Face Generation with Conditional Generative Adversarial Networks

Xuwen Cao, Subramanya Rao Dulloor, Marcella Cindy Prasetio

Abstract

Conditioned face generation is a complex task with many applications in several domains such as security (e.g., generating portraits from description), styling and entertainment. In this project, we explore extensions to Generative Adversarial Networks (GANs) to generate faces conditioned on identity. We implement several conditional GAN (CGAN) variants with multiple network architectures and loss functions – with special emphasis on quantifying the performance of each of these networks using state-of-the-art face detection and recognition systems. Face images generated by our best performing CGAN implementation and trained on a custom celebrity dataset achieves 98% detection rate and 60% accuracy in matching identities.

1 Introduction

Realistic face generation conditioned on identity has many potential applications, ranging from security to styling and entertainment. From the technical viewpoint, the objective is to build a generative model that accurately captures the distribution of data (face images) and reproduce samples from the same distribution that are indistinguishable from the real-world samples. One option is to use "traditional" generative models that are based on maximum likelihood techniques. An example of a "traditional" generative model is the Gaussian mixture model (GMM) – a probabilistic model that assumes all data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. Parameters of the GMM model are learned using the EM algorithm.

Gerative Adversarial Networks (GANs) [2, 5] provide an attractive alternative to "traditional" generative models. GANs are composed of two competing neural networks that train simultaneously in a zero-sum game. One is a generator network that takes random noise as input and generates samples. The other is a discriminative model that tries to differentiate between the samples from generative source and real training data. GANs have proven to be an effective, yet simple (using only backpropagation), framework for training generative models even when the distribution of data is complex.

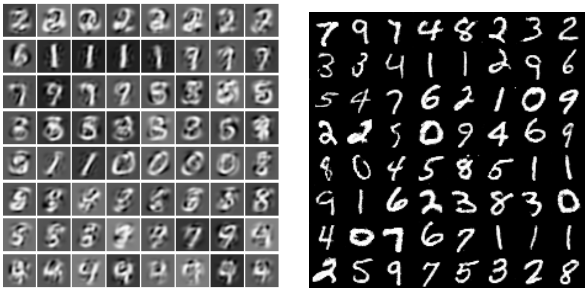


Figure 1: Generated images with varying generative models (a) GMM (with PCA), (b) DCGAN + Wassertein loss

To understand the qualitative differences between the distributions learned and samples generated by the GMM and GAN models, we trained a GMM model on the MNIST dataset [1]. Figure 1 shows generated samples from the distributions learned by the GMM and GAN models. Each data point in MNIST is considered as a 784 dimensional vector by both the models. In the case of GMM, we first reduce the dimensionality of the input vector to 50 using PCA. The GAN model uses a 4-layer convolutional network for generator and discriminator (similar to the DCGAN architecture [5]), and Wasserstein loss [6, 7] (described later). As can be seen, GAN clearly generates much better samples than GMM even in the simple case of MNIST – primarily because GAN is not constrained by the assumptions about the distribution of data (unlike GMM).

For face generation conditioned on identity, we built a variant of GAN called conditional GAN (CGAN) [3]. CGANs allow use of additional variables (conditionals) as input, in addition to the random noise input in a regular GAN. Unlike regular GAN where there is no control over the type of samples generated, a CGAN can generate samples with specific attributes (e.g., samples corresponding to a label). Figure 2

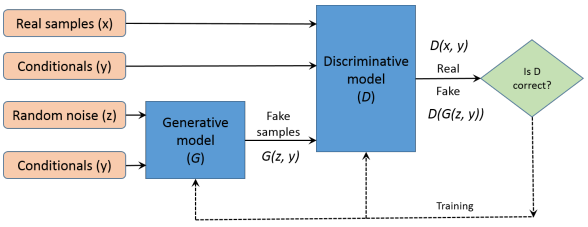


Figure 2: High-level GAN architecture

shows the high-level overview of the CGAN architecture. For this project, we built a CGAN model trained with a custom celebrity face dataset (§2) and conditioned on the identity. For qualitative evaluation, we inspect samples generated by our model and visually confirm that the generated faces are indeed similar to those in the original dataset. In addition, we built classifiers based on state-of-the-art face detection and recognition systems for quantitative evaluation of the faces generated by our models.

2 Dataset and features

We used three datasets in our project - MNIST [1], CelebA [4] and our own custom dataset. MNIST dataset is used for quick validation of our experimental models. Detailed results from MNIST experiments can be found in the Appendix.

The CelebA dataset consists of over 10K identities and over 200K total images. However, there are only around 80 images for each identity in the dataset. And even those identity labels are extremely noisy, with a lot of inaccuracies. After initial experiments with CelebA, it became clear that it is not a good fit for building CGANs conditioned on identity. In the end, we decided to put together our own custom dataset for training our CGAN models.

For our custom dataset, we collected around 3000 face images for 10 famous celebrities using Googles image search. After downloading the images, we first used an open source MT-CNN based tool [8] for detecting, cropping, aligning and extracting faces out of the images. Faces are then rescaled to identical size (160×160), before they are vertically aligned once again using another image rotation tool [9]. We provide each image with identity labels, which are then used as conditionals in the CGAN models. Data preprocessing and clean labeling has a significant impact on the quality of the trained models. (§5.2)

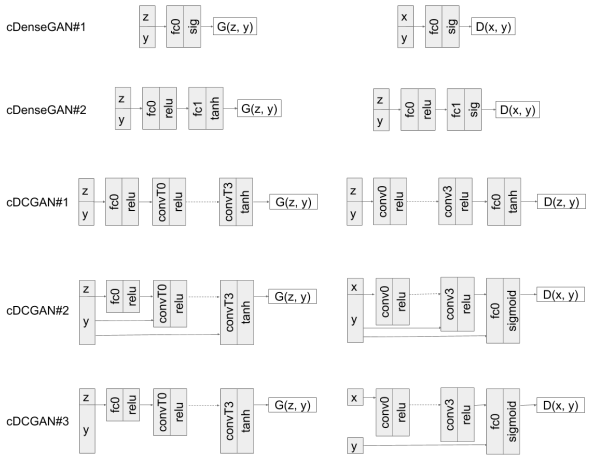


Figure 3: GAN Discriminator and Generator networks

3 Approach

GANs stipulate a high-level framework with a lot of freedom in designing its various components: (i) Generator and Discriminator networks (including how conditionals are fed to these networks), (ii) loss functions, (iii) training/optimization algorithms. There is an additional degree of freedom in choosing hyperparameters for each of these components. We explored the design space in a methodical manner, starting with simple models before implementing models inspired by current state-of-the-art in GANs for image generation [5, 6].

Networks: There are two key considerations in choosing the generator $G(z)$ and discriminator $D(x)$: (i) how many layers and of what type (fully-connected or convolutional), and (ii) how to inject conditionals into the network. Figure 3 shows a subset of explored network architectures. We first started with simple networks containing only of fully-connected (FC) layers. c-DenseGAN#1 contains a single FC layer with 1024 neurons in both G and D, while c-DenseGAN#2 contains two FC layers (each with 1024 neurons) in both G and D. We found that the generated output with both the dense networks are too grainy (shown in Figure 5 for c-DenseGAN#2). This result is not surprising, since (unlike convolutional layers) FC layers are fundamentally limited in their ability to learn spatial structures in an image due to their all-to-all connectivity.

Next we implemented a more complex set of networks inspired by the recent work on deep convolutional GANs (or DCGANs) [5]. In all of the explored DC-

GAN variants, $G(z)$ consists of one FC layer followed by four transponsed convolutional layers (sometimes incorrectly called deconvolutions). And $D(x)$ consists of four convolutional layers followed by a single FC layer. The variants differ in the activation functions used in G and D (§5.2), and more importantly in the way conditionals are injected into G and D. Figure 3 shows some of these c-DCGAN variants.

It is important to note that we encode the conditionals as one-hot vectors. As a result, when we inject conditionals into the network (at any layer), it could potentially convert a dense tensor to a sparse one. We found that the place where we inject conditionals in the network (both for G and D) has a major impact on GAN's overall performance. We explored a large number of network designs between the two limits – conditionals fed only to the first layer of G and D (c-DCGAN#1) & conditionals fed to every single layer of G and D (c-DCGAN#2). Figure 3 shows these two networks and also c-DCGAN#3, which is our best performing model. We describe in a later section intuitions and experiments that led us to c-DCGAN#3 (§5.2).

Loss functions: The objective of training a GAN model is to arrive at a Nash equilibrium between G and D, where D is unable to distinguish between the real and generated samples. I.e., *Dloss* is close to 0.5. We started with the *log* (or *cross-entropy/CE*) *loss* function proposed in the original GAN paper [2]. The objective for GAN training with CE loss can be written as follows.

$$\min_G \max_D V(D,G) = E\Big[\log(D(x))\Big] + E\Big[\log(1 - D(G(z)))\Big].$$

In addition to the CE loss, we also experimented with *Wasserstein loss* [6], which is purported to fix two common problems with CE loss: (i) prevent mode collapse by increasing the penalty for generators that cheat by producing a very small number of almost perfect samples, (ii) provide a metric for parametric convergence. The objective for GAN training with Wasserstein loss is as follows.

$$\min_G \max_D V(D,G) = E\Big[D(x)\Big] + E\Big[-D(G(z))\Big].$$

We found that Wasserstein loss produces worse results than the simpler log loss in our models (§5.2).

4 Experiments

We experimented on a large number of c-DenseGAN and c-DCGAN models, but (for brevity) we show results only for the models in Figure 3. Unless mentioned otherwise, all CGAN models use Adam optimizer with learning rate 0.0002 and CE loss. We use c-DenseGAN as a simple baseline model to compare how the c-DCGAN models perform. We present results only for a subset of our conditional face generation experiments. The entire set of our experiments can be found in our implementation code.

Model	Accuracy		
	Face detection	ID match full	ID match detected
c-DenseGAN#1	89.5%	10.1%	10.5%
c-DenseGAN#2	96.1%	15.7%	16.1%
c-DCGAN#1	99.5%	5.6%	5.7%
c-DCGAN#2	3.5%	7.4%	0%
c-DCGAN#3	98%	59.4%	60.2%

Figure 4: An evaluation of the generated samples with the MT-CNN face detector and Facenet embeddings for identity matching. *ID match full* measures identity matches on all generated samples, while *ID match detected* measures this metric only samples for which MT-CNN successfully detected a face.

For qualitative analysis, we compare the generated images from our best c-DenseGAN and c-DCGAN models with the original input for each identity. For quantitative analysis, we have two metrics:

(1) the accuracy of face detection, which measures the percentage of generated samples for which the MT-CNN face detector was able to detect a face, and

(2) the accuracy of the face identity matching, which measures the "similarity" of the generated faces with the original face images of the same identity. To measure "similarity" we built an identity classifier trained on Facenet embeddings [8] for the images in our custom dataset. We compute this accuracy measure separately on the entire generated set and on successfully detected faces.

5 Results

5.1 Overall Result

Table 4 shows the overall results for face generation for the five CGAN networks in Figure 3 with cross-entropy (CE) loss. As we can observe, the best out of the five models is the c-DCGAN#3.

Figure 5 shows c-DenseGAN#2 generates images with good alignment. c-DenseGAN#2 also cap-

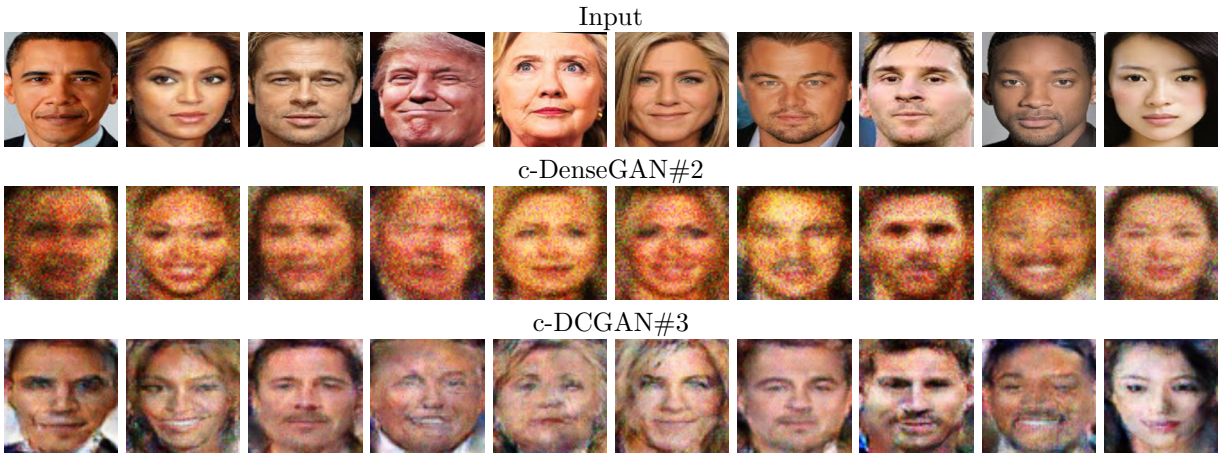
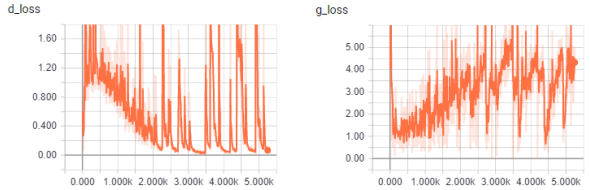


Figure 5: Comparison for generated images of the five best model with the input images. Each column represents a unique identity.

tures the identity better than c-DCGAN#1 and c-DCGAN#2 as seen from the accuracy score in Table 4. However, the quality of the images is far inferior (grainy and blurry) to the input images. On the other hand, c-DCGAN#3 not only has better visual quality than c-DenseGAN#2, but is also objectively better than all other models across all metrics. We present detailed error analysis on the c-DCGAN experiments and explain how we came up with the best performing model in the next section.



(a) d_loss (b) g_loss

Figure 6: Discriminator loss (d_loss) and Generator loss (g_loss) for c-DCGAN#3

5.2 Error analysis

Network related: In our initial experiments with c-DCGAN models, we were using lrelu activation for the intermediate layers and sigmoid activation in the final layer of the generator. Samples generated by this c-DCGAN model were non-face images with dark pixels, as seen in Figure 7a. The same model worked well with the simpler MNIST dataset. We attribute this to the fact that with the custom dataset we are dealing with 3-channel images (different from the grayscale MNIST images) and also a large number of "zero" pixels introduced by the one-hot encoded conditional vector. In addition, the restricted domain of sigmoid function (0 to 1) results in a diminishing gradient problem at the generator's output layer. The first change to the network was to use tanh activation function in the final layer of the generator. However, as shown in Figure 7b, this change alone was not enough to generate visible face images. In addition, we had to replace the leaky ReLu activation with ReLu in the generator's hidden layers as suggested in prior work [5], which resulted in the slightly better result in Figure 7c.

Data pre-processsing related: After dealing with the above mentioned issues, the images generated by our models started looking like faces but were still of very poor quality (obsure, malformed, and blurry). We suspected that this may be due to noisy data. Some images in our custom dataset were not aligned and had different margins between the border and the face. Figure 7c shows the result of this problem. After rotating and cropping the images, we managed to get better and more consistent results as shown in the results for c-DCGAN#1 in Figure 7f.

Loss function related: In terms of loss function variants, we experimented with Wassertein distance [6] as the c-DCGAN loss function to see if it brings any significant improvements. Figure 7d shows the results for this experiment. Although the model produces valid face images, the samples generated by this model are significantly worse than those of CE loss based models. For this part of the problem, we surmised that implementing Wassertein loss may require additional modifications to the model architecture such as weight clipping. However, weight clip-

(a) (b) (c) (d) (e) (f)

Figure 7: Error analysis of the c-DCGAN. (a) Generator activation function (sigmoid instead of tanh), (b) Leaky RELU in generators hidden layers, (c) Noisy data input (unaligned), (d) Wasserstein loss, (e) Mixing images and conditionals directly in every layer (c-DCGAN#2), (f) Conditionals on the first layer for both discriminator and generator (c-DCGAN#1)

ping (using an existing TensorFlow operator) was resulting in obscure OOM errors with our DCGAN implementation. Since the CE loss models are already performing well, we decided to explore other variables. Figure 6 shows the discriminator and generator loss (over the training period) for our c-DCGAN#3 model which uses CE loss.

Conditionals placement related: One of the key differences between our c-DCGAN models is where we inject the one-hot conditional vector in both the generator and discriminator networks. In our first implementation we injected the conditionals only at the first layer in both the discriminator and generator (c-DCGAN#1). Figure 7f shows samples generated by the model. We observe that while the image quality is superior to that of c-DenseGAN, some images are still blurry. However, the ID-match accuracy is too low (just 5.6%), confirming what we see visually – the model does not do a good job of generating faces conditioned on identity.

We then experimented with many different ways of injecting the conditionals into the networks, but in this report we want to highlight and compare two such models. The first model mixes images with the conditionals in every layer in both the discriminator and generator networks (c-DCGAN#2). The notion behind this method is that given sufficient mixing of conditionals at every layer, the model will be able to better learn the joint distribution of images and labels and produce good quality images with the correct identities. However, as we can see in Figure 7e, the generated images are significantly worse – off color and with too many bright pixels. This is due to the fact that mixing conditionals in every layer results in a very sparse tensor with a lot of zeros in each of the layers.

Therefore, we started exploring other network architectures between the extremes of cDCGAN#1 and cDCGAN#2, and arrived at our best performing model c-DCGAN#3. The idea behind c-DCGAN#3 is that there is no good reason to inject conditionals at any of the convolutional layers since conditionals do not have any spatial aspect. Therefore, in c-DCGAN#3, we inject conditionals only at the dense layers of both the discriminator (the last layer FC0 in D) and the generator (the first layer FC0 in G). As shown in Figure 5 and Table 4, c-DCGAN#3 model results in a very significant improvement in both the accuracy score and the generated image quality, compared to all other c-DenseGAN and c-DCGAN models.

6 Conclusion and Future Work

Face generation conditioned on identity is a complex task that requires a model to learn not only the general data distribution for face images, but also the face embeddings for each individual identity. We have demonstrated in this project that well-designed conditional GANs (CGANs) can perform admirably on this very complex task. Provided high-quality input data, these CGANs can be trained so that the combination of a conditional and the noise vector (z) actually encodes a face embedding. We plan to expand this work in two directions – (i) improving the quality of image generation, (ii) supporting a very large number of identities.

7 Contributions

Equal contributions from all project members.

References

[1] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database". In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[2] Zoubin Ghahramani et al., eds. *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. 2014.

[3] Mehdi Mirza and Simon Osindero. "Conditional Generative Adversarial Nets". In: *CoRR* abs/1411.1784 (2014). URL: http://arxiv.org/abs/1411.1784.

[4] Ziwei Liu et al. "Deep Learning Face Attributes in the Wild". In: *Proceedings of International Conference on Computer Vision (ICCV)*. 2015.

[5] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: *CoRR* abs/1511.06434 (2015).

[6] Martin Arjovsky, Soumith Chintala, and Lfffdfffdon Bottou. "Wasserstein GAN". In: *arXiv* 1701.07875 (2017). URL: https://arxiv.org/abs/1701.07875.

[7] Ishaan Gulrajani et al. "Improved Training of Wasserstein GANs". In: *CoRR* abs/1704.00028 (2017).

[8] *Face Recognition using Tensorflow*. https://github.com/cindy21td/facenet.

[9] *Face-Align*. https://github.com/juniorxsound/Face-Align.

A MNIST

For quick validation of our experimental CGAN models, we perform the first-order evaluation with the simpler MNIST dataset. MNIST requires significantly less resources than the CelebA and custom celebrity face dataset, while still providing useful feedback on the CGAN architecture. The MNIST dataset contains 60000 images of handwritten digits. We used it to train and verify the correctness of our initial conditional GANs. The inputs for the discriminator are 28x28 image pixels of the digits, and the number digit the images represent as conditionals provided by the dataset. For evaluation of the generated digit samples produced by our CGANs, we built a deep MNIST digit classifier with 99.2% accuracy. We compare the performance of this MNIST classifier on the original MNIST dataset with its performance on generated samples to evaluate the quality of the GAN model.

Model	Accuracy
Dense CGAN (Baseline)	88.2812%
DCGAN	98.4375%
DCGAN + Extra Dense layer	99.2188%
DCGAN + Logloss	100.0000%
DCGAN + Wasserstein loss	97.6562%

Table 1: An evaluation of the generated samples with the MNIST classifier

Table 1 shows results from the MNIST classifier test on the various CGAN models. Dense CGAN has only 88.28% accuracy in our MNIST classifier test and it generates visibly low quality images (Figure 8). We suspect that the simply fully-connected network is not sophisticated enough to represent the underlying data distribution of the MNIST dataset. Using convolutional layers (similar to DCGAN) [5] improves the accuracy result and image quality better than the Dense CGAN. Going forward we will refer to this model as DCGAN. The image quality and accuracy are improved further when we add an additional dense layer to DCGAN (DCGAN + Extra Dense Layer). A combination of convolutional, deconvolutional, and dense (fully-connected) layers results in the best performance, both in terms of accuracy and the visible image quality.

For the loss function experiments, we compare cross-entropy with log loss, which we learned in class. We expected them to have similar performance. However, notably, the log loss gives better performance on our classifier (100%). On the other hand, Wassertein loss, which is generally known to have better performance in GAN, actually gives worse accuracy than the cross-entropy loss, even though it generates better image quality in Figure 9. We suspect that Wassertein loss requires modifications to our current model architecture to perform well.

(a) (b) (c)

Figure 8: Generated images with varying discriminator and generator models (a) Dense-CGAN, (b) DCGAN, (c) DCGAN + Extra Dense Layer

(a) (b) (c)

Figure 9: Generated images with varying loss functions (a) DCGAN, (b) DCGAN + Logloss, (c) DCGAN + Wassertein loss