Using GAN to Generate Handwritten Digits

Hwai-Jin Peng
Department of Electrical and Computer Engineering
University of Washington
hjpeng@uw.edu

Cheng-Yen Yang
Department of Electrical and Computer Engineering
University of Washington
cycyang@uw.edu

1 Introduction

In this work, we utilize GAN-based models to generate handwritten digits. First, we implement a generative adversarial network (GAN) [1] as our base model. However, since the input of the vanilla GANs is randomly sampled from noise z, we have no idea what will the output image be like. Therefore, we build two other conditional version of GANs, conditional GAN (cGAN) [2] and Auxiliary Classifier GAN (ACGAN) [3], which allow us to condition on to both the generator and discriminator. We'll first illustrate the model structures and how we incorporate digit class (i.e. from 0 to 9) into the networks. Then we'll conduct extensive experiments to compare the results of each model.

2 Models

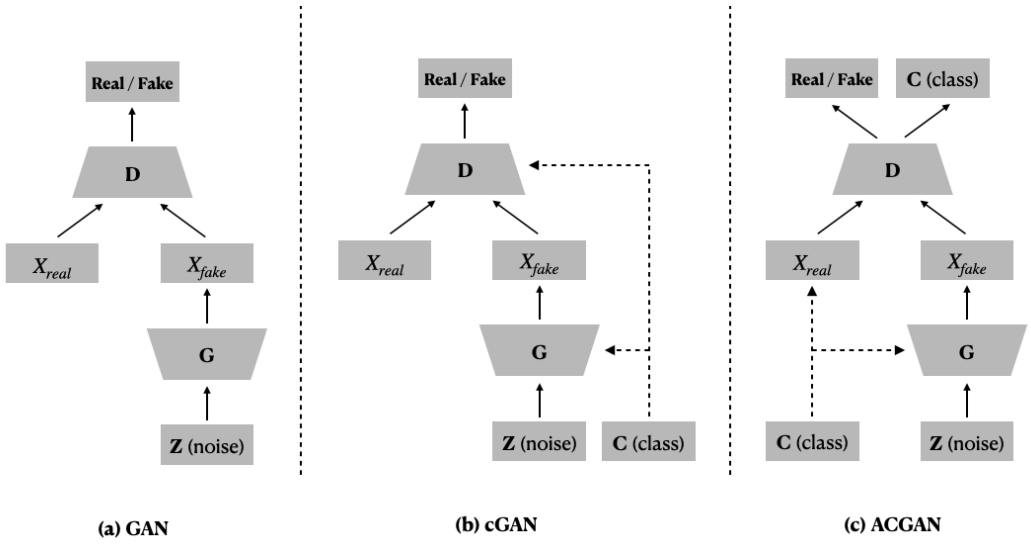

Figure 1: Overview of our GAN, cGAN and ACGAN.

University of Washington: EE 562: Artificial Intelligence for Engineers (Autumn 2020).

In this section, we'll explain how the models work and their objective functions. Also, we also show the architecture of each model. The overview of the vanilla GAN, cGAN and ACGAN is illustrated in Figure. 1. For the implementation details, please refer to our Appendix A.

2.1 Vanilla GAN

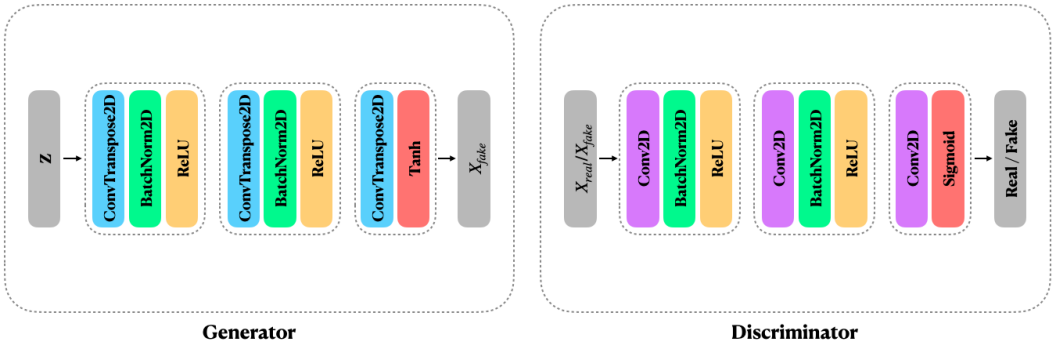

Figure 2: Architecture of GAN.

A generative adversarial network (GAN) consists of two neural networks trained in opposition to one another. The generator G captures the data distribution, while the discriminator D estimates the probability that a sample came from the training data rather than G.

The generator G takes as input a random noise vector z and outputs an image $X_{fake} = G(z)$. The discriminator D receives as input either real images X_{real} from the training set or fake images X_{fake} from G and outputs a probability distribution $D(X)$ over possible image sources. G and D are trained alternately: we adjust the weights for G to minimize $\log\left[1 - D(X_{fake})\right]$ while the weights for D to minimize $\log D(X_{real})$, as if they are following the two-player min-max game with objective function $\mathcal{L}(G, D)$:

$$L = \mathbb{E}[\log D(X_{real})] + \mathbb{E}[\log\left[1 - D(X_{fake})\right]] \tag{1}$$

To relatively stabilize the training of GAN, we follow the protocols and implementations in DCGAN [4]. The size of the input and output image are both 28×28, and we use a 100 dimensional normal distribution z as our noise. The model architecture is illustrated in Figure. 5.

2.2 Conditional GAN (cGAN)

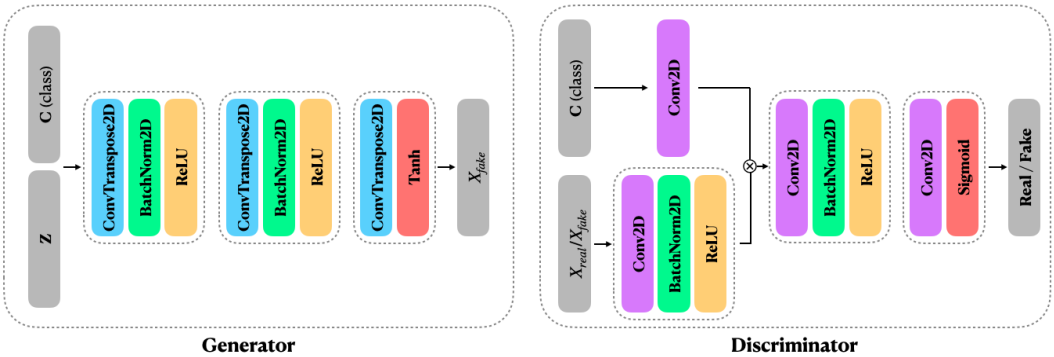

Figure 3: Architecture of cGAN.

To incorporate additional information (e.g., digit classes) into the model, we can extend the vanilla GAN to a conditional one if both G and D are conditioned on some extra constraints y, which could be any kind of auxiliary information such as class labels. In cGAN, we apply the constraints by feeding the digit class labels c into both G and D as additional input layer. Therefore, the adjective function becomes:

$$L = \mathbb{E}[\log D(X_{real}|Class = c)] + \mathbb{E}[\log [1 - D(X_{fake}|Class = c)]] \quad (2)$$

The training approach remains the same as vanilla GAN: D is trained to maximize the log-likelihood that try to assign the incoming images to the correct source while G is trained to minimize the second term in Equation 2.

To feed the class labels into G, we simply concatenate the one-hot vector of the class label to z. The dimension of the mixed input vector becomes 110. As for D, the class label is first forwarded through a convolution layer and then stacked with the original input image. Other configurations remain the same as the vanilla GAN. The model architecture is illustrated in Figure. 3.

2.3 Auxiliary Classifier GAN (ACGAN)

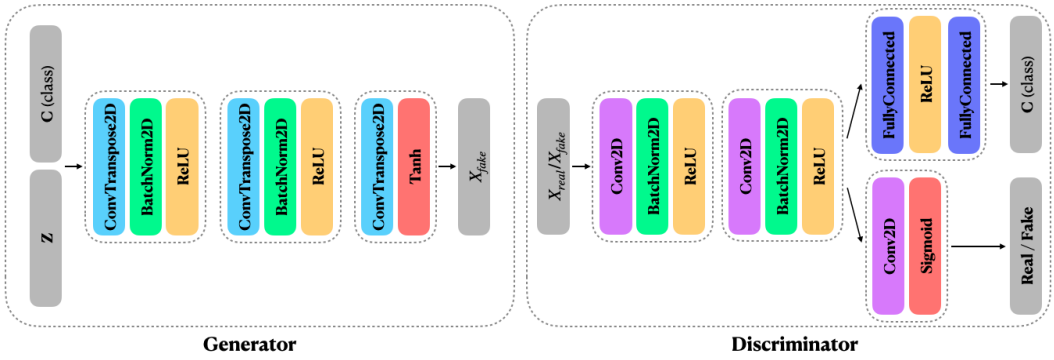

Figure 4: Architecture of ACGAN.

ACGAN introduces an auxiliary classifier to make the GAN model controllable. G takes both noise z and class c as input to generate X_{fake}, while D now outputs one more probability distribution of the class label $P(C|X)$. As a result, the objective function becomes:

$$L_S = \mathbb{E}[\log D(X_{real}|Class = c)] + \mathbb{E}[\log [1 - D(X_{fake}|Class = c)]] \quad (3)$$

$$L_C = \mathbb{E}[logP(Class = c|X_{real})] + \mathbb{E}[logP(Class = c|X_{fake})]. \quad (4)$$

D is trained to maximize $L_S + L_C$, while G is trained to maximize $L_S - L_C$. The generator and discriminator are identical as which in the vanilla GAN, while the classifier extract features from the intermediate layers of the discriminator. Again, we encode the class labels into one-hot vector and concatenate with noise z and feed them into G. The model architecture is illustrated in Figure. 4.

3 Experiment

In this section, we will focus on the experiment setup including dataset and environment we used as well as the results of the three models we implemented: vanilla GAN, cGAN and ACGAN.

3.1 Dataset

The MNIST dataset[5] is a well-known dataset of handwritten digits widely used for training and testing earlier in the field of machine learning. We choose to experiment with the original MNIST digits which are black and white images with 28x28 in terms of size and contains 60,000 training images and 10,000 testing images.

Figure 5: Sample images from the MNIST dataset.

We will use the same train-test splitting in [6] to perform experiments on vanilla GAN, conditional GAN and auxiliary classifier GAN in this section.

3.2 Environment Setup

We choose to implement the models using Pytorch and other open source packages including: numpy and tqdm, as they were manage with conda for easier setup and reproduce purpose. We also use the module tensorboard to help us monitor the training processes in term of the loss from generator and discriminator along with visualize samples generated using the fixed noise.

Our training process can be run with CPU or GPU, the results shown in this section were trained on a 2.3 GHz Quad-Core Intel Core i7 CPU and 32 GB 3733 MHz LPDDR4X memory. Each epoch takes approximately 100 seconds to train for both vanilla GAN, conditional GAN and ACGAN.

3.3 Vanilla GAN

For vanilla GAN, we train the models using a initial learning rate of 0.001 and train for 20 epochs using all 60,000 training samples. We show the samples generated by the generator from the 1st, 10th and 20th epochs in Figure 9.

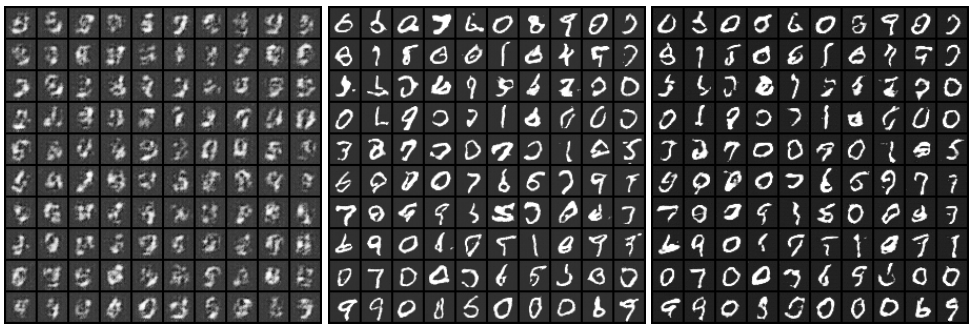

Figure 6: Results from our vanilla GAN (left: 1^{st} epoch, middle: 10^{th} epoch, right: 20^{th} epoch).

We can observe a very interesting trend, which is that even though our noise are randomly sampled, the generator still generates more zero-look-alike image which we think it is because the handwritten digit 'zero' is easier to generalize. From the samples from different epochs during the training process, it is clear to tell that our generator and discriminator are gradually learning to generated different handwritten digits from stretch.

3.4 cGAN

As mentioned in the previous section, we apply the constraints by feeding the digit class labels c into both generator and discriminator in conditional GAN as additional input layer with the noise. We train the models using a initial learning rate of 0.001 and train for 20 epochs using all $60,000$ training samples. We show the samples generated by the generator with fixed noise plus the digit labels from the 1st, 10th and 20th epochs in Figure 7 as each column represents one of the digit class labels from 0 to 9.

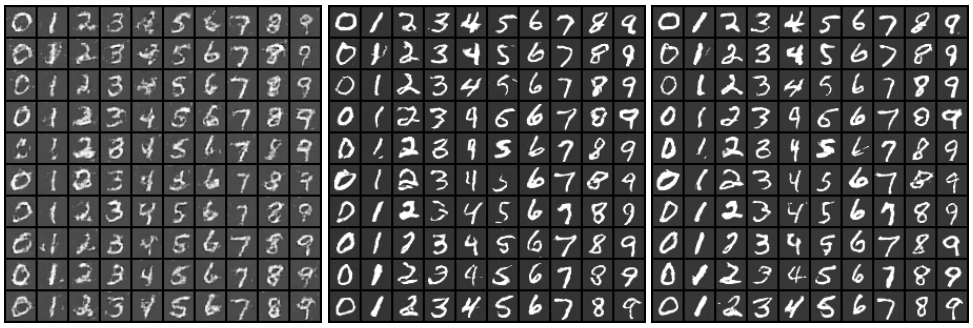

Figure 7: Results from our cGAN (left: 1^{st} epoch, middle: 10^{th} epoch, right: 20^{th} epoch).

We can observe that when we add the digit class labels as input to both generator and discriminator, the generator gradually learn to generate corresponding base on the input digit class. However, we can still observe some mis-generated samples for the digits that have closer appearances like 3 and 8 in 7.

3.5 ACGAN

As mentioned in the previous section, we introduces an auxiliary classifier in additional to the original setup of GAN aiming to take advantage of using the auxiliary classifier to guide our generator to generate better image when the digit class labels is provided. We train the models using a initial learning rate of 0.001 and train for 20 epochs using all $60,000$ training samples. We show the samples generated by the generator with fixed noise plus the digit labels from the 1st, 10th and 20th epochs in Figure 8 as each column represents one of the digit class labels from 0 to 9.

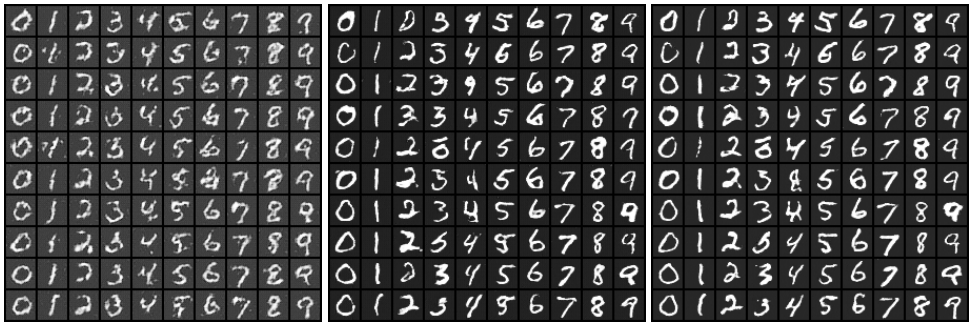

Figure 8: Results from our ACGAN (left: 1^{st} epoch, middle: 10^{th} epoch, right: 20^{th} epoch).

We can observe similar behavior in ACGAN, the overall image qualities are better than vanilla GAN as the generator sort of disentangles the different training samples with the help of digit labels. In

addition to that, we think that with the help of auxiliary classifier, the generator model now have a better performance in terms of generating different handwritten digits.

4 Discussion

4.1 Comparison

In this section, we want to make an overall comparison between the three models: vanilla GAN, conditional GAN, and auxiliary classifier GAN. When using similar training settings, we can clearly tell that cGAN and ACGAN generate better qualities samples in comparison to GAN as GAN tends to generate easier digits like 0 or 1 in order to fool the discriminator. Whereas cGAN and ACGAN use the help of extra digit class label to make the generators capable of generating better quality conditional handwritten digits.

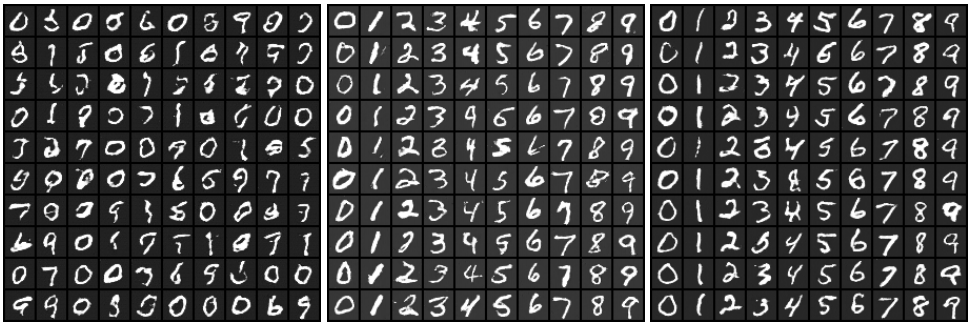

Figure 9: Samples generated from the generators after 20 epochs of training (left: vanilla GAN, middle: conditional GAN, right: auxiliary classifier GAN).

4.2 Data Augmentation

As GAN heavily rely on the training image, we also adapt data augmentation techniques we learn in the course to increase the performance of our models. However, unlike some other real world image datasets, there are some limits when applying the data augmentation technique on the MNIST dataset as they are supposed to be handwritten digits, therefore, something like flipping may not make sense to all of the digits. However the results are not very impressive as we can observe that a number of the samples with the label 3 being generated into label 5.

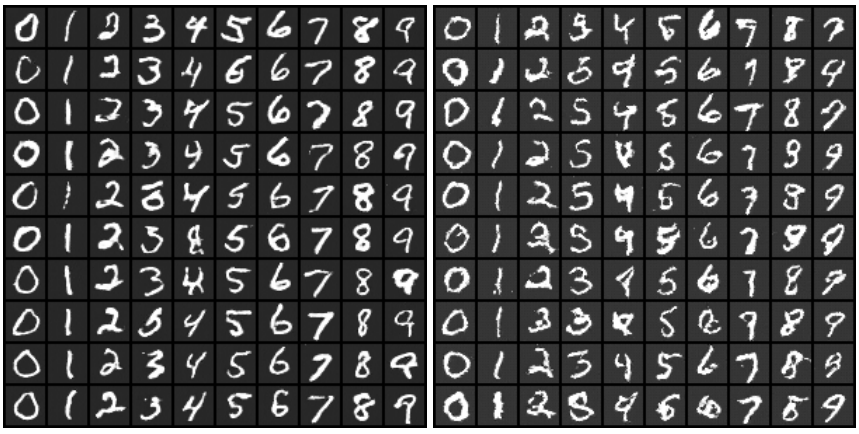

Figure 10: Samples generated from the generators after 20 epochs of training (left: original ACGAN, right: ACGAN w/ data augmentation).

4.3 Mode Collapse

Mode collapse can be observed occasionally when training GANs[1], however, it can be triggered in a seemingly random mechanism. However, in our project, due to the simplicity of the MNIST dataset, mode collapse is rarely observed in comparison to the GAN works on other dataset we surveyed.

5 Conclusion

This work introduces and implements the architecture of GAN, cGAN, and ACGAN and shows that GANs can generate plausible images when sufficient training samples are provided. Moreover, we demonstrate that, with the help of extra digit class labels, our cGAN and ACGAN is able to disentangle and learn to generate good quality handwritten digits samples after numbers of training iterations. While we only consider digit class label as extra conditioned input in our project, other feature like different fonts, different width of digits, and different rotation angles can also be consider as additional input if related labels in available in future works.

A Appendix A: Implementation Details

In this section, we present the detailed architectures for each network in our work.

A.1 GAN

A.1.1 Generator G

The implementation details of the generator G of GAN is shown in Table 1.

Table 1: Architecture of the generator G of GAN.

Layer	Filter Size	Stride	Output Size
Input			100
ConvTranspose2D	7×7	1×1	$64 \times 7 \times 7$
BatchNorm2D			$64 \times 7 \times 7$
ReLU			$64 \times 7 \times 7$
ConvTranspose2D	4×4	2×2	$32 \times 14 \times 14$
BatchNorm2D			$32 \times 14 \times 14$
ReLU			$32 \times 14 \times 14$
ConvTranspose2D	4×4	2×2	$1 \times 28 \times 28$
Tanh			$1 \times 28 \times 28$
Output			$1 \times 28 \times 28$

A.1.2 Discriminator D

The implementation details of the discriminator D of GAN is shown in Table 2.

Table 2: Architecture of the discriminator D of GAN.

Layer	Filter Size	Stride	Output Size
Input			$1 \times 28 \times 28$
Conv2D	4×4	2×2	$32 \times 14 \times 14$
BatchNorm2D			$32 \times 14 \times 14$
ReLU			$32 \times 14 \times 14$
Conv2D	4×4	2×2	$64 \times 7 \times 7$
BatchNorm2D			$64 \times 7 \times 7$
ReLU			$64 \times 7 \times 7$
Conv2D	7×7	1×1	$1 \times 1 \times 1$
Sigmoid			$1 \times 1 \times 1$
Output			1

A.2 cGAN

A.2.1 Generator G

The implementation details of the generator G of cGAN is shown in Table 3. Note that the input is the concatenation of noise vecter and one-hot vector of class labels.

Table 3: Architecture of the generator G of cGAN.

Layer	Filter Size	Stride	Output Size
Input			110
ConvTranspose2D	7×7	1×1	$64 \times 7 \times 7$
BatchNorm2D			$64 \times 7 \times 7$
ReLU			$64 \times 7 \times 7$
ConvTranspose2D	4×4	2×2	$32 \times 14 \times 14$
BatchNorm2D			$32 \times 14 \times 14$
ReLU			$32 \times 14 \times 14$
ConvTranspose2D	4×4	2×2	$1 \times 28 \times 28$
Tanh			$1 \times 28 \times 28$
Output			$1 \times 28 \times 28$

A.2.2 Discriminator D

The implementation details of the discriminator D of cGAN is shown in Table 4. Note that the input images (1) and the input class labels (2) are forwarded through the Conv-BN-ReLU block and one single Conv2D layer, respectively. After that, the output of first Conv-BN-ReLU block (3) and the output of the single Conv2D layer (4) are stacked before fed into the second Conv-BN-ReLU block.

Table 4: Architecture of the Discriminator D of cGAN.

Layer	Filter Size	Stride	Output Size
Input (Image) (1)			$1 \times 28 \times 28$
Input (Class) (2)			$1 \times 28 \times 28$
(1)			$1 \times 28 \times 28$
Conv2D	4×4	2×2	$32 \times 14 \times 14$
BatchNorm2D			$32 \times 14 \times 14$
ReLU (3)			$32 \times 14 \times 14$
(2)			$1 \times 28 \times 28$
Conv2D (4)	4×4	2×2	$32 \times 14 \times 14$
(3) + (4)			$64 \times 14 \times 14$
Conv2D	4×4	2×2	$64 \times 7 \times 7$
BatchNorm2D			$64 \times 7 \times 7$
ReLU			$64 \times 7 \times 7$
Conv2D	7×7	1×1	$1 \times 1 \times 1$
Sigmoid			$1 \times 1 \times 1$
Output			1

A.3 ACGAN

A.3.1 Generator G

The implementation details of the generator G of ACGAN is shown in Table 5. Note that the input is the concatenation of noise vecter and one-hot vector of class labels.

Table 5: Architecture of the Generator G of ACGAN.

Layer	Filter Size	Stride	Output Size
Input			110
ConvTranspose2D	7×7	1×1	$64 \times 7 \times 7$
BatchNorm2D			$64 \times 7 \times 7$
ReLU			$64 \times 7 \times 7$
ConvTranspose2D	4×4	2×2	$32 \times 14 \times 14$
BatchNorm2D			$32 \times 14 \times 14$
ReLU			$32 \times 14 \times 14$
ConvTranspose2D	4×4	2×2	$1 \times 28 \times 28$
Tanh			$1 \times 28 \times 28$
Output			$1 \times 28 \times 28$

A.3.2 Discriminator D

The implementation details of the discriminator D of ACGAN is shown in Table 6. Note that the output of the second Conv-BN-ReLU block is fed into the Conv2D/Sigmoid part (2) to get the real/fake output and the multiple fullyconnected layers (3) to obtain the predicted class labels.

Table 6: Architecture of the Discriminator D of ACGAN.

Layer	Filter Size	Stride	Output Size
Input			$1 \times 28 \times 28$
Conv2D	4×4	2×2	$32 \times 14 \times 14$
BatchNorm2D			$32 \times 14 \times 14$
ReLU			$32 \times 14 \times 14$
Conv2D	4×4	2×2	$64 \times 7 \times 7$
BatchNorm2D			$64 \times 7 \times 7$
ReLU (1)			$64 \times 7 \times 7$
(1)			$64 \times 7 \times 7$
Conv2D	7×7	1×1	$1 \times 1 \times 1$
Sigmoid (2)			$1 \times 1 \times 1$
(1)			$64 \times 7 \times 7$
Flatten			3136
FullyConnected			128
ReLU			128
FullyConnected (3)			10
(2) Output (Real/Fake)			1
(3) Output (Class)			10

B Appendix B: Training Loss

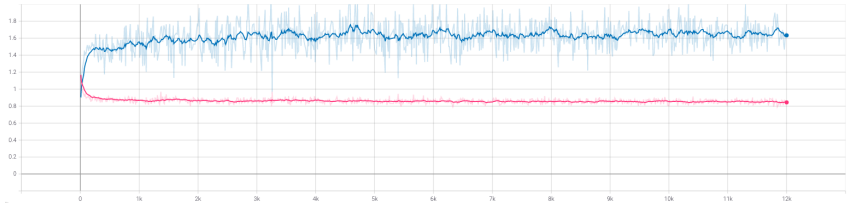

Figure 11: Plot of generator loss (blue) and discriminator loss (pink) from vanilla GAN to global steps.

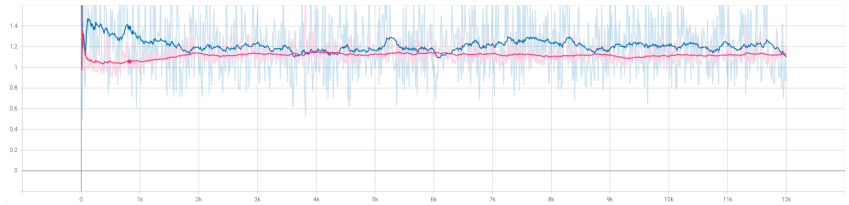

Figure 12: Plot of generator loss (blue) and discriminator loss (pink) from cGAN to global steps.

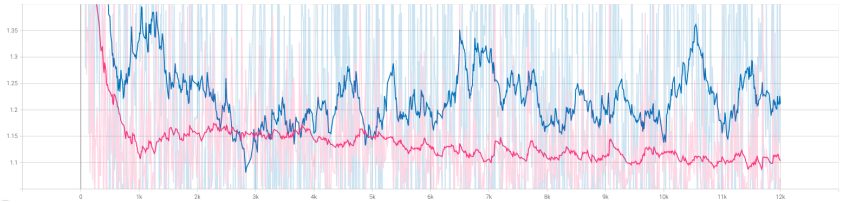

Figure 13: Plot of generator loss (blue) and discriminator loss (pink) from ACGAN to global steps.

References

[1] Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks (2014)

[2] Mirza, M., Osindero, S.: Conditional generative adversarial nets (2014)

[3] Odena, A., Olah, C., Shlens, J.: Conditional image synthesis with auxiliary classifier gans (2017)

[4] Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks (2016)

[5] LeCun, Y., Cortes, C., Burges, C.: Mnist handwritten digit database. ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist **2** (2010)

[6] Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE. pp. 2278–2324 (1998)