior define multiple generators that correctly cover the data distribution.

Mathematical framework of BGAN. Based on classical GAN, Y. Saatchi and A.G. Wilson suggest to make random the parameters θ_G and θ_D of generator G and discriminator D with priors $P(\theta_G|\alpha_G)$ and $P(\theta_D|\alpha_D)$ respectively where α_G and α_D are hyperparameters. If we do this, we can now build a pipeline similar to a bayesian hierarchical model - we use these priors and the likelihood to estimate posteriors on the parameters. In this case, we would obtain generated data by first sampling from the posterior on θ_G , then sampling some noise from $P_{noise}(z)$, and finally using the noise and the sampled parameters to compute the output of the generator. Mathematically, to get a new set of samples $\bar{x}^{(j)}$, this is what we want to be able to do :

$$\begin{aligned}\theta_G &\sim P_{posterior}(\theta_G) \\ z^{(1)}, z^{(2)} \dots z^{(3)} &\sim P_{noise}(z) \quad (i.i.d.) \\ \bar{x}^{(j)} &= G(z^{(j)}, \theta_G)\end{aligned}$$

Note the equality in the last equation above. As we have sampled, from generator's distribution, a particular value of θ_G and $z^{(j)}$, we obtain a single value for $\bar{x}^{(j)}$.

A fondamental question is how to compute posteriors distributions? To adress this question it is important to remember that the generator and discriminator are intertwined - the generator's parameters are directly influenced by the discriminator's, and vice-versa. So to find the posterior for the generator, we need the posterior for the discriminator, which will allow us to sample θ_G to calculate our likelihood. The authors propose to iteratively sample parameters from the following conditional posteriors :

$$P(\theta_G|z, \theta_D) \propto \left(\prod_{i=1}^{n_G} D(G(z^{(i)}, \theta_G), \theta_D) \right) P(\theta_G|\alpha_G) \quad (5)$$

$$P(\theta_D|z, x, \theta_G) \propto \prod_{i=1}^{n_D} D(x^{(i)}, \theta_D) \times \prod_{i=1}^{n_G} \left(1 - D(G(z^{(i)}, \theta_G), \theta_D)\right) P(\theta_D|\alpha_D) \quad (6)$$

Intuition behind posteriors forms We can intuitively understand posteriors formulations starting from the generative process for data samples. Suppose we were to sample weights θ_G from the prior $P(\theta_G|\alpha_G)$, and then condition on this sample of the weights to form a particular generative neural network. We then sample white noise z from $P_{noise}(z)$, and transform this noise through the network $G(z, \theta_G)$ to generate candidate data samples. The discriminator, conditioned on its weights θ_D , outputs a probability that these candidate samples came from the data distribution. Eq. (5) says that if the discriminator outputs high probabilities, then the posterior $P(\theta_G|z, \theta_D)$ will increase in a neighbourhood of the sampled setting of θ_G (and hence decrease for other settings). For the posterior over the discriminator weights θ_D , the first two terms of Eq. (6) form a discriminative classification likelihood, labelling samples from the actual data versus the generator as belonging to separate classes. And the last term is the prior on θ_D .

BGAN generalizes GAN. Authors proved that their proposed probabilistic approach is a natural Bayesian generalization of the classical GAN : if one uses uniform priors for θ_G and θ_D , we can perform an iterative optimization instead of posterior sampling over Eq. (5) and (6) to re-found results of GAN original paper by Goodfellow et al. (2014).

GAN and BGAN for semi-supervised learning. So far, we have described the unsupervised learning use of GANs : given just data, learn to reproduce the distribution so that new examples can be generated. However, GANs can also be used for semi-supervised learning. Appendix 1 of this report describes what this is because authors used BGAN with a view to semi-supervised learning in their experiments part of article.

2.3 Computational aspect of BGAN (posteriors sampling)

In the Bayesian GAN, we wish to marginalize the posterior distributions over the generator and discriminator weights for data distribution learning. For this purpose, the authors use Stochastic Gradient Hamiltonian Monte Carlo (SGHMC) for posterior sampling. Appendix 2 of this report describes main algorithm (pseudo-codes) based on SGHMC for posterior sampling.

2.4 Paper experiments and results

The experiments proposed in the paper involve 5 datasets : MNIST, CIFAR-10, SVHN, CelebA and a synthetic one, to illustrate robustness over mode collapse. Authors compare BGAN with different variants of neural networks GAN (Deep Convolutional GAN —DCGAN— and others GAN). In all tests, as we mentioned before, Algorithm 1 in Appendix 2 is repeated 5000

times, collecting the weights every 1000 iterations. Their values for the internal parameters are : $J_d = 10, J_g = 1, M = 2$ and $n_g = n_d = 64$. Note that authors place a $\mathcal{N}(0, 10I)$ prior on the generator and discriminator weights, and use a 5-layer deconvolutional GAN as a generator.

Synthetic experiment. The synthetic experiment is arguably the most interesting, and the one that better illustrates the robustness to mode collapse of the BGAN. What they do here is generate a very simple dataset (shown in 2D after PCA) with a very clear multi-modal distribution. They then observe samples produced by the generators from the Maximum-likelihood GAN (ML-GAN) and the Bayesian GAN (BGAN) respectively. They show that with the ML-GAN, there is clear mode collapse : the generated samples all fall in a very particular area of the distribution. With the BGAN, the approximated distribution is much more similar to the original, which shows that the BGAN effectively worked around mode-collapse and produced diverse enough samples that represent the underlying process better.

MNIST experiment. The authors highlight their impressive performance on the semi supervised MNIST classification task. They achieve a very respectable accomplishment : with only 100 labeled training examples, they get to the same classification accuracy (99.3%) of a state of the art supervised method using all 50000 training examples. On the unsupervised side, they remark that there are some clear differences between the samples they produce and the samples created from a simple Deep Convolutional GAN. Basically, they are showing the expresiveness of their Bayesian formulation : because they are modeling the entire distribution of generator parameters, they can get much more varied MNIST generations simply by sampling new generators from that distribution. They show that the digits created by 6 different generators sampled from the posterior create plausible digits, but from different handwritings. While a DCGAN learns to generate faithful digits with the same style, the sampled generators create different styles³.

CIFAR, SVHN, CelebA experiments. These experiments are all very similar. In all cases, they are working with more complex images, and thus more complex underlying distributions. They compare their results on the semi-supervised case, and again, they beat all other methods of GAN on classification accuracy. The unsupervised experiment is not as easy to compare, but it is clear that the diversity of samples coming from the BGAN is superior than the other methods.

3. The results are, however, not as high-fidelity as in the DCGAN case. The authors explain that this is to be expected : because we are sampling from a distribution over generators, we are not necessarily getting the generator that maximizes the likelihood, which would intuitively give the most high-fidelity results. Instead, the idea here is not to generate photorealistic samples, but instead to generate diverse enough samples so that the full underlying distribution of the data is represented.

3 Our implementation in Python of BGAN

In this part we present our application of BGAN for data images generation. Indeed, we implement the authors results on the MNIST⁴ dataset. This implementation was done in Python. The name of the codes folder is **Bayesian-GANS**. Codes folder **is available on our Github Repositories** (see appendix 3 for the link). This folder contains the following files :

- **bgan.py** : which contains the main implementation of the Bayesian GAN with the definition of the architecture of the neural network, the learning procedure, the definition of hyperparameters and so on. This script also contains a basic implementation of the classical GAN ;
- **download_dataset.py** : which contains the script to automatically download the MNIST data used for training. ;
- **Opt.py** : which contains the implementation of Algorithm 1 i.e. Stochastic Gradient Hamiltonian Monte-Carlo Sampler that uses in BGAN process ;
- **requirements.txt** : which contains all the modules required to run this project on your personal computer ;
- **readme.md** : which contains a description of how to execute the project based on the requirement.txt and the all previous source codes.

Nota. Our code is based on an adaptation of the authors' main code available at the following link. Their code is very old and is written in Python 2 with Tensforflow 1.0 for GAN and BGAN training. We propose a newer version in Python 3 with Pytorch inspired from many open-source codes. It is important to keep in mind that the training takes a lot of time and that a computer with GPU-CUDA is required for time optimisation. Else, to make sure that the user can execute the code in a reasonable time on CPU, we have fixed the number of epochs to train the network to 30 while others say that it is from 350 epochs that the generation of images almost correct and not a noise. You can change this number in the script bgan.py at line 41. Overall, we obtain the same results on MNIST as the authors. See appendix 4.

4 Conclusion

In conclusion, this work allowed us to study the richness that Bayesian analysis can bring to Deep Learning. Our main takeaway from the BGAN paper is that the Bayesian GAN formulation is useful for generating samples from across an underlying distribution, avoiding the mode collapse that too often plagues GANs.

4. The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems.

Appendixs

Appendix 1 : semi-supervised learning. Semi-supervised learning can be described as follows : we have a small dataset of labeled points and a larger dataset of unlabelled points, and we want to build a classifier. Semi-supervised learning aims to make the most of that large, unlabelled dataset, even without the labels, to improve the classifier accuracy. This topic is an active research field, and is quite useful - unlabelled data can be found everywhere. The paper that we study presents a way to use the BGAN to do pretty good semi-supervised learning. It turns out that the BGAN is suited to this kind of problem, because of the strong coverage of the underlying data distribution. Multiple papers [Kingma & Welling, 2013 ; Odena, 2016] show that using generative models, and in particular GANs, to leverage large unlabelled datasets leads to more accurate classifiers. It is clear then that our BGAN should do well in this setting : we are saying that it approximates the underlying distribution better, so it should generate better examples that push the discriminator to become a stronger classifier.

Let's see that mathematically on task of images labelling : Consider a total of n images coming from K classes, such that labels are only known for a small subset. As we said, the goal is to leverage the unlabelled images and the small labelled subset simultaneously to make a good classifier. Any image can either belong to one of the K ground truth classes, or be an image from the generator. Let label 0 correspond to a fake generated image, and $1 - K$ correspond to the K labels on true data. The discriminator only needs to be tweaked such that now $D(x(i) = y(i), \theta_D)$ gives the probability that sample $x(i)$ belongs to class $y(i)$. For each sample, we sum over the probability predictions for each class to obtain a tweaked version of the posterior inference formulae for θ_D and θ_G . This is the only difference in the posterior inference. Here, by summing over probabilities for all possible labels, we are incorporating evidence from both labelled samples (for $y \in (1, \dots, K)$), and for unlabelled samples like in the above case (label 0). Just as before, the posteriors are marginalized over the noise vectors using Monte Carlo.

Appendix 2 : SGHMC algorithm. The main algorithm, implemented below, shows how to compute one iteration of sampling for the Bayesian GAN. The objective is to iteratively repeat these steps to sample the marginalized posterior distributions derived above. Once we run this algorithm on multiple iterations, we end up with an array of θ_g and θ_d samples from their respective posteriors. It is interesting to observe that we get new samples by iterating on previous ones, but we end up with both the old and the new samples on the final array (because we are appending every new iteration). The first samples will be, of course, not as representative as the later ones, so something similar to a burn-in period is needed. In the paper, the authors compute this algorithm for 5000 Iterations, and then take samples every 1000 iterations, which serves this purpose.

Algorithm 1 One iteration of sampling for the Bayesian GAN. α is the friction term for SGHMC, η is the learning rate. We assume that the stochastic gradient discretization noise term $\hat{\beta}$ is dominated by the main friction term (this assumption constrains us to use small step sizes). We take J_g and J_d simple MC samples for the generator and discriminator respectively, and M SGHMC samples for each simple MC sample. We rescale to accommodate minibatches as in Appendix A.1

- Represent posteriors with samples $\{\theta_g^{j,m}\}_{j=1,m=1}^{J_g,M}$ and $\{\theta_d^{j,m}\}_{j=1,m=1}^{J_d,M}$ from previous iteration
- for** number of MC iterations J_g **do**
- Sample J_g noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(J_g)}\}$ from noise prior $p(\mathbf{z})$. Each $\mathbf{z}^{(i)}$ has n_g samples.
- Update sample set representing $p(\theta_g|\theta_d)$ by running SGHMC updates for M iterations:

$$\theta_g^{j,m} \leftarrow \theta_g^{j,m} + \mathbf{v}; \mathbf{v} \leftarrow (1 - \alpha)\mathbf{v} + \eta \left(\sum_{i=1}^{J_g} \sum_{k=1}^{J_d} \frac{\partial \log p(\theta_g|\mathbf{z}^{(i)}, \theta_d^{k,m})}{\partial \theta_g} \right) + \mathbf{n}; \mathbf{n} \sim \mathcal{N}(0, 2\alpha\eta I)$$

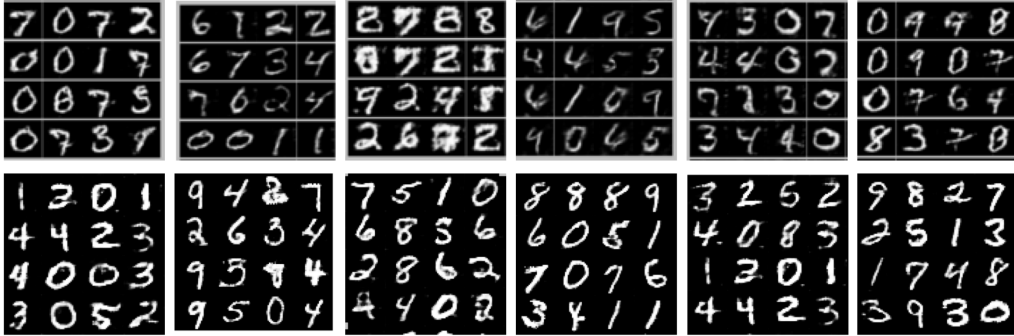
- Append $\theta_g^{j,m}$ to sample set.
- end for**
- for** number of MC iterations J_d **do**
- Sample minibatch of J_d noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(J_d)}\}$ from noise prior $p(\mathbf{z})$.
- Sample minibatch of n_d data samples \mathbf{x} .
- Update sample set representing $p(\theta_d|\mathbf{z}, \theta_g)$ by running SGHMC updates for M iterations:

$$\theta_d^{j,m} \leftarrow \theta_d^{j,m} + \mathbf{v}; \mathbf{v} \leftarrow (1 - \alpha)\mathbf{v} + \eta \left(\sum_{i=1}^{J_d} \sum_{k=1}^{J_g} \frac{\partial \log p(\theta_d|\mathbf{z}^{(i)}, \mathbf{x}, \theta_g^{k,m})}{\partial \theta_d} \right) + \mathbf{n}; \mathbf{n} \sim \mathcal{N}(0, 2\alpha\eta I)$$

- Append $\theta_d^{j,m}$ to sample set.
 - end for**
-

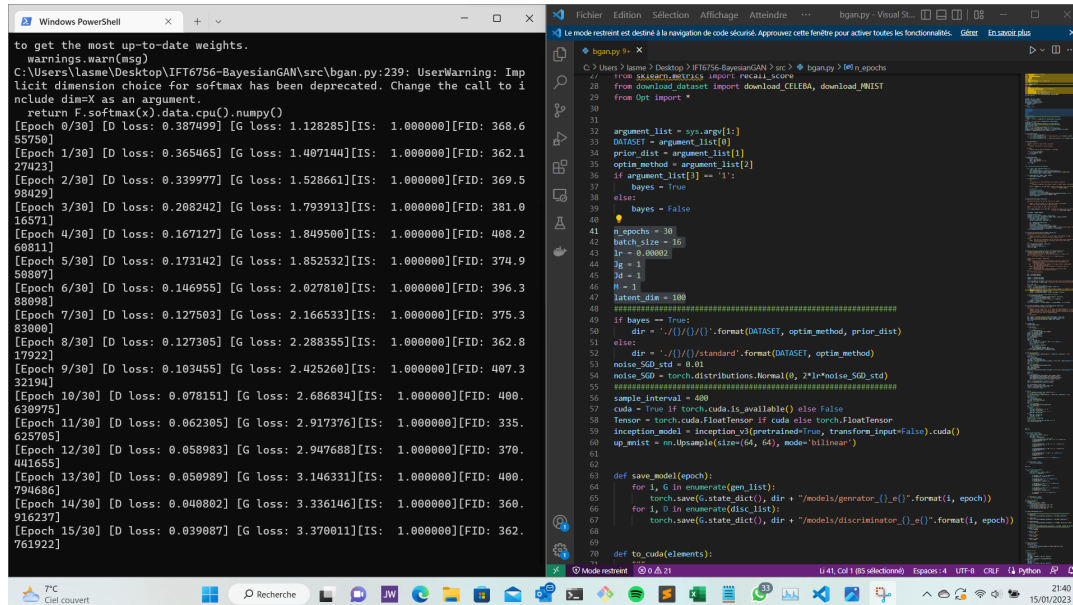
Appendix 3 : Code source of project. All code of this project is available on our GitHub, here : <https://github.com/kouame1999/Bayesian-GANS>.

Appendix 4 : MNIST results.



Top : Data samples from six different generators corresponding to six samples from the posterior over θ_G . The data samples show that each explored setting of the weights θ_G produces generators with complementary high-fidelity samples, corresponding to different styles. The amount of variety in the samples emerges naturally using the Bayesian approach. Bottom : Data samples

from a standard DCGAN (trained six times). By contrast, these samples are homogenous in style.



```
to get the most up-to-date weights.
warnings.warn(msg)
C:\Users\lasme\Desktop\IFT6756-BayesianGAN\src\bgan.py:239: UserWarning: Imp
licit dimension choice for softmax has been deprecated. Change the call to i
nclude dimX as an argument.
  return F.softmax(x).data.cpu().numpy()
[Epoch 0/30] [D loss: 0.387499] [G loss: 1.128285] [IS: 1.000000] [FID: 368.6
55750]
[Epoch 1/30] [D loss: 0.365465] [G loss: 1.487144] [IS: 1.000000] [FID: 362.1
27423]
[Epoch 2/30] [D loss: 0.339977] [G loss: 1.528661] [IS: 1.000000] [FID: 369.5
98029]
[Epoch 3/30] [D loss: 0.208242] [G loss: 1.793913] [IS: 1.000000] [FID: 381.0
16571]
[Epoch 4/30] [D loss: 0.167127] [G loss: 1.849500] [IS: 1.000000] [FID: 408.2
68811]
[Epoch 5/30] [D loss: 0.173142] [G loss: 1.852532] [IS: 1.000000] [FID: 374.9
58897]
[Epoch 6/30] [D loss: 0.146955] [G loss: 2.027810] [IS: 1.000000] [FID: 396.3
88098]
[Epoch 7/30] [D loss: 0.127503] [G loss: 2.166533] [IS: 1.000000] [FID: 375.3
83080]
[Epoch 8/30] [D loss: 0.127305] [G loss: 2.288355] [IS: 1.000000] [FID: 362.8
17922]
[Epoch 9/30] [D loss: 0.103455] [G loss: 2.425260] [IS: 1.000000] [FID: 407.3
32194]
[Epoch 10/30] [D loss: 0.078151] [G loss: 2.686834] [IS: 1.000000] [FID: 400.
630975]
[Epoch 11/30] [D loss: 0.062385] [G loss: 2.917376] [IS: 1.000000] [FID: 335.
625705]
[Epoch 12/30] [D loss: 0.058983] [G loss: 2.947688] [IS: 1.000000] [FID: 370.
441655]
[Epoch 13/30] [D loss: 0.050989] [G loss: 3.146331] [IS: 1.000000] [FID: 400.
794686]
[Epoch 14/30] [D loss: 0.040802] [G loss: 3.336146] [IS: 1.000000] [FID: 360.
916237]
[Epoch 15/30] [D loss: 0.039087] [G loss: 3.370011] [IS: 1.000000] [FID: 362.
761922]
```

```
bgan.py
1 from sklearn.metrics import recall_score
28 from download_dataset import download_CelebA, download_FHIST
29 from opt import *
30
31 argument_list = sys.argv[1:]
32 DATASET = argument_list[0]
33 prior_dist = argument_list[1]
34 optim_method = argument_list[2]
35 if argument_list[3] == '1':
36     bayes = True
37 else:
38     bayes = False
39
40 n_epochs = 30
41 batch_size = 16
42 lr = 0.00002
43 beta = 1
44 beta = 1
45 M = 1
46 latent_dim = 100
47
48 #####
49 if bayes == True:
50     dir = './(())/' + format(DATASET, optim_method, prior_dist)
51 else:
52     dir = './(())/' + format(DATASET, optim_method)
53 noise_std = 0.01
54 noise_std = torch.distributions.Normal(0, 2*lr*noise_std)
55 #####
56 sample_interval = 400
57 cuda = True if torch.cuda.is_available() else False
58 Tensor = torch.cuda.FloatTensor if cuda else torch.FloatTensor
59 inception_model = inception_v3(pretrained=True, transform_input=False).cuda()
60 up_mnist = nn.Upsample(size=(64, 64), mode='bilinear')
61
62
63 def save_model(epoch):
64     for i, G in enumerate(gen_list):
65         torch.save(G.state_dict(), dir + '/models/generator_{}_e({})'.format(i, epoch))
66     for i, D in enumerate(disc_list):
67         torch.save(D.state_dict(), dir + '/models/discriminator_{}_e({})'.format(i, epoch))
68
69
70 def to_cuda(elements):
```

The image above shows some epochs of the BGAN training on our PC for 30 epochs.