Criminal Sketch Net

Alex Richards*, Mina Yi†
Charles W. Davidson College of Engineering
San José State University
1 Washington San José, CA 95192
*alex.richards006@gmail.com, †dearmeenuh@gmail.com

***Abstract*—Accompanying the development of graphics processing units (GPUs) is a tremendous amount of excitement surrounding the field of deep learning [1] [2]. Despite all of this attention, techniques for training deep neural network models are still in their infancy. Perhaps the most exciting of these techniques, aside from the class of methods associated with deep reinforcement learning (e.g. Q-learning [3] and model-based learning) are generative adversarial networks (GANs) first developed by Ian Goodfellow [4]. While impressive when executed properly, GANs are known to be quite difficult to train at times. In an effort to make training them more manageable and to produce higher quality samples with the converged models, alterations to the specifics of GAN training is being heavily researched. One advancement that has been shown to improve the results of GANs with respect to these concerns is conditional GAN training [5]. Conditional GANs provide controlled modality for the traditional GAN model by introducing an additional input layer to both the generator and discriminator. One of the state-of-the-art conditional GAN models is StackGAN [6] which, as the name implies, is an architecture which stacks GANs to increase the resolution of generated images and uses conditioning augmentation to overcome the tendency for discontinuous latent variable manifolds being created between the text embedding (the conditioning data) and the learned parameters of the generator. In this paper, we use a simplified version of StackGAN to condition images of human faces on text attributes to create a model which can generate realistic renderings of people based on their descriptions similar to the way a criminal sketch artist would. In addition, we explore alterations to the StackGAN architecture to minimize training time.**

Keywords*—*Text-to-image synthesis, StackGAN, generative adversarial networks, generative models, conditional GANs

I. Introduction

Generative modeling is an interesting problem in statistics and probability that involves the generation of inputs and outputs from hidden parameters. It is counter to discriminative modeling which only generates outputs from inputs. In a machine learning setting, generative models learn a probability density function of their observed inputs. There is a well developed body of knowledge surrounding generative models; some popular ones include Gaussian mixture models, auto-encoders, restricted Boltzmann machines (RBMs), and, most recently, GANs. All of these models have their own advantages and disadvantages. Auto-encoders, while very efficient at learning the manifold on which the training data exists, may have drastically different performance when attempting to reconstruct inputs that deviate from this manifold. Variational auto-encoders (VAEs) mitigate this instability by introducing noise (often Gaussian), but at the cost of generating blurred samples. RBMs are similar to multilayer perceptrons only, instead of learning parameters that minimize the error between target values and the output of the network, they propagate activation signals back to the inputs in a reconstruction phase and the difference of the reconstructed input and the original input is used as an error signal when learning the network's parameters. However, RBMs can be difficult to train. The error computing mechanism, Contrastive Divergence aims to minimize the non-overlapping parts of the true and reconstructed probabilities; this involves the use of a partition function whose gradient solution is intractable. As a result, learning the RBM's parameters requires the use of Monte Carlo Markov methods which can be challenging to implement correctly.

GANs aim to mitigate the challenges of the aforementioned generative models via a novel training method. GAN architectures are "adversarial" because they employ a loss function that equates to a zero-sum game between two component networks; a generator that produces data from a noise vector, and a discriminator that classifies instances as real data or generated data. The discriminator network's goal is to maximize the probability of assigning the correct label to both real data and data generated by the generator network while the generator's goal is to fool the discriminator into classifying its own generated data as being real.

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

Fig. 1. The simplified zero-sum mimmax value function used in GAN architectures. [4]

Let us now break down the value function shown in Figure 1. $D(x)$ represents the probability that the discriminator of the GAN correctly assigns a label $real$ to the real images of the training set; thus, the discriminator aims to maximizes this value to 1 while the generator wishes to minimize this value to 0. $D(G(z))$ represents the probability that the discriminator incorrectly assigns a label $real$ to the generated images sampled from the generator during training; thus, the discriminator will learn to minimize this value to 0 while the generator seeks to maximize this value to 1. It should be noted that, at the beginning of training, the discriminator will be able to trivially distinguish between real data and the noise sampled from the generator. As such, the gradients of the generator will be very small. To mitigate this, implementations of the value function have the generator maximize $log(D(G(z)))$ rather than minimzie $log(1 - D(G(z)))$ [4]. In practice, the system reaches global optimality at the midpoint of these probability outputs. In otherwords, the model stabilizes once the probability of the discriminator correctly classifying a $real$

or $fake$ data sample is 0.5. Once this point has been reached, the generator has learned the probability density function of the observed data allowing it to produce realistic samples of the training data. With GANs, samples are generated in one shot and are asymptotically consistent meaning they have no bias, unlike VAEs. Also, GANs do not rely on any of the probabilistic approximation mechanisms that RBMs and many other generative models use. The cons of GANs include the difficulty of generating discrete outputs with them and their dependence on a Nash equilibrium, a game theory solution concept which involves two or more players who know the equilibirum strategy of all other players yet no player has anything to gain by changing their own strategy [7].

Generating realistic samples has many useful applications; completing missing data, converting between formats, style transfer, and increasing image resolution are just a few examples. In this paper, we investigate text-to-image synthesis, an application of conditional GANs, which has many potential uses. Generating images from text can be used to manipulate photos by textual description rather than through the use of the traditional photo editing toolset. Drafting character models for animations, video games, or concept art would be made much faster. Furthermore, as speech-to-text models continue to improve and become more ubiquitous, text-to-image synthesis can easily extend the utility of any speech-based system. The domain that our research focus on is the generation of faces from textual descriptions of facial features. One application of this is the generation of criminal suspect sketches from eyewitness accounts. Our challenges in achieving a model capable of this application include preprocessing a cropped version of the Labeled Faces in the Wild dataset [8], provided by Sanderson and Lovell [9], adapting a TensorFlow [10] implementation of StackGAN produced by Hao Dong [11] to accept this dataset, and altering the architecture of this model in order to expedite the training process.

II. Related Works

Existing models used to tackle text-to-image synthesis generally rely on the image generation capabilities of GANs and text embeddings learned through either character level convolutional network or recurrent neural network architectures. Conditioned GANs learn joint embeddings by using, in this case, text data as input for both the generator and discriminator components of the model. This additional input layer enables the discriminator to minimize further, possibly allowing for the use of a lower value of k (the number of discriminator training iterations before switching to training the generator). For the generator, the additional input layer allows for controlled modality when sampling [5]. This addition of controlled modality is what enables a conditional GAN to output samples that fall under a particular class rather than producing a typically Gaussian distribution of samples indepedent of any sort of mode.

The first component technology, the image generation processing, has been explored extensively. Work by Denton [12] and Dosovitskiy [13] incorporate the use of multiple "deconvolutional" decoders which effectively upsample lower resolution images in a sort of inverse pooling operation to produce higher resolution images without compromising on realism. Yang, Reed, Yang, and Lee extended the work of Denton by prefixing deconvolutional layers with auto-encoders to permit the transformation of image inputs [14]. This research demonstrated that it is possible to leverage the reduced dimensionality of input images to transform them into high resolution images upsampled from this lower dimensional representation.

The second component technology, the joint embedding of conditional (particularly text) data, also has a diverse history of research supporting it. Researchers like Vinyals, Toshev, Bengio, Erhan Karpathy, and Fei began this body of work by modeling the inverse of the problem we are focusing on in this paper; conditioning the generation of text descriptions from image inputs [15] [16]. Others, like Kiros, Salkhutdinov, and Zemel, focused on information retrieval applications which use the multi-modal text-image embedding space to retrieve images from a database using sentence queries; in addition, they propose a joint embedding model for constructing textual descriptions from image queries as well. Another improvement that is indirectly utilized in the model used for this paper is latent manifold interpolation. By interpolating the text embeddings, we can disincentivize the discriminator from learning relationships between image and text pairings [17]. In the StackGAN architecture [6], multiple text descriptions are given for each image and these descriptions are randomly drawn from during the sampling phase of training.

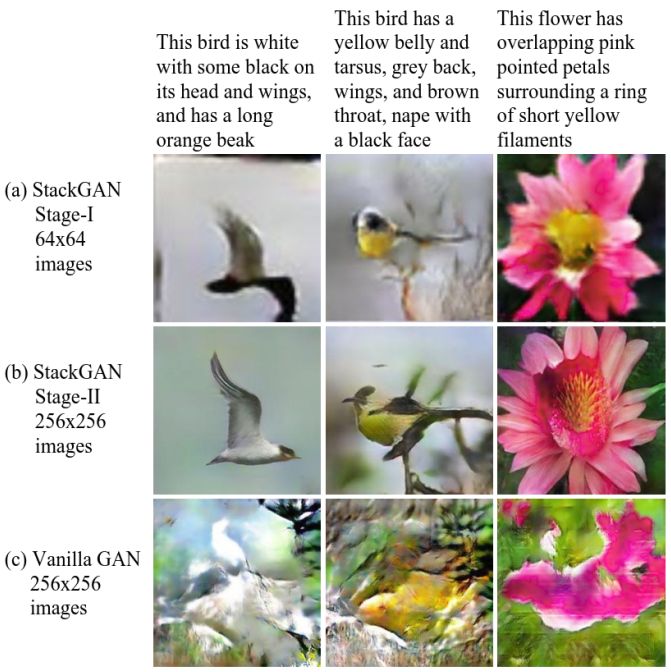

Fig. 2. Comparison of images generated with StackGAN phases I and II and also simple one-shot upsampling of a generator. [6]

The most relevant work, of course, is the contribution of StackGAN by Zhang et al [6]. StackGAN is capable of generating much higher resolution photos in which small details of the subjects are generated that are otherwise not visible in the aforementioned works. For example, StackGan trained on the Caltech-UCSD 200 dataset generates birds that have beak detail and eyes, as can be seen in figure 2. It is able to do this because it decomposes the task of upsampling generated images into two phases. In phase I, StackGAN constructs low-resolution detail in 64×64 images. Phase II of StackGAN

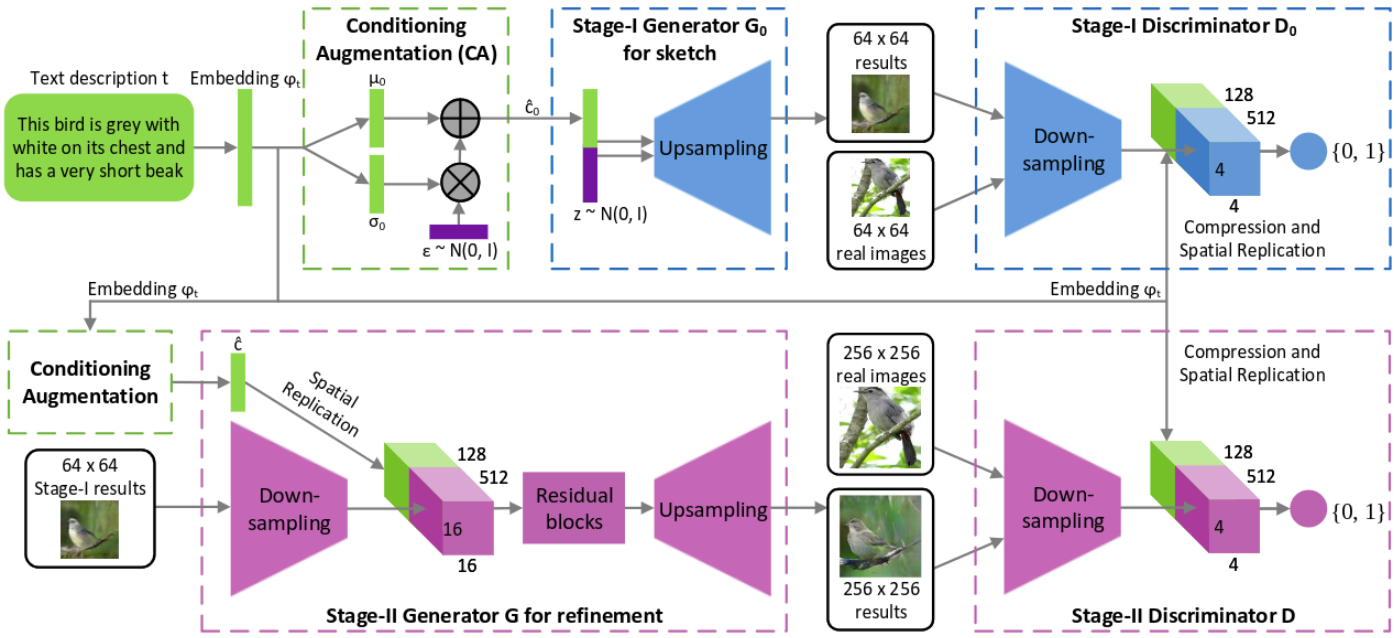

Fig. 3. Illustration of StackGAN's two phase architecture. [6]

upsamples while conditioning on the text data again and also the phase I images in order to capture more information from the text embeddings and correct defects in the first layer's output. The intuition behind StackGAN's design is that the model is able to better learn an overlapping distribution, in high-dimensional pixel space when the joint embedding formed from the conditional data can be fully utilized after already producing such an overlap in a lower dimensional space. The phase II architecture begins by downsampling the phase I output until it has a dimensionality such that it can be concatenated along the channels axis with the text embedding. This concatenated feature space is then fed through residual blocks which employ a technique found by He et al. [18] which combats the problem of vanishing gradients in deep networks by reintroducting the first input layer. These residual layers are what permit the learning of the data distribution through high-dimensional space. Finally, phase II concludes with several upsampling layers to produce the final 256×256 image. A view of this two phase GAN architecture can be seen in figure 3.

III. Methods

A. Materials

Our model was fit using images from the LFWCrop Face Dataset provided by Conrad Sanderson [9]. We opted to use the cropped version of Labeled Faces in the Wild [8] because the background noise of the original dataset would likely introduce difficulties with synthesizing the desired features (face characteristics). To condition the image synthesis process, we created sentence-like configurations of facial features using the LFW attributes file [19] which includes numerical values for each of 72 different facial characteristics for many of the images in LFWCrop. The base model used for producing our results was a TensorFlow implementation of StackGAN phase I [6] constructed by Hao Dong [11].

B. Procedure

We began by ensuring that all of the images in the LFWCrop Face Dataset has a corollary row in the LFW attributes file. This was accomplished by writing a Python 3.5.5 script $create\_face\_data.py$ which removed any row from the LFW attributes file whose name did not match any file name in the LFWCrop Face Dataset. Similarly, the script removed any image whose file name did not match the name of any person among the rows of the LFW attributes file. Originally, the LFWCrop Face Dataset contained $13,233$ images and the LFW attributes file contained $13,143$ rows. After executing the data cleaning script, both the attributes file and the set of images contained $13,040$ images. Additionally, $create\_face\_data.py$ removed the column "Wearing Necklace" from the LFW attributes file because it contained Nan values. Also, since we want our model to explicitly recognize the text attribute of "female" during inference time, we inserted a column titled "Female" whose values are the negative of the numerical value of the existing column "Male" for each row.

The implementation of StackGAN [11] that we are using leverages three NumPy arrays to aid in the tokenization of text inputs prior to their insertion into the recurrent neural network used to create the text embedding. These arrays include a set of the vocabulary contained within the text model, a term-to-ID mapping and an ID-to-term mapping. The second step of our data preprocessing was to recreate these three arrays using the columns of the LFW attributes file. In addition to the vocabulary terms formed by the columns of LFW attributes, we retained the meta terms $< PAD >$ and $< RARE >$ which are used, respectively, to ensure that text inputs to the recurrent neural network with fewer than 20 words are padded and words not appearing in the vocabulary are handled properly.

The next stage of data preprocessing was the creation of captions for our face images. For every row in the LFW attributes file, we sorted its columns in non-increasing order

of their numerical values. We then selected the first 20 column titles as words to include in the captions for the image. Since, StackGAN implements latent manifold interpolation, we had to create variations of these captions so that the discriminator would not learn the mapping of a specific text caption to its corresponding image. We opted to create ten variations of captions for each image by shuffling the top 20 attributes. Lastly, we converted the $.ppm$ images in the LFWCrop Face Dataset into NumPy arrays in order to be compatible with this implementation of StackGAN.

After reaching 100 iterations of training, we observed that the samples output by the generator had particular characteristics that could be used to inform and further improve data preprocessing. Namely, we saw that, since mostly none of the cropped images included features like hair or earrings, there was no benefit to including any hair related features in the captions. Similarly, other image attributes were not relevant given that the text queries would not include things like "outdoors" or "harsh lighting". Another key insight was that the choice to use 20 features resulted in selecting perhaps too many descriptors. Manually constructing sample text queries made it apparent that it is difficult to even come up with 20 high-level descriptors of a criminal suspect's face. Given this, we produced a revised set of vocabulary indexing arrays and training captions that excluded the terms in table I and used only 10 features rather than 20.

TABLE I. FEATURES EXCLUDED FROM REVISED CAPTION SET

Feature
Wearing Necklace
Wearing Earrings
Wearing Necktie
Black Hair
Blond Hair
Brown Hair
Gray Hair
Curly Hair
Wavy Hair
Straight Hair
Outdoor
Receding Hairline
Wearing Hat
Bangs
Bald
Harsh Lighting
Soft Lighting
Flash

Due to time constraints, we opted to focus on training a simplified StackGAN model which only included the phase I architecture. While training the complete StackGAN model would have yielded much higher resolution images with more realistic detailing, doing so would have required training four networks instead of two. If we had attempted to use the full two phase StackGAN implementation, the hyperparameter space, relative to the space we dealt with, would have been squared. By limiting the number of hyperparameters we were better able to scan the hyperparameter space in order to find ways to decrease training time while minimizing the amount of detail lost in the generated images.

Phase I of StackGAN utilizes a generator, discriminator, CNN encoder, and a RNN encoder. For a full description of the generator, discriminator, and encoder architectures, please reference tables II, III, and IV respectively. Note that the first dimension in the layer shape descriptions, 64, is the batch size. The RNN encoder includes a single long-short-term-memory (LSTM) cell which had 128 hidden units in it. This LSTM cell unrolls along the maximal length input, which is 53 in the case of our reduced vocabulary dataset, to capture the time axis. Finally, a dynamic RNN is created using this LSTM cell. The RNN is provided with a zero initialized state and the word embedding IDs so that it may produce the outputs of shape $[batch\_size,\ hidden\_neurons] = [64,\ 128]$.

The hyperparameters we tuned between the seven training runs we performed include the learning rate α, the decay rate of α, and the dimensionality of the word embedding created by the recurrent neural network. Initially, we trained the model with a learning rate of 0.0002, a learning rate decay of 0.5, and a word embedding size of 256. As mentioned above, without the second phase GAN architecture in place, creating very high-detail and realistic faces was not possible. However, we noticed that there was room for improvement regarding the stability of the conditioned data's effect on the images sampled from the generator. Neither the generator nor the discriminator of our model had monotonically decreasing losses so we adjusted the learning rate to 0.0001 and increased the learning rate decay value to 0.6 in hopes that the observed loss of the components of our model would discontinue bouncing around a possible optimum. However, we found that multiple successive tweaks to these two parameters had minimal effect on controlling the sporadic behavior of our observed loss.

After this, we decided to focus our efforts on controlling the interaction of the word embedding with the rest of the model. We hypothesized that, given the dramatic decrease in the vocabularly size of the LFW dataset over the CalTech-UCSD 200 Birds dataset, the default word embedding size of 256 was likely much too sparse for properly conditioning the image generation process. On successive training runs, we changed the word embedding size to 128 and 53. Initially, we wanted to observe the affect of halving the word embedding size. We had expected a smaller word embedding size to lead to more consistent generation of the features indicated in the text conditioning data. Although we did see a stablizing in the generated faces across the many iterations of a given training run, we found that the images and fewer well defined features all together. Additionally, we attempted training with a word embedding size of 53 as this matched the number of unique words in a vocabularly after removing some of the hair and background related features which were not relevant. These features were deemed irrelevant because the cropped LFW dataset did not show the hair of subjects or their surroundings. However, further decreasing the word embedding size showed similar issues with producing more defined features in the sampled images.

IV. RESULTS

Assessing the multiple variations of our model proved to be quite challenging. Heuristics that can be utilized in simpler models for informing the tuning of hyperparameters were not nearly as effective in our GAN architecture. For instance, discovering a relationship between the loss function and the learning rate or learning rate decay value was not

TABLE II. GENERATOR ARCHITECTURE

Layer	In	Out	Kernel (H x W)	Kernel Channels	Activation	Padding	Stride (H x W)
FC (rnn_fc)	[64, 256]	[64, 128]			Leaky ReLU		
Concat (noise + word_embedding)	([64, 512] + [64, 128])	[64, 640]					
FC (g_h0/fc)	[64, 640]	[64, 16384]					
Batch Norm							
Reshape	[64, 16384]	[64, 4, 4, 1024]					
Conv2D (g1_h1_res/conv2d)	[64, 4, 4, 1024]	[64, 4, 4, 256]	[1, 1]	256		VALID [0, 0]	[1, 1]
Batch Norm					ReLU		
Conv2D (g1_h1_res/conv2d2)	[64, 4, 4, 256]	[64, 4, 4, 256]	[3, 3]	256		SAME [1, 1]	[1, 1]
Batch Norm					ReLU		
Conv2D (g1_h1_res/conv2d3)	[64, 4, 4, 256]	[64, 4, 4, 1024]	[3, 3]	1024		SAME [1, 1]	[1, 1]
Batch Norm (net)							
Add (net_h0 + net) (g_h1_res/add)	[64, 4, 4, 1024]	[64, 4, 4, 1024]			ReLU		
UpSample (g_h2/upsample2d)	[64, 4, 4, 1024]	[64, 8, 8, 1024]			Nearest Neighbors		
Conv2D (g_h2/conv2d)	[64, 8, 8, 1024]	[64, 8, 8, 512]	[3, 3]	512		SAME [1, 1]	[1, 1]
Batch Norm							
Conv2D (g_h3_res/conv2d)	[64, 8, 8, 512]	[64, 8, 8, 128]	[1, 1]	128		VALID [0, 0]	[1, 1]
Batch Norm					ReLU		
Conv2D (g_h3_res/conv2d2)	[64, 8, 8, 128]	[64, 8, 8, 128]	[3, 3]	128		SAME [1, 1]	[1, 1]
Batch Norm					ReLU		
Conv2D (g_h3_res/conv2d3)	[64, 8, 8, 128]	[64, 8, 8, 512]	[3, 3]	512		SAME [1, 1]	[1, 1]
Batch Norm							
Add (g_h3/add)	[64, 8, 8, 512]	[64, 8, 8, 512]			ReLU		
UpSample (g_h4/upsample2d)	[64, 8, 8, 512]	[64, 16, 16, 512]			Nearest Neighbors		
Conv2D (g_h4/conv2d)	[64, 16, 16, 512]	[64, 16, 16, 256]	[3, 3]	256		SAME [1, 1]	[1, 1]
Batch Norm					ReLU		
UpSample (g_h5/upsample2d)	[64, 16, 16, 256]	[64, 32, 32, 256]			Nearest Neighbors		
Conv2D (g_h5/conv2d)	[64, 32, 32, 256]	[64, 32, 32, 128]	[3, 3]	128		SAME [1, 1]	[1, 1]
Batch Norm					ReLU		
UpSample (g_h0/upsample2d)	[64, 32, 32, 128]	[64, 64, 64, 128]			Nearest Neighbors		
Conv2D (g_h0/conv2d)	[64, 64, 64, 128]	[64, 64, 64, 3]	[3, 3]	3	Tanh	SAME [1, 1]	[1, 1]

TABLE III. DISCRIMINATOR ARCHITECTURE

Layer	In	Out	Kernel (H x W)	Kernel Channels	Activation	Padding	Stride (H x W)
Conv2D (d_h0)	[64, 64, 64, 3]	[64, 32, 32, 64]	[4, 4]	64	Leaky ReLU	SAME [1, 1]	[2, 2]
Conv2D (d_h1)	[64, 32, 32, 64]	[64, 16, 16, 128]	[4, 4]	128		SAME [1, 1]	[2, 2]
Batch Norm					Leaky ReLU		
Conv2D (d_h2)	[64, 16, 16, 128]	[64, 8, 8, 256]	[4, 4]	256		SAME [1, 1]	[2, 2]
Batch Norm					Leaky ReLU		
Conv2D (d_h3)	[64, 8, 8, 256]	[64, 4, 4, 512]	[4, 4]	512		SAME [1, 1]	[2, 2]
Batch Norm							
Conv2D (dh4_res/conv2d)	[64, 4, 4, 512]	[64, 4, 4, 128]	[1, 1]	128		VALID [0, 0]	[1, 1]
Batch Norm					Leaky ReLU		
Conv2D (dh4_res/conv2d2)	[64, 4, 4, 128]	[64, 4, 4, 128]	[3, 3]	128		SAME [1, 1]	[1, 1]
Batch Norm					Leaky ReLU		
Conv2D (dh4_res/conv2d3)	[64, 4, 4, 128]	[64, 4, 4, 512]	[3, 3]	512		SAME [1, 1]	[1, 1]
Batch Norm							
Add	[64, 4, 4, 512]	[64, 4, 4, 512]			Leaky ReLU		
FC (RNN_text)	[64, 256]	[64, 128]			Leaky ReLU		
Expand (d_txt/expanddim1)	[64, 128]	[64, 1, 128]					
Expand (d_txt/expanddim2)	[64, 1, 128]	[64, 1, 1, 128]					
Tile (d_txt/tile)	[64, 1, 1, 128]	[64, 4, 4, 128]					
Concat (tile + dh4)	([64, 4, 4, 512] + [64, 4, 4, 128])	[64, 4, 4, 640]					
Conv2D (d_h3/conv2d_2)	[64, 4, 4, 640]	[64, 4, 4, 512]	[1, 1]	512		VALID [0, 0]	[1, 1]
Batch Norm					Leaky ReLU		
Conv2D (d_h0/conv2d)	[64, 4, 4, 512]	[64, 1, 1]	[4, 4]	1	Sigmoid	VALID [0, 0]	[4, 4]

possible when studying the behavior of the four variants of the model that aimed to optimize this. However, the three model variants that sought to optimize the word embedding size showed us that the generator loss tended to be less when using a larger word embedding size despite the significantly smaller vocabulary of the LFW attributes when compared with the CalTech-UCSD Birds dataset that the model's default hyperparameters were set for.

Due to our inability to train the significantly larger two stage StackGAN implementation [6], we were unable to produce the high-resolution (256×256) images that Zhang et al. created during their research. Our single stage StackGAN implementation's generator caputures the distribution of observed data best when the output image size is capped at 64×64. The results of our best model, seen in figure 4, showed reasonable detail and contained the highest variance of facial features when compared with all the other models.

Additionally, we believe that our technique for shuffling the word order in our pseudo-captions was not an effective means for interpolating the latent manifold used by the generator to condition the sample generation process. In the future, we'd like to explore ways to emulate organic sentence construction using synonym lookups.

A final point to make about our results versus those put out in the StackGAN paper is that we were only able to train each variation of our model for 100-200 iterations whereas the best models produced by Zhang et al. were trained using 800 iterations. Given that our priority was to produce many variants of the base model in order to compare them, we could

TABLE IV. CNN ENCODER ARCHITECTURE

Layer	In	Out	Kernel (H x W)	Kernel Channels	Activation	Padding	Stride (H x W)
Conv2d (cnnf/h0/conv2d)	[64, 64, 64, 3]	[64, 32, 32, 64]	[4, 4]	64	Leaky ReLU	SAME [1, 1]	[2, 2]
Conv2d (cnnf/h1/conv2d)	[64, 32, 32, 64]	[64, 16, 16, 128]	[4, 4]	128		SAME [1, 1]	[2, 2]
Batch Norm Conv2d (cnnf/h2/conv2d)	[64, 16, 16, 128]	[64, 8, 8, 256]	[4, 4]	256	Leaky ReLU	SAME [1, 1]	[2, 2]
Batch Norm Conv2d (cnnf/h3/conv2d)	[64, 8, 8, 256]	[64, 4, 4, 512]	[4, 4]	512	Leaky ReLU	SAME [1, 1]	[2, 2]
Batch Norm					Leaky ReLU		
Flatten (cnnf/h4/flatten)	[64, 4, 4, 512]	[64, 8192]					
FC (cnnf/h4/embed)	[64, 8192]	[64, 128]					

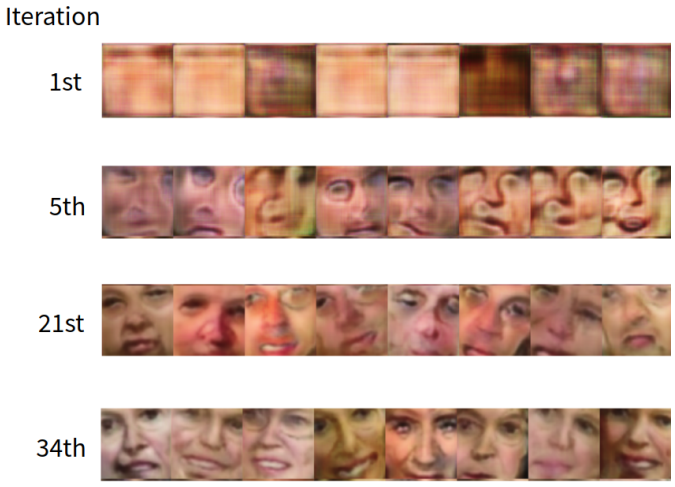

Fig. 4. Samples generated by our best performing model

not commit that much training time for a given model.

V. FUTURE WORK

As is the case with many-parameter models, especially neural networks, more data would be helpful for reducing overfitting of the discriminator and increasing the variance of the probability density function learned by the generator. In particular, a face dataset with well formed text captions, similar to the Caltech-UCSD Birds 200 [20] or Oxford-102 Flowers [21], would help produce a larger distribution of generated faces and assist with more accurate inference.

Aside from the data, there are a number of additional architectural and algorithmic tweaks that we would like to implement. Due to a lack of training resources, we were unable to experiment with values other than $k = 1$ for the hyperparameter that controls the number of gradient ascent iterations on the discriminator before performing gradient descent on the generator. We feel that, if the descriminator was given the opportunity to fully optimize in between improvements of the generator, higher k values would put more pressure on the generator to leverage the text embedding more during the forward pass. Another area of exploration that we would like to delve into would be altering the network's architecture more dramatically to observe how narrowing and shallowing the networks effects the number of iterations to reach comparable accuracy and how widening and deepening the networks effects on the level of realism in the generated images.

VI. CONCLUSION

Conditional GANs are an impressive tool that allows for the learning of potentially multi-modal probability density functions corresponding to observed data. With such a tool, we can generate high resolution images conditioned on input queries. A wide-variety of interesting applications are enabled by this capability including the domain of text-to-image synthesis which is a technique we investigated in this paper. Although we were not able to generate photo-realistic images of faces, we were pleased to see that, during inference, we could produce images conditioned on textual descriptions of facial features. There are many paths forward, even when constrained by this particular framework, data, and GAN model implementation. We feel that there is much exciting work to be done towards stabalizing the training process of GANs. Also, as hardware continues to improve, more advanced training regimes will be accessible enabling larger networks and systems of networks to be trained in an end-to-end fashion.

VII. CONTRIBUTIONS

The contributions to the project by each group member is as follows. Alex Richards collected the data, did the data preprocessing, and wrote the paper summarizing the project. Mina Yi adapted the simplified StackGAN code to interoperate with the dataset, performed the training of our model, and created our project presentation. Both Alex and Mina carried out research of the state-of-the-art in text-to-image synthesis. Also, they both evaluated the model during the multiple training phases in order to inform the decision making processes used to fine tune the hyperparameters of the model and the additional data preporcessing steps taken in an attempt to produce the best model possible.

REFERENCES

[1] D. Steinkraus, I. Buck, and P. Simard, "Using gpus for machine learning algorithms," in *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, p. 11151120, IEEE, 2006.

[2] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, (New York, NY, USA), pp. 873–880, ACM, 2009.

[3] C. Watkins, "Learning from delayed rewards," 1989.

[4] I. Goodfellow, "Generative adversarial networks," 2014.

[5] M. Mirza and S. Osindero, "Conditional generative adversarial networks," 2014.

[6] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. Metaxas, "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks," 2016.

[7] J. F. Nash, "Equilibrium points in n-person games," *Proceedings of the National Academy of Sciences*, vol. 36, no. 1, pp. 48–49, 1950.

[8] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," Tech. Rep. 07-49, University of Massachusetts, Amherst, October 2007.

[9] C. Sanderson and B. C. Lovell, "Multi-region probabilistic histograms for robust and scalable identity inference," in *ICB*, vol. 5558 of *Lecture Notes in Computer Science*, pp. 199–208, Springer, 2009.

[10] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[11] H. Dong, "text-to-image." https://github.com/zsdonghao/text-to-image, 2018.

[12] E. L. Denton, S. Chintala, a. szlam, and R. Fergus, "Deep generative image models using a laplacian pyramid of adversarial networks," in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 1486–1494, Curran Associates, Inc., 2015.

[13] A. Dosovitskiy, J. T. Springenberg, M. Tatarchenko, and T. Brox, "Learning to generate chairs, tables and cars with convolutional networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 692–705, April 2017.

[14] J. Yang, S. E. Reed, M. Yang, and H. Lee, "Weakly-supervised disentangling with recurrent transformations for 3d view synthesis," *CoRR*, vol. abs/1601.00706, 2016.

[15] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," *CoRR*, vol. abs/1411.4555, 2014.

[16] A. Karpathy and F. Li, "Deep visual-semantic alignments for generating image descriptions," *CoRR*, vol. abs/1412.2306, 2014.

[17] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text to image synthesis," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 1060–1069, JMLR.org, 2016.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[19] N. Kumar, A. C. Berg, P. N. Belhumeur, and S. K. Nayar, "Attribute and simile classifiers for face verification," in *The 12th IEEE International Conference on Computer Vision (ICCV)*, October 2009.

[20] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona, "Caltech-UCSD Birds 200," Tech. Rep. CNS-TR-2010-001, California Institute of Technology, 2010.

[21] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.