Constrained Generative Adversarial Networks for Interactive Image Generation

Eric Heim

Air Force Research Laboratory
Information Directorate
Rome, NY USA

eheim602@gmail.com

Abstract

Generative Adversarial Networks (GANs) have received a great deal of attention due in part to recent success in generating original, high-quality samples from visual domains. However, most current methods only allow for users to guide this image generation process through limited interactions. In this work we develop a novel GAN framework that allows humans to be "in-the-loop" of the image generation process. Our technique iteratively accepts relative constraints of the form "Generate an image more like image A than image B". After each constraint is given, the user is presented with new outputs from the GAN, informing the next round of feedback. This feedback is used to constrain the output of the GAN with respect to an underlying semantic space that can be designed to model a variety of different notions of similarity (e.g. classes, attributes, object relationships, color, etc.). In our experiments, we show that our GAN framework is able to generate images that are of comparable quality to equivalent unsupervised GANs while satisfying a large number of the constraints provided by users, effectively changing a GAN into one that allows users interactive control over image generation without sacrificing image quality.

1. Introduction

Learning a generative model from data is a task that has gotten recent attention due to a number of breakthroughs in complex data domains [15, 30, 13]. Some of the most striking successes have been in creating novel imagery using Generative Adversarial Networks (GANs) [6]. While GANs show promise in having machines effectively "draw" realistic pictures, the mechanisms for allowing humans to guide the image generation process have been largely limited to conditioning on class labels [21] (e.g. "Draw a zero.") or domain-specific attributes [35] (e.g. "Draw a coat with stripes."). Such feedback, though powerful, limits the user to expressing feedback through a pre-defined set of labels. If the user is unable to accurately express the characteristics
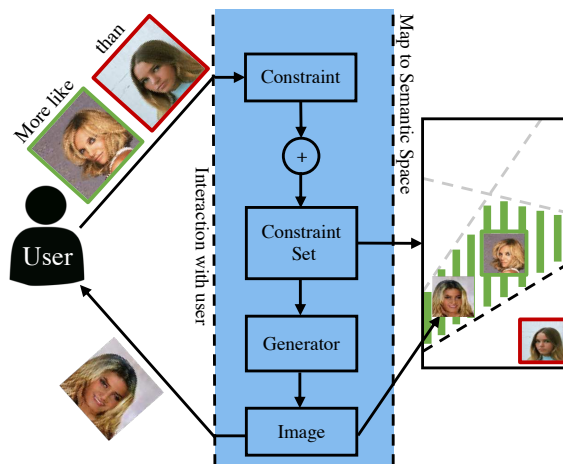

Figure 1: Interaction with the CONGAN generator: A user provides a relative constraint in the form of two images meaning "Generate an image more like image A than image B." The constraint is combined to previously given constraints to form a set, which is input to the generator to produce an image. This image is shown to the user to drive further iterations of feedback. The goal for the generator is to "satisfy" the constraints with respect to a mapping to an underlying semantic space. The generator satisfies a constraint (A, B) by producing an image that is mapped to a coordinate closer to where A is mapped than to where B is mapped.

that they desire using this label set, then they cannot guide the model to produce acceptable images.

In this work, we seek a more natural and powerful way for humans to interact with a generative model. To this end, we propose a novel GAN technique we call CONstrained GAN (CONGAN). Our model is designed to accept human feedback iteratively, effectively putting users "in-the-loop" of the generation process. Figure 1 illustrates how a user interacts with the CONGAN generator. The generator accepts *relative* constraints of the form "More like image A than image B."

These constraints are used to define a feasible region within a given *semantic space* that models an underlying notion of similarity between images. The goal of the generator is to accept relative constraints as input, and output an image that is within the corresponding feasible region.

Modeling interaction in this way has two primary benefits. First, such relative pair-wise assessments have been shown to be an easy medium for humans to articulate similarity [14, 26]. As such, CONGAN allows users to refine its output in a natural way. Relative constraints can also be used to allow for different interactions besides providing pair-wise comparisons. If the output from the generator is then input as the B ("less similar") image in the next iteration, the user need only provide the A ("more similar") image. In this way, the user can provide single examples, meaning "More like image A than what was previously generated", to refine the output. Second, within the CONGAN framework, the semantic space defines the characteristics that users guide the image generation process. This allows for the option of a variety of notions of similarity, such as class and attribute information, but also more continuous or complex notions such as color spaces, or size of objects within the image.

To achieve this form of interaction, our model must have multiple interrelated components. The generator must be able to accept a variable number of constraints as a *set*, i.e. the output should be invariant to the order of the input constraints. For this, we leverage recent work in Memory Networks [7, 34, 27, 31] within the CONGAN generator to learn a fixed length vector representation over the constraint set as a whole. In addition, the generator must not only be able to generate realistic looking images, but images that are within the feasible region of a given semantic space. During training, the CONGAN generator is trained against a *constraint critic* that enforces the output to satisfy given constraints. The result is a generator that is able to produce imagery guided by iterative relative feedback.

The remainder of the paper will proceed as follows. First, we discuss prior related work. Then, we describe our method beginning with a formal definition of the constrained generation problem, continuing to an outline of the CONGAN training algorithm, and ending with a description of the CONGAN generator. Next, we perform an evaluation where we compare our method to an unsupervised GAN, showing qualitative and quantitative results. Finally, we conclude.

2. Related Work

Our proposed CONGAN method follows from a long line of work in neural network image generation. Specifically, autoencoders [15], autoregressive models [30], and generative adversarial networks [6] (GANs) have all shown recent success. We chose to learn a model using the GAN framework, as GANs are arguably the best performing generative models in terms of qualitative image quality.

Much of the fundamental work in GANs have focused on unsupervised learning settings [6, 39, 1]. The output of these models can be controlled by manipulating the latent space used as input [23, 22]. However, such manipulation is limited in that the latent space often has no obvious human understandable interpretation. Thus finding ways to manipulate it requires either trial and error or interpolating between two points in the latent space. Other works learn *conditional* GAN models [21], where generation is guided by side information, such as class labels [21], visual attributes [35], text [24], and images [28]. In this work, we aim to develop a method that allows more intuitive manipulation of a GANs output that generalizes to many different forms of similarity.

The GAN method most similar to ours is the one introduced in [40]. This method first maps an image to a manifold of natural images using a GAN. Then, they provide a series of image editing operations that users can use to move the image along that manifold. We see our work as related but orthogonal to this work as both the means for manipulation, as well as the goals of the methods differ.

Another line of research that motivates this work is interactive learning over imagery. Much of the work in this field has focused on classification problems [3, 32, 17, 33], but also others such as learning localized attributes [4]. Most notably, in [16] the authors propose an interactive image search method that allows users to provide iterative refinements to their query, based on visual attributes. This is similar in principle to our method in that their method searches images through interactive comparisons to other images in the domain of interest. However, our method does not necessarily require predefined attributes and generates novel imagery instead of retrieving relevant images from a database.

3. A Model for Constrained Image Generation

The goal of this work is to learn an image generation model in the form of a mapping from a set of pair-wise relative constraints to a realistic looking image. Let $\mathcal{X}$ be a domain of images. We wish to learn the mapping:

$$g_\Theta : \left\{ (\mathcal{X} \times \mathcal{X})^i \mid i \geq 1 \right\} \times Z \mapsto \mathcal{X}$$

This *generator* maps a set of constraints $\mathcal{C} = \{C_1, C_2, ...\}$ and a random noise vector $\mathbf{z} \in Z$ to an image, where a constraint $C = (\mathbf{X}_+, \mathbf{X}_-) \in \mathcal{X} \times \mathcal{X}$ is a pair of images meaning "Generate an image more like $\mathbf{X}_+$ than $\mathbf{X}_-$." Intuitively, $\mathbf{z}$ represents the variation of imagery allowed within the constraints, and different $\mathbf{z}$ will produce different images. Practically, $\mathbf{z}$ provides the noise component necessary for our generator to be trained within the GAN framework.

For training our generator, we require a mechanism that determines whether the output of g_Θ satisfies input constraints. To this end, we assume the existence of a mapping $\phi : \mathcal{X} \mapsto \mathcal{S}$ that maps images to a *semantic space*. The only

Algorithm 1 CONGAN Training Procedure

Input: Gradient penalty coefficient λ, constraint penalty coefficient γ, discriminator iterations per generator iteration n_{disc}, batch size m, Adam optimizer parameters α, β_1, β_2
  repeat
    for $t = 1, ... n_{disc}$ **do**
      for $i = 1, ..., m$ **do**
        Sample $\mathbf{X} \sim \mathbb{P}_\mathcal{D}, \mathcal{C} \sim \mathbb{P}_\mathcal{C}, \mathbf{z} \sim Z, \epsilon \sim \mathcal{U}(0, 1)$
        $\hat{\mathbf{X}} \leftarrow g_\Theta(\mathcal{C}, \mathbf{z})$
        $\tilde{\mathbf{X}} \leftarrow \epsilon \mathbf{X} + (1 - \epsilon) \hat{\mathbf{X}}$
        $L^i \leftarrow d_W(\hat{\mathbf{X}}) - d_W(\mathbf{X}) + \lambda(||\nabla_{\tilde{\mathbf{X}}} d_W(\tilde{\mathbf{X}})||_2 - 1)^2$
      end for
      $W \leftarrow \text{Adam}\left(\nabla_W \frac{1}{m} \sum_{i=1}^m L^i, \alpha, \beta_1, \beta_2\right)$
    end for
    Sample batches $\{\mathbf{z}^i\}_{i=1}^m \sim Z, \{\mathcal{C}^i\}_{i=1}^m \sim \mathbb{P}_\mathcal{C}$
    $\{\hat{\mathbf{X}}^i\}_{i=1}^m \leftarrow \{g_\Theta(\mathcal{C}^i, \mathbf{z}^i)\}_{i=1}^m$
    $L \leftarrow \frac{1}{m} \sum_{i=1}^m -d_W(\hat{\mathbf{X}}^i) + \gamma l_{\phi, \mathcal{S}}(\hat{\mathbf{X}}^i)$
    $\Theta \leftarrow \text{Adam}(\nabla_\Theta L, \alpha, \beta_1, \beta_2)$
  until Θ converged

---

requirements are that ϕ be differentiable, and that there exists a distance metric $d_\mathcal{S}$ over elements of $\mathcal{S}$. For instance, if one wanted to have users manipulate generated images by their attributes (i.e. the dimensions of $\mathcal{S}$ correspond to attributes), ϕ could be a learned attribute classifier (for binary attributes) or regressor (for continuous attributes). We say a generated image $\hat{\mathbf{X}}$ satisfies a given constraint $C = (\mathbf{X}_+, \mathbf{X}_-)$ with respect to $\mathcal{S}$ if the following holds:

$$d_\mathcal{S}\left(\phi(\hat{\mathbf{X}}), \phi(\mathbf{X}_+)\right) < d_\mathcal{S}\left(\phi(\hat{\mathbf{X}}), \phi(\mathbf{X}_-)\right) \qquad (1)$$

Given a set of constraints $\mathcal{C}$, the goal of g_Θ is to produce an $\hat{\mathbf{X}}$ that satisfies all constraints in the set. In doing so, the generator produces images that are closer in the semantic space to "positive" images $\mathbf{X}_+$ than "negative" images $\mathbf{X}_-$. Put another way, $\mathcal{C}$ defines a feasible region in $\mathcal{S}$ for which $\hat{\mathbf{X}}$ must lie in. How we use this idea of relative constraints to train g_Θ is discussed in the following section.

3.1. Adversarial Training with Relative Constraints

To train the generator g_Θ, we utilize the GAN framework that pits a generator g_Θ against a discriminator d_W, where both g and d and neural networks parameterized by Θ and W, respectively. The discriminator is trained to distinguish outputs of the generator from real image samples. The generator is trained to produce images that the discriminator cannot differentiate from real samples. The two are trained against one another; at convergence, the generator is often able to produce instances that are similar to real samples.

While d_W ensures output images look realistic, we use another model to enforce constraint satisfaction. For this,

we introduce the idea of a *constraint critic* that informs the training procedure in a similar manner as d_W. We define the constraint critic loss as the average loss over each constraint after mapping images into the semantic space:

$$l_{\phi, \mathcal{S}}(\hat{\mathbf{X}}, \mathcal{C}) = -\frac{1}{|\mathcal{C}|} \sum_{(\mathbf{X}_+, \mathbf{X}_-) \in \mathcal{C}} p_\mathcal{S}(\phi(\hat{\mathbf{X}}), \phi(\mathbf{X}_+), \phi(\mathbf{X}_-))$$

Loss over each constraint $p_\mathcal{S}$ is inspired by the loss used in t-Distributed Stochastic Triplet Embedding (STE) [29]:

$$p_\mathcal{S}(a, b, c) = \frac{\left(1 + \frac{d_\mathcal{S}(a,b)}{\alpha}\right)^{-\frac{\alpha+1}{2}}}{\left(1 + \frac{d_\mathcal{S}(a,b)}{\alpha}\right)^{-\frac{\alpha+1}{2}} + \left(1 + \frac{d_\mathcal{S}(a,c)}{\alpha}\right)^{-\frac{\alpha+1}{2}}}$$

This loss compares pairs of objects according to a t-Student kernel and is motivated by successes in dimensionality reduction techniques that use heavy tailed similarity kernels [19]. By minimizing the negation of $p_\mathcal{S}$ for each constraint, $\hat{\mathbf{X}}$ is "pulled" closer to images $\mathbf{X}_+$ and "pushed" farther from images $\mathbf{X}_-$ in $\mathcal{S}$. As a result, using this loss during training will produce images more likely to satisfy constraints.

We leverage the constraint critic loss in tandem with the discriminator to train the CONGAN generator. More specifically, our training algorithm is an extension of the Wasserstein GAN [1, 8]. We aim to optimize the following:

$$\min_\Theta \max_W \mathop{\mathbb{E}}_{\mathbf{X} \sim P_\mathcal{D}} [d_W(\mathbf{X})] - \mathop{\mathbb{E}}_{\hat{\mathbf{X}} \sim P_g}\left[d_W(\hat{\mathbf{X}}) - \gamma l_{\phi, \mathcal{S}}(\hat{\mathbf{X}}, \mathcal{C})\right]$$

Here, $P_\mathcal{D}$ is a data distribution (i.e. $\mathbf{X}$ is a sample from a training set), P_g is the generator distribution (i.e. $\hat{\mathbf{X}} = g_\Theta(\mathcal{C}, \mathbf{z})$ for a given $\mathbf{z} \sim Z$ and a given $\mathcal{C}$ drawn from a training set of constraint sets). Finally, d_W is constrained to be 1-Lipschitz. This objective is optimized by alternating between updating discriminator parameters W and generator parameters Θ using stochastic gradient descent, sampling from the training set and generator where necessary. Intuitively, the discriminator's output can be interpreted as a score of how likely the input is from the data distribution. When the discriminator updates, it attempts to increase its score for real samples and decrease its score for generated samples. Conversely, when the generator updates, it attempts to increase the discriminator's score for generated images. In addition, generator updates decrease the constraint loss by a factor of the hyperparameter γ. As a result, generator updates encourage g_Θ to produce images similar to those in the image training set, while also satisfying samples from a constraint training set. To enforce the 1-Lipschitz constraint on d_W we use the gradient penalty term proposed in [8].

The CONGAN training procedure is outlined in Alg. 1. This algorithm is very similar to the WGAN training algorithm (Algorithm 1 in [8]) with a few key additions. First, when updating both the discriminator and generator, batches
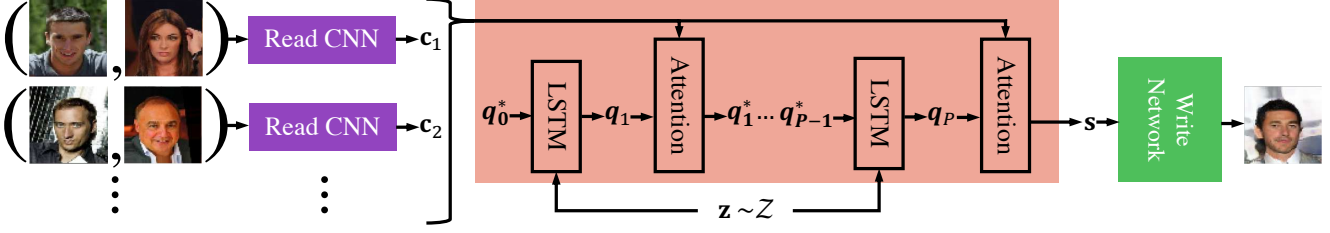
Figure 2: The CONGAN generator. Purple is the read network, orange is the process network, and green is the write network

of constraint sets are selected from a training set. In practice, we use ϕ to construct ground truth constraint sets of variable length from images in the image train set, ensuring that our generator is trained on constraint sets that are feasible in $\mathcal{S}$. Second, the generator update has an additional term: the constraint critic term that encourages constraint satisfaction.

3.2. A Constrained Generator Network

While Alg. 1 outlines how to train g_Θ, we have yet to formally define g_Θ. In order for g_Θ to accept $\mathcal{C}$ as a *set* it must 1) accept a variable number of constraints, and 2) output the same image regardless of the order in which constraints are given. For this we leverage the work of [31] that introduces a neural network framework capable of considering order-invariant inputs, such as sets. An illustration of the CONGAN generator is depicted in Fig. 2. Our generator has three components: 1) A *read* network used to learn a representation of each constraint 2) a *process* network that combines all constraints in a set into a single set representation, and 3) a *write* network that maps the set representation to an image. Below we describe each of these components.

The read network puts images within a constraint set through a Convolutional Neural Network (CNN) to extract visual features. Feature vectors of images from a common constraint pair are concatenated and input to a fully connected layer. The result is a single vector $\mathbf{c}_i$ for each constraint, which are collectively input to the process network.

The process network consists of a "processing unit" that is repeated p times. Let $\{\mathbf{c}_1, ..., \mathbf{c}_n\}$ be the output of the read network for a size n set of constraints. For each of the t repetitions of the processing unit, an iteration through an LSTM cell with "content-based" attention is performed:

$$\mathbf{q}_t = LSTM\left(\mathbf{z}, \mathbf{q}_{t-1}^*\right) \quad (2)$$

$$e_{i,t} = \mathbf{c}_i \cdot \mathbf{q}_t \quad (3)$$

$$a_{i,t} = \frac{\exp\left(e_{i,t}\right)}{\sum_j^n \exp\left(e_{j,t}\right)} \quad (4)$$

$$\mathbf{r}_t = \sum_i^n a_{i,t}\mathbf{c}_i \quad (5)$$

$$\mathbf{q}_t^* = [\mathbf{q}_t, \mathbf{r}_t] \quad (6)$$

First, $\mathbf{z}$ (as "input") and the hidden state from previous repetition are put through LSTM unit. The resultant hidden state output of the LSTM $\mathbf{q}_t$ is then combined with each $\mathbf{c}_i$ via dot product to create a scalar value $e_{i,t}$ for each constraint. These are used in a softmax function to obtain scalars $a_{i,t}$, which in turn are used in a weighted sum. This sum is the key operation that combines the constraints. Because addition is commutative, the result of (5), and thus the output of the processing network, is invariant to the order that the constraints were given. The result $\mathbf{r}_t$ is concatenated with $\mathbf{q}_t$ and is used as the input in the next processing iteration. After p steps, $\mathbf{q}_p^*$ is put through a fully connected layer to produce $\mathbf{s}$, which is input to the write network.

One way of interpreting this network is that each processing unit iteration refines the representation of the constraint set produced by the previous iteration. The output of the processing unit has two parts. First, $\mathbf{r}_t$ is a learned weighted average of the constraints, ideally emphasizing constraints with stronger signal. Second, $\mathbf{q}_t$ is the output of the LSTM which combines the noise vector and the output from the previous iteration, using various gates to retain certain features while removing others. These two components are sent back through the processing unit for further rounds of refinement.

Similar to the generator in the unconditional GAN framework, the write network maps a noise vector to image space. Motivated by this, we use the transpose convolutions [5, 25] utilized in Deep Convolutional GANs (DC-GANs) [23]. Transpose convolutions effectively learn an upsampling transformation. By building a network from transpose convolutional layers, our write network is able to learn how to map from a lower dimensional representation of constraint set to a higher dimensional image.

4. Empirical Evaluation

In order to evaluate CONGAN we aim to show its ability to satisfy constraints while achieving the image quality of similar WGAN models. Further, we wish to highlight some examples of how a user can interact with a CONGAN generator. To this end, we perform experiments with three data sets: MNIST [18], CelebA [36], and Zappos50k [37, 38].

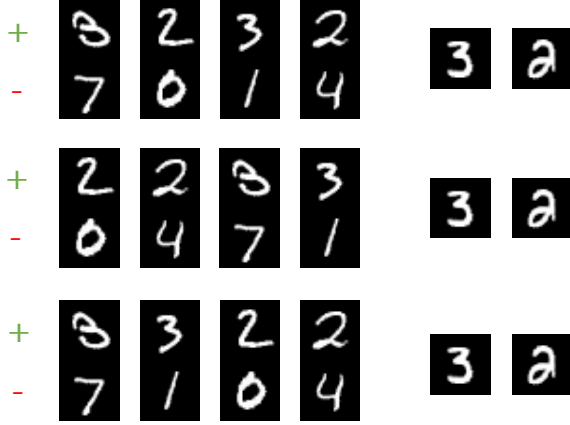In all experiments, we use the hyperparameters suggested

Figure 3: Example illustrating the order invariance property of CONGAN. On the left are relative constraints (top is positive image, bottom is negative) in the order they are input to the CONGAN generator. On the right are the images produced for two different **z** vectors. The output remains the same even when the constraints are given in different orders.

Figure 4: Examples from WGAN (left) and CONGAN (right) generators trained on the MNIST data set.

in [8]: ($\lambda = 10, n_{disc} = 5, \alpha = 0.0001, \beta = 0, \beta = 0.9$), follow Algorithm 1 from the same work to train WGAN, and set the batch size $m = 32$. We seed WGANs with noise vectors **z** drawn from a standard normal ($Z = \mathcal{N}(0, \mathbf{I})$), and CONGANs with a uniform distribution ($Z = \mathcal{U}(-1, 1)$). We opt to use the uniform distribution as it allows both inputs into the processing network to be in the same range. The noise vectors are of size 64 for the MNIST experiments and of size 128 for the CelebA and Zappos50k experiments. We set p (number of "processing" steps) to 5 in both experiments, but have observed that CONGAN is robust to this setting.

In [1] the authors observe that the Wasserstein Distance can be used to determine convergence. In our experiments, the Wasserstein Distance stopped improving by 100,000 generator update iterations for all models and use that as the iteration limit. We chose values for γ that were able to reduce the t-STE train error significantly while maintaining Wasserstein Distance close to what was achieved by the WGAN. To strike a good balance we set $\gamma = 10$ on MNIST, $\gamma = 250$ on CelebA, and $\gamma = 100$ on Zappos50k.

WGAN models were trained on the designated trained sets for MNIST and CelebA. For Zappos50k, we randomly chose 90% of the images as the train set, leaving the rest as test. Similarly, CONGAN model constraint sets $\mathcal{C}$ in the training set of constraint sets are created by first randomly choosing an image of the train set to be a reference image. Then, anywhere between 1 and 10 pairs of images are randomly chosen to be constraints. Next, ϕ is applied to the reference image and each pair. The resultant representations in $\mathcal{S}$ are used to determine which elements of the pairs are considered $\mathbf{X}_+$ (positive examples) and $\mathbf{X}_-$ (negative exam-
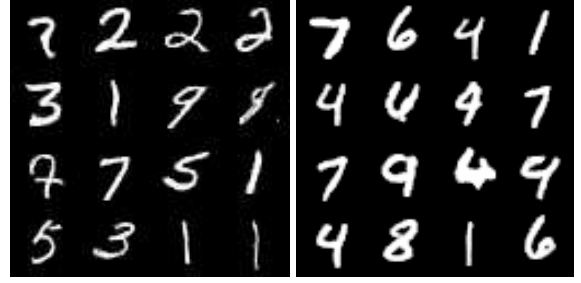
ples) according to (1). Test sets are constructed similarly.

The CONGAN network architectures used in these experiments are as follows [1]. For MNIST: The discriminator and read networks are five layer CNNs. The write network is a five layer transpose convolutional network. For CelebA and Zappos50k: The discriminator and read networks are residual networks [10] with four residual CNN blocks. The write network has four transpose convolutional residual blocks. To maintain some regularity between models in the interest of fair comparison, we use the same discriminator architectures for both WGAN and CONGAN and use the WGAN generator architecture as the CONGAN write network architecture. Other than a few special cases, we use rectified linear units as activation functions and perform layer normalization [2].

4.1. MNIST

MNIST is a well known data set containing 28x28 images of hand-written digits. For preprocessing we zero pad the images to 32x32 and scale them to [-1,1]. For ϕ we train a "mirrored" autoencoder on the MNIST train set using squared Euclidean loss. The encoder portion consists of four convolutional layers and a fully connected layer with no activation to a two-dimensional encoding. We use the encoder as ϕ. The decoder has a similar structure but uses transpose convolutions to reverse the mapping. Simply autoencoding MNIST digits reveals a loose class structure in the embedding space ($\mathcal{S}$ in this experiment). As such, this experiment shows how class relationships can be retrieved even if ϕ does not precisely map to classes.

We seek to evaluate the CONGAN's ability to satisfy given constraints. To this end, we constructed ten different test sets, each containing constraint sets of a fixed size. For example, each constraint set in the "2" test set has two constraints. We call an evaluation over a different test set an "experiment". In each experiment, we performed ten different trials where the generator was given different noise vectors per constraint set. With these experiments we can

_____

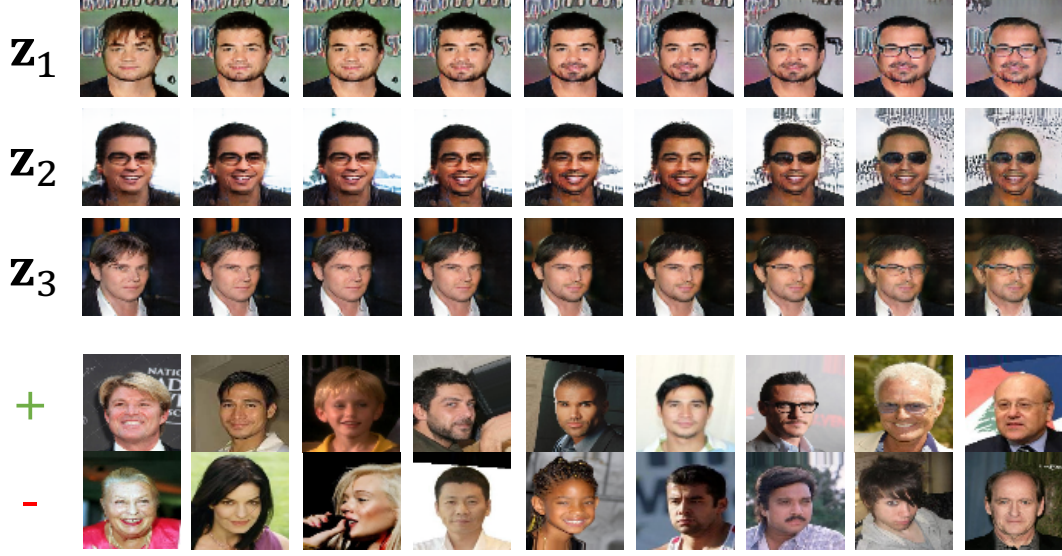[1]A more rigorous description can be found in the supplement.

Figure 5: Example of CONGAN generator outputs when trained on the CelebA data set. The bottom two rows of images are constraints, where the positive and negative images only differ by a single attribute. The first three constraints differ by only the "Male" attribute, the second three by only the "Beard" attribute, and the third three by only the "Eyeglasses" attribute. The top three rows are images produced from three different seeds when the constraints are provided to the CONGAN generator from left to right. For example, the third image in the first row is generated when z_1 and the first three constraints are given.
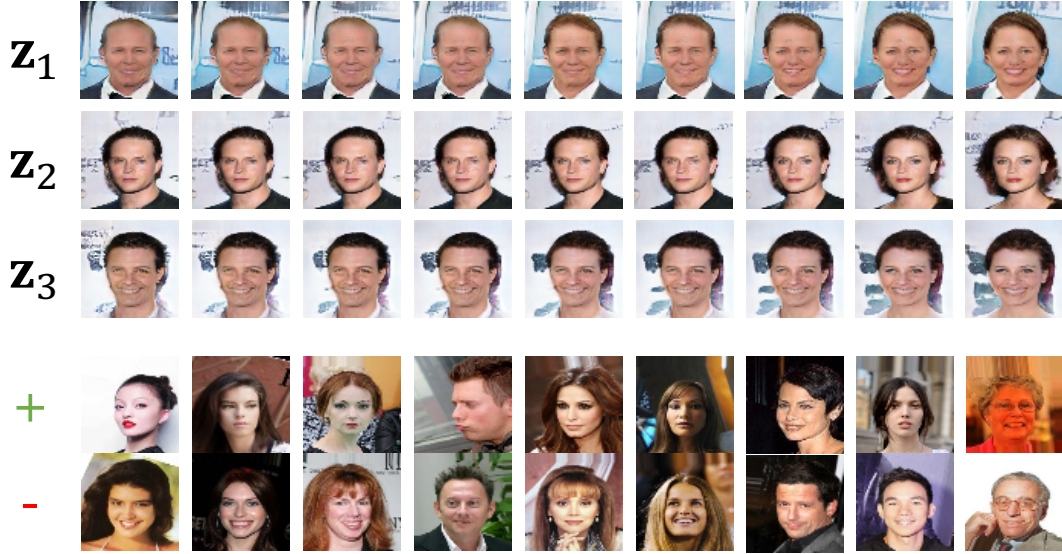

Figure 6: Another example of CONGAN generator outputs when trained on the CelebA data set. This is the same experiment as in Fig. 5, but with the attributes "Pale Skin", "Brown Hair", and "Female" from left to right.

observe the effect constraint set size has on the generator.

Results: Table 1 shows the mean constraint satisfaction error (i.e one minus the prevalence of (1)) of the CONGAN generator for each MNIST experiment. Overall, it was able to satisfy over 90% of given constraints. Note that the generator performs slightly better when more constraints are given. This is somewhat counter-intuitive. We believe that

in this case the generator is using constraints to determine what class of digit to produce. If given few constraints, it is more difficult for the generator to determine the class of the output. Figures 3 and 4 show example outputs of CONGAN when trained on MNIST: One showing the order invariance property of CONGAN and the other showing CONGAN generated images next to ones produced by a similar WGAN.

# input constraints									
1	2	3	4	5	6	7	8	9	10
0.0931	0.0895	0.0860	0.0831	0.0808	0.0784	0.0775	0.0756	0.0743	0.0733

Table 1: Mean constraint satisfaction errors of CONGAN on MNIST constraints per input set size (10 trials).

	WGAN	CONGAN (# input constraints)									
		1	2	3	4	5	6	7	8	9	10
-WGAN	20.31	18.32	18.90	19.34	19.64	19.82	19.93	20.00	19.99	19.96	19.90
-CONGAN	481.12	479.27	480.08	480.74	481.29	481.74	482.07	482.36	482.57	482.71	482.80
MCSE		0.0885	0.1047	0.1154	0.1202	0.1257	0.1279	0.1296	0.1307	0.1318	0.1325

	WGAN	CONGAN (# input constraints)									
		1	2	3	4	5	6	7	8	9	10
WGAN	60.31	44.85	46.03	46.99	47.45	47.30	46.76	45.84	44.86	43.81	42.78
CONGAN	5.43	-27.04	-18.64	-11.39	-5.53	-1.37	1.09	2.06	1.69	0.10	-2.41
MCSE		0.0950	0.0967	0.0974	0.1001	0.1009	0.1019	0.1052	0.1065	0.1065	0.1066

Table 2: Evaluation results on CelebA (top table) and Zappos50K (bottom table) data sets (10 trials). Rows 1-2 of each table: Mean discriminator scores for WGAN and CONGAN discriminators at convergence on WGAN and CONGAN generators (negative scores for CelebA). Row 3 of each table: Mean constraint satisfaction error of CONGAN models per input set size.

4.2. CelebA

The CelebA data set contains 202,599 color images of celebrity faces. For our experiments, we resize each image to 64x64 and scale to [-1,1]. Associated with each image are 40 binary attributes ranging from "Blond Hair" to "Smiling". We chose twelve of these attributes to be $\mathcal{S}$. More specifically, an image's representation in $\mathcal{S}$ is a binary vector of attributes, which differs from the MNIST experiment. In the previous experiment, $\mathcal{S}$ was both lower dimensional and continuous. As such, this experiment will evaluate CONGAN's ability to adapt to different semantic spaces.

For ϕ we construct a simple multi-task CNN (MCNN) [9] [2] that consists of one base network and multiple specialized networks, trained end-to-end. The base network accepts the image as input and extracts features for detecting all attributes. The specialized networks split from the base network and learn to detect to their predetermined subset. Our ϕ base network consists of two convolutional layers. The specialized networks (one for each of twelve attributes) consists of three convolutional layers followed by a fully connected layer that maps to a scalar attribute identifier.

For this experiment we sought to more objectively compare the WGAN generated images with those produced by CONGAN. To this end we first train a WGAN on the CelebA train set. Then, we initialize the CONGAN write network and discriminator to the trained WGAN generator and discriminator, respectively, before training the CONGAN generator. By doing this, we can observe how image quality is

affected by adding the CONGAN components to a WGAN.

Results: Rows one and two of Table 2 (top table) show the mean negative discriminator scores for both the WGAN and CONGAN generators against the WGAN and CONGAN discriminators at convergence over ten trials. We can see that for both discriminators, WGAN generated images are scored very similarly to those generated by CONGAN. This is especially true when considering the standard deviation for the WGAN generator against the WGAN and CONGAN discriminators is 8.75 and 16.22, respectively, and slightly higher on both for the CONGAN generator. We believe this result shows evidence that adding the CONGAN framework to the WGAN training did not drastically alter image quality.

The last row of Table 2 shows the mean constraint satisfaction error on the test set for each experiment. Here, the CONGAN generator is able to satisfy around 87% or more of the constraints. Figures 5 and 6 show images generated by CONGAN. As constraints are provided, the image produced from different seeds take on the attributes indicated by the constraints. In Fig. 5, the first three constraints indicate the "Male" attribute, the next three indicate "Beard", and the last "Eyeglasses". In Fig. 6, "Pale Skin", "Brown Hair", and "Female" are indicated. These examples show that a user can iteratively refine the images to have desired characteristics, and still be given a variety of realistic, novel images.

4.3. Zappos50K

The Zappos50K data set contains 50,025 color images of shoes. We resize each image to 64x64 and scale to [-

---

[2] Details and an evaluation of the MCNN can be found in the supplement.

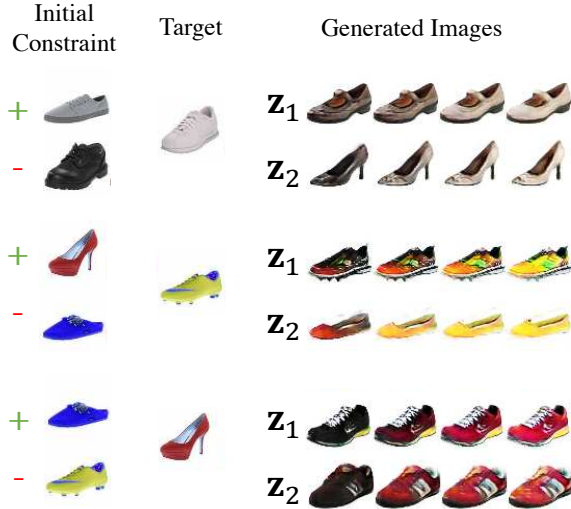|	Initial Constraint	Target	Generated Images

Figure 7: Three sets of two examples from the CONGAN generator trained on the Zappos data set. The generator was first provided the initial constraint of the left, generating the first (left-most) image in the generated images column. To generate each of the next three images, the generator was fed a constraint where the positive image was the target image, and the negative image was the previously generated image.

1,1]. For this experiment, we chose $\mathcal{S}$ to be a color space. To accomplish this, we computed a 64 bin color histogram over each image and trained a nine-layer CNN to embed the images in 2-dimensions using a triplet network [11] [3], and used this as ϕ. We opted to use the T-STE loss in the objective as it produced a clear separation of colors.

There is inherent bias in the Zappos50K data set when it comes to color, as most shoes tend to be black, brown, or white. This poses a challenge in that if constraint sets used for training are formed by uniformly sampling over the train set, the model will tend to favor few colors, making it difficult to guide generation to other colors. To combat this, we constructed constraints to include a more uniform sampling over colors. When constructing the train set of constraint sets, with probability 0.5 we uniformly sampled over training images as in the other experiments. When not sampling uniformly, we focused on a single color bin by first selecting a bin and choosing all positive images in the constraint set to be images where the highest histogram value corresponded to that bin (e.g. all positive examples would be "light blue"). Negative examples would be chose uniformly from the other bins. We found this allowed the CONGAN generator to more easily learn to produce a variety of colors.

Results: Table 2 (bottom table) shows the discriminator scores and mean constraint satisfaction errors for each Zappos50K experiment. Here, the CONGAN generator pro-

duced lower scores than the WGAN for both discriminators, though within one standard deviation. We believe this is due to training the generator to produce a wider variety of colors. If training data contains many brown, black, and white shoes, then training the generator to produce blue, red and yellow shoes will force it to produce images that differ than those provided to the discriminator. Nevertheless, we believe that image quality was only slightly degraded as a result.

Figure 7 shows examples of the images produced by the CONGAN generator. Here, we wanted to test the use case of providing single images, instead of pair-wise constraints, to guide the generator to a result. An initial constraint is provided to produce a starting images. After that, a single target image is used repeatedly as the positive example to generate shoes more similarly colored to the target.

5. Conclusion and Future Work

In this work, we introduce a Generative Adversarial Network framework that is able to generate imagery guided by iterative human feedback. Our model relies on two novel components. First, we develop a generator, based on recent work in memory networks, that maps variable-sized sets of constraints to image space using order-invariant operations. Second, this generator is informed during training by a critic that determines whether generated imagery satisfies given constraints. The result is a generator that can can be guided interactively by humans through relative constraints. Empirically our model is able to generate images that are of comparable quality to those produced by similar GAN models, while satisfying a up to 90% of given constraints.

There are multiple avenues of future work that we believe are worthy of further study. First, it may not be feasible for users of CONGAN to search through large image databases to find the exact constraints they desire. We will apply pairwise active ranking techniques [12] to suggest constraint queries in order to quickly constrain the semantic space without requiring users to search through images themselves. Second, we will investigate the output of the process network more closely seeing if constraint representations have properties that match intuition about how sets of constraints are classically reasoned about, similar to word embeddings [20].

---

[3]A visualization of this embedding can be found in the supplement.

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017.

[2] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *NIPS Deep Learning Symposium*, 2016.

[3] S. Branson, C. Wah, F. Schroff, B. Babenko, P. Welinder, P. Perona, and S. Belongie. Visual recognition with humans in the loop. *ECCV*, 2010.

[4] K. Duan, D. Parikh, D. Crandall, and K. Grauman. Discovering localized attributes for fine-grained recognition. In *CVPR*, 2012.

[5] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014.

[7] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[8] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

[9] E. M. Hand and R. Chellappa. Attributes for improved attributes: A multi-task network utilizing implicit and explicit relationships for facial attribute classification. In *AAAI*, 2017.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[11] E. Hoffer and N. Ailon. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, 2015.

[12] K. G. Jamieson and R. Nowak. Active ranking using pairwise comparisons. In *NIPS*, 2011.

[13] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *ICLR*, 2018.

[14] M. G. Kendall and J. D. Gibbons. Rank correlation methods. 1990.

[15] D. P. Kingma and M. Welling. Auto-encoding variational bayes. 2014.

[16] A. Kovashka, D. Parikh, and K. Grauman. Whittlesearch: Image search with relative attribute feedback. In *CVPR*, 2012.

[17] N. Kumar, P. N. Belhumeur, A. Biswas, D. W. Jacobs, W. J. Kress, I. C. Lopez, and J. V. Soares. Leafsnap: A computer vision system for automatic plant species identification. In *ECCV*. 2012.

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[19] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *JMLR*, 2008.

[20] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[21] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[22] G. Perarnau, J. van de Weijer, B. Raducanu, and J. M. Álvarez. Invertible conditional gans for image editing. *arXiv preprint arXiv:1611.06355*, 2016.

[23] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *ICLR*, 2016.

[24] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. In *ICML*, 2016.

[25] W. Shi, J. Caballero, L. Theis, F. Huszar, A. Aitken, C. Ledig, and Z. Wang. Is the deconvolution layer the same as a convolutional layer? *arXiv preprint arXiv:1609.07009*, 2016.

[26] N. Stewart, G. D. Brown, and N. Chater. Absolute identification by relative judgment. *Psychological review*, 112(4):881, 2005.

[27] S. Sukhbaatar, J. Weston, R. Fergus, et al. End-to-end memory networks. In *NIPS*, 2015.

[28] A. van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al. Conditional image generation with pixelcnn decoders. In *NIPS*, 2016.

[29] L. Van Der Maaten and K. Weinberger. Stochastic triplet embedding. In *MLSP*, 2012.

[30] A. Van Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *ICML*, 2016.

[31] O. Vinyals, S. Bengio, and M. Kudlur. Order matters: Sequence to sequence for sets. In *ICLR*, 2016.

[32] C. Wah, S. Branson, P. Perona, and S. Belongie. Multiclass recognition and part localization with humans in the loop. In *ICCV*, 2011.

[33] C. Wah, G. Van Horn, S. Branson, S. Maji, P. Perona, and S. Belongie. Similarity comparisons for interactive fine-grained categorization. In *CVPR*, 2014.

[34] J. Weston, S. Chopra, and A. Bordes. Memory networks. In *ICLR*, 2015.

[35] X. Yan, J. Yang, K. Sohn, and H. Lee. Attribute2image: Conditional image generation from visual attributes. In *ECCV*, 2016.

[36] S. Yang, P. Luo, C.-C. Loy, and X. Tang. From facial parts responses to face detection: A deep learning approach. In *ICCV*, 2015.

[37] A. Yu and K. Grauman. Fine-grained visual comparisons with local learning. In *CVPR*, 2014.

[38] A. Yu and K. Grauman. Semantic jitter: Dense supervision for visual comparisons via synthetic images. In *ICCV*. IEEE, 2017.

[39] J. Zhao, M. Mathieu, and Y. LeCun. Energy-based generative adversarial network. *ICLR*, 2017.

[40] J.-Y. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros. Generative visual manipulation on the natural image manifold. In *ECCV*. Springer, 2016.