From 2D Sketch to 3D Shading and Multi-view Images

Anna Revinskaya
Stanford University
annare@stanford.edu

Yifei Feng
Stanford University
yife@stanford.edu

Abstract

In this paper, we explore applying various conditional generative adversarial networks (cGANs) to sketches in order to generate colored images with 3D looking shading. We use screen-shots of 3D models from various angles as real image data, and apply canny edge detection[2] to get sketch inputs. We first implement and train a pix2pix model[8] to output 3D-looking colored images as our baseline. Then we share the result of applying StackGAN[16] to refine and improve upon our baseline. Lastly, we extend the model to tackle a more challenging task of generating colored images of different view angles for a given sketch.

1. Introduction

Generating fully shaded color images from black-and-white sketches is an interesting problem. For example, designers or artists may want to iterate through multiple ideas with sketches but do not have the time to implement each drawing in detail with colors. The models introduced in this paper can serve as a creative tool to provide a more vivid visualization of these simple sketches.

Recently, generative models, especially Generative Adversarial Networks (GANs) have had great success in image related applications. cGANs have shown results that outperformed many start-of-the-art methods in tasks such as super-resolution [9] and inpainting [11].

In order to further understand GANs and state-of-the-art neural network models developed for these systems, we implemented and analyzed two different models: an encoder-decoder generator model and a two-stage model [8, 16] and experimented with various improvements. Both model were able to generate outputs with color, shading and details. We further extended our models to handle generation of colored image of different view angles and were able to produce spatially coherent images for various viewpoints.

2. Related work

2.1. Image Synthesis With GANs

Generative Adversarial Networks (GANs) have gained a phenomenal popularity and success at creating images from noise input since they were introduced in 2014 [6]. The idea is that there is a generator model that takes noise and input and generates samples, and a discriminator model that tries to distinguish generated samples from real data. Adversarial losses are minimized to force the generated samples to be indistinguishable from real data. GANs currently generate relatively sharp images compared to other generative models[5], but they can be difficult to train due to unstable training dynamics. [13] introduced Deep Convolutional Generative Adversarial Networks (DCGANs) as a set of constraints on the architectural topology of Convolutional GANs that make them more stable to train. In this paper, these guidelines will be followed closely.

2.2. Conditional GANs

Instead of synthesizing images from noise, several works explored conditional GANs where the generator is conditioned on some type of inputs such as text [14] and images [11, 12, 15]. The pix2pix[8] method was proposed to handle general image-to-image transfer. It utilizes a 'U-net' architecture which allows the decoder to be conditioned on encoder layers to get more information. We further build upon this general model with a different up-sampling method and a two-stage GAN.

2.3. Multi-View GANs

Multiple papers have explored generating 2D object views from different angles. Hinton et al. [7] proposed using an auto-encoder to apply transformations such as rotation to an object image at a certain angle to produce an image at another view angle. More recent work explored generating objects given object style, viewpoint, and color [4]. This work uses a supervised CNN to generate images from feature vector representations using 'up-convolutions'.

In contrast to the work listed above, we explore using a GAN to generate 10 specific discrete views based on an

input sketch.

3. Method

Our main goal is to take a black-and-white sketch and generate a colored and 3D-shaded image that matches the sketch. We also explore the problem of generating images of a different view angle. Our approach to both problems utilizes cGAN, which conditions both the generator and discriminator on some input the output intends to follow. We implemented all our models in TensorFlow [1] and based some parts of implementation on CS231N [1].

3.1. Pix2pix Model

Our baseline model is based on the pix2pix paper [8]. The input of our generator is a 256x256x1 black and white edge image. The generator(see Fig.1) consists of 8 encoder and 8 decoder convolution layers. Each layer of the encoder contains a convolutional layer, a batch normalization layer (except the very first layer), and a Leaky ReLU activation. The last encoder layer output is a 512-element vector. Each layer of the decoder consists of a transpose convolutional layer, a batch normalization layer (except the very first layer), and a ReLU activation. The 8th layer of decoder is followed by an additional convolutional layer to map to the output image dimensions (i.e. 256x256x3). It uses $tanh$ function to map values to (-1, 1). First 3 activation layers of decoder are followed by dropout. Note that here dropout is applied instead of adding noise to the input images. Consequently, dropout is applied both at training and validation time.

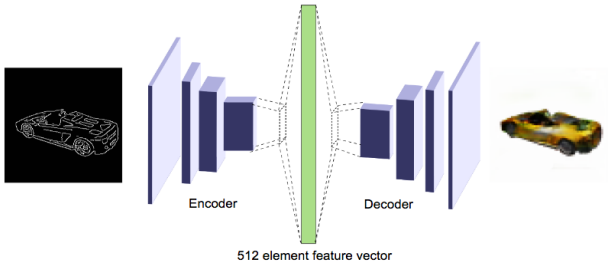


Figure 1: Generator consists of encoder that down-samples the edge inputs to a 512 element feature vector, and a decoder that up-samples it to a full size colored image.

The input of the discriminator is the edge as well as either the generated samples or the original images. The discriminator(see Fig.2) consists of 6 convolutional layers with additional layer at the end to map to a single output value. Then, we use sigmoid function to calculate probability of the input image being either real (i.e. ground truth) or fake (i.e. output of generator) conditioned on the edge input.

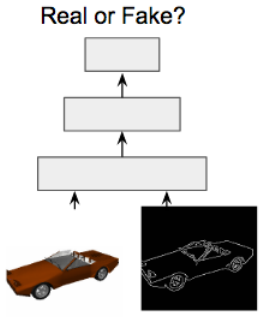


Figure 2: The discriminator learns to classify between real and synthesized images.

3.2. U-Net

The down-sampling step in the naive encoder-decoder architecture could cause loss of some detailed information. For the application in this paper, we especially want to keep the input edge information to ensure the quality of the output. We also implemented the U-Net design introduced in [8] that consists of an Encoder-Decoder architecture with skip-connections (Fig.7). The skip connections add encoder activations to corresponding decoder layers. That is, layer i activations get added to activations at layer $n - i$ where n is the total number of layers.

U-net gives decoder layers access to more detailed information available in encoder layers. In our experiments, training with U-Net architecture starts to produce images that follow original "sketch" much faster.

3.3. Resize Convolution

Transposed convolution is used to up-sample the latent vector back to the full size image in pix2pix architecture. In each decoder layer, the model takes pixel data in the smaller image to generate a square in the next layer. Recent study by Odena et al. [10] shows that transpose convolution is the reason why many neural network generated images suffer from checkerboard pattern artifacts. This is especially obvious when transpose convolution has uneven overlap, that is when the kernel size is not divisible by the stride. However, even when the kernel size and stride are carefully chosen, transpose deconvolution can still be unreliable as it tends to represent artifact creating functions.

One initial decoder layer has kernel size of 4, which is divisible by stride of 2. In order to further avoid artifacts, we replace transpose convolution with resize convolution, which first resizes the image to the dimension of the next layer, and then does a standard 2D convolution. We use

[1] We reused some of the code structure and parts of loss function implementations from CS231N Assignment 3.

nearest-neighbor interpolation for resizing as suggested by [10].

3.4. StackGAN Model

To improve the quality of the output images, we implement a modified version of StackGAN model introduced in [16]. Stage-I GAN reuses pix2pix model to generate a low resolution(64x64) intermediate output conditioned on the edge. Stage-II GAN conditions on both low resolution image generated by the previous stage, and also the edge input again to correct defects in Stage-I results and add more compelling details.

For Stage-II model architecture(Fig.3), the generator is similar to the pix2pix model. But instead of encoding the input with a single vector, the input is down-sampled through 3 network layers to a 16x16x512 block. Then it is fed into 4 residual blocks to learn the additional details, and finally 5 layers of up-sampling blocks to yield the 256x256 output image. The discriminator stays the same.

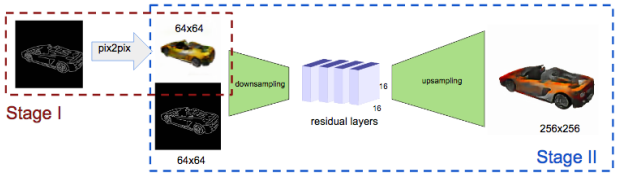


Figure 3: Architecture for stackGAN. Stage I generates a 64x64 blurry image from pix2pix model. Stage-II generator generates full resolution image with more details by conditioning on both the Stage-I result and the edge input.

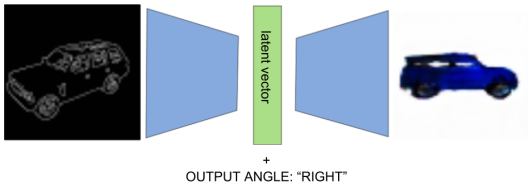
3.5. Multi-View Model

To build on our sketch-to-image results, we experiment with generating other views of the object corresponding to the input sketch. For e.g. we might want to get a "top" view of the car based on a "left-side" view. We experiment with two architectures: generating a single image output for a specified angle, and generating 10 views at predefined angles based on a single input sketch.

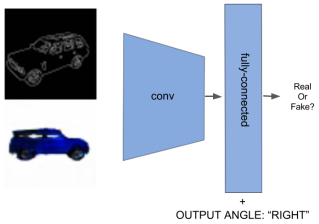
3.5.1 Multi-View Architecture I: Map to one view

The goal of this model is to take an input edge image and orientation and output a shaded image of the object that corresponds to the requested orientation. To achieve this task, we condition both generator and discriminator on the output orientations in addition to the edge image.

To represent "orientation" we assign a numeric label from 0 to 9 to each of the 10 views in our dataset. We then convert these numeric labels to one-hot vectors. To condition our generator on the input views, we add the one-hot



(a) Generator



(b) Discriminator

Figure 4: Multi-View Architecture I: generating a view at another angle. Orientation identifier is added both to the latent vector of the generator and a fully-connected layer in discriminator.

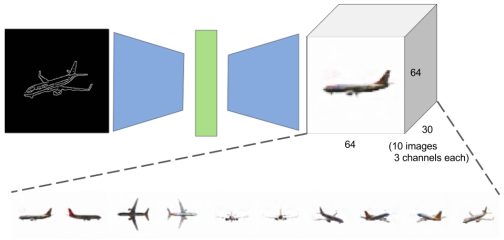


Figure 5: Multi-View Architecture II: outputting 10 views. The figure shows generator architecture that produces 10 output views from a single input sketch.

orientation vector to the latent vector output of the encoder (Fig.4). To condition discriminator on the orientation, we first convert the last layer of the discriminator to a fully-connected layer. Then, we append the one-hot vectors to this fully-connected layer (Fig.4).

Note that volume of training data results in over 600k training data pairs since any of the original 67k edge images can now be combined with any of the 10 output orientations. To speed up and simplify training we train only on a subset of possible view combinations. First, we narrow down our dataset by training just on car models. Second, we experiment with taking only one specific input orientation view. Specifically, we pick "front-left" since front, side and top of the car are all visible in this view.

3.5.2 Multi-View Architecture II: Map to all views

For our second Multi-view architecture, we consider the task of generating shaded images of all views of an object based on a single sketch.

This model builds upon our original same-view model but outputs a 64x64x30 volume instead of the original 64x64x3 image. Each 64x64x3 slice of this volume corresponds to one of the 10 views (Fig.5).

Here, once again, we simplify the task just by training on "front-left" edge input. However, we consider all the model types (cars, aeroplanes and ships).

3.6. Loss Objectives

Our generator loss is a slightly modified GAN generator loss. We take edges image as input instead of noise and compare generated image to ground truth output with additional $L1$ loss term. λ hyperparameter lets us trade off between magnitude of GAN and the $L1$ loss. We set λ to 100 in our experiments.

Our generator loss is defined below (note that x here stands for an edge image instead of a noise image and y is the ground truth image):

$$
\begin{aligned}
L_{G_{orig}} &= -E_{x \sim p_{data}(x)}[logD(G(x))] \\
L_{L1} &= E_{x,y \sim p_{data}(x,y)}[||y - G(x)||_1] \\
L_G &= L_{G_{orig}} + \lambda L_{L1}
\end{aligned} \tag{1}
$$

Our discriminator loss follows traditional definition:

$$
\begin{aligned}
L_D = &- E_{x,y \sim p_{data}(x,y)}[logD(x,y)] \\
&- E_{x \sim p_{data}(x)}[log(1 - D(x, G(x)))]
\end{aligned} \tag{2}
$$

Losses for our first multi-view model that outputs single output image are also conditioned on orientation. Specifically:

$$L_{G_{orig}} = -E_{x \sim p_{data}(x,o)}[logD(G(x,o))]$$

$$L_{L1} = E_{x,y \sim p_{data}(x,o,y)}[||y - G(x,o)||_1]$$

$$
\begin{aligned}
L_D = &- E_{x,y \sim p_{data}(x,o,y)}[logD(x,o,y)] \\
&- E_{x \sim p_{data}(x,o)}[log(1 - D(x,o,G(x)))]
\end{aligned}
$$

where o corresponds to output orientation vector.

Losses for our second multi-view model that outputs all views are not conditioned on orientation and just use the original loss functions at (1) and (2).

4. Experiment

For all experiments, we used an Adam optimizer.

4.1. Dataset

We generate our training data based on ShapeNet dataset [3]. Specifically, we use model screenshots of cars, airplanes and ships as training data for a total of 9517 models. We run `shttps://github.com/ShapeNet/shapenet-viewer` to generate 10 images for each model in the dataset: top, bottom, left, right, front and back views as well as 4 additional turntable views (using cameraAngleFromHorizontal = 15 and cameraStartOrientation = 45 shapenet-viewer settings). In total, our input dataset consists of 95170 examples. We then split the 9517 models to designate 70% of data for train, 20% for validation and 10% for test subsets. ShapeNet viewer generates images proportional to 192x226. So, as additional step, we also pad the images to get the final 256x256 square size.

To simulate "sketch" training data, we apply Canny [2] edge detector to the input screenshots (Fig. 6). Finally, we scale input data so that all values are between -1 and 1.

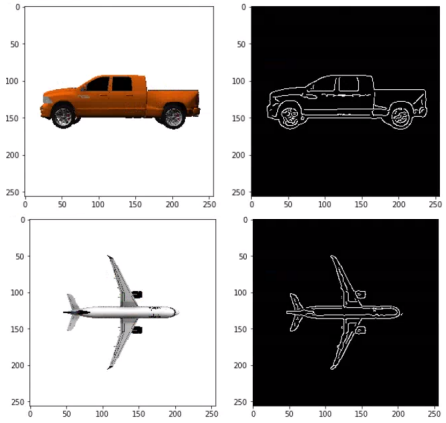


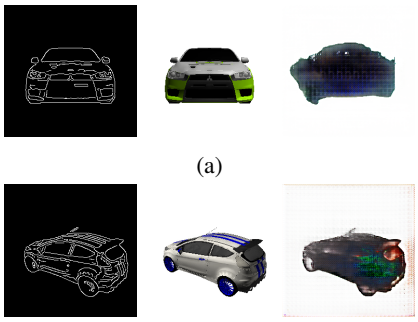
Figure 6: Examples of padded square input images(left), and edges(right) after applying Canny edge detector.

4.2. U-Net vs no U-Net comparison

We ran our model with and without U-Net skip-connections. Without U-Net, images still look like blobs at 20k iterations and frequently look quite different from input edges. At the same iteration, outputs from our U-Net based architecture closely follow the input outer edges(Fig.7).

4.3. Transpose Convolution vs. Resize Convolution

At first, we ran our model with transpose convolution in each layer of the decoder, and as shown in Fig.8, it suffered from the checkerboard artifacts. Then we switched to resize convolution and it greatly reduced the artifacts. All results shown in the following sections were run with resize convolution in decoder.



(a)

(b)

Figure 7: Validation output trained with and without U-Net skip-connections for 20k iterations. The results with U-Net more effectively capture the edge information from input images. Left: edge input. Center: ground truth. Right: Generated images.

Figure 8: Validation output trained with transpose convolution and resize convolution. Resize convolution greatly reduces the checkerboard artifacts. From left to right: edge input, ground truth, output with transpose convolution, output with resize convolution.

4.4. Pix2pix Results

We ran our final pix2pix model for about 90k iterations with batch size of 4. We trained our neural net on screenshots of cars, ships and planes. As shown in Fig.9, the model was able to generate pretty good results. For example, it correctly colored car headlight silver and tire black. The model was also able to generate different colors from the randomness introduced by dropout, such as the blue car and the red ship. However, some outputs contain blurry edges and lacks certain details.

4.5. StackGAN Results

We ran our StackGAN model for about 80k iterations with batch size of 4. It was also trained on screenshots of cars, ships and planes. As shown in Fig.10, the final output images are getting more details and 3D-looking as the iteration count increases. After about 80k iterations, Stage-II was able to consistently generate compelling outputs that complete the details and correct the defects from Stage-I result as shown in Fig.11.

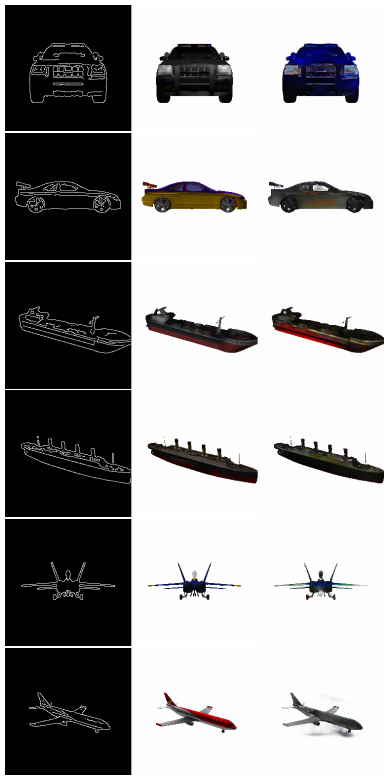


Figure 9: Validation results for pix2pix after 90k iterations. From left to right: edge input, ground truth, generated output.

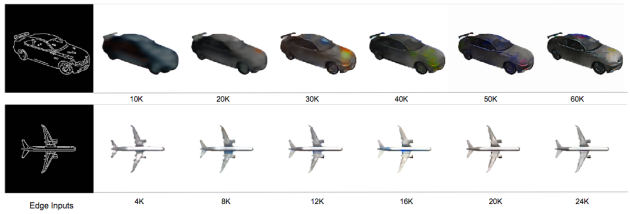


Figure 10: StackGAN Stage-II output for the same edges at various iterations. The model learns to generates results with better details and 3D-looking shading as training continues.

4.6. Multi-view Results

4.6.1 Multi-View Architecture I: Map to one view

We ran our 1st multi-view architecture for 80k iterations with batch size of 4 and a fixed input orientation ("front-left") on 64x64 images. This model takes in an edge image and requested orientation as input and outputs a single output image. To simplify the experiment, we only trained our neural net on car models. We found that this architecture is able to capture general shape of the car in most cases. For e.g. the model is able to recognize that a car has an open

Figure 11: Validation results for StackGAN after 80k iterations. From left to right: edge input, ground truth, stage I output, final output.

Figure 12: Test results for Multi-View Architecture I model after 77k iterations. From left to right: edge input, our output, ground truth. Orientations from top to bottom: ”right”, ”left”, ”front”, ”back”, ”top”, ”bottom”, ”back-left”

top or a spoiler (Fig. 12).

However, our model seems to achieve better results for views ”visible” in the input edge image. Specifically, ”right”, ”top” and ”front” orientations are visible in ”front-left” edge input images and ”back” and ”bottom” are hidden. Consequently, ”back” view of the car often looks blurry and lacks symmetry. This difference is apparent when comparing the 3rd and 4th images from the top in Fig. 12. Front view looks well defined and almost symmetrical. Back view was able to capture the spoiler, but looks blurry overall.

We experimented with removing some of the U-Net connections from our multi-view model. We found that keeping more U-Net connections significantly improves output views ”visible” from in the input edge image but has less impact on ”invisible” views.

4.6.2 Multi-View Architecture II: Map to all views

We ran our 2nd multi-view architecture for 60k iterations with batch size of 4 and a fixed input orientation (”front-

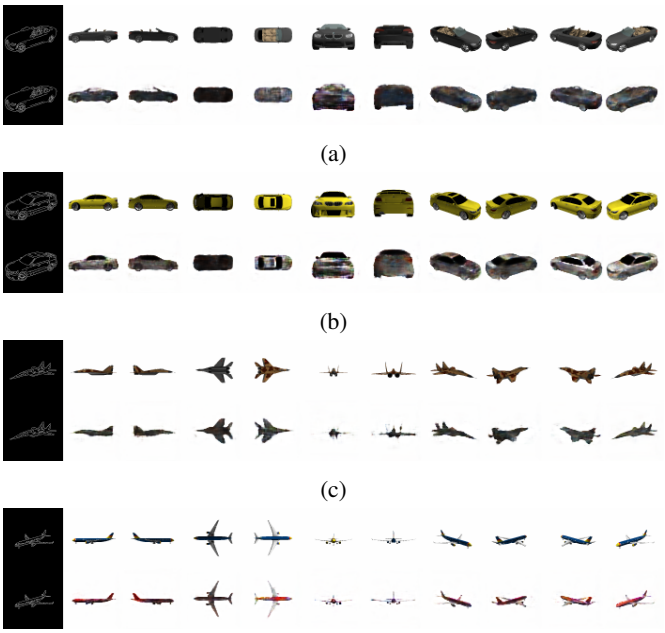


Figure 13: Test results for Multi-View Architecture II model after about 60k iterations. Left column: input sketch. Top: ground truth output. Bottom: our model output.

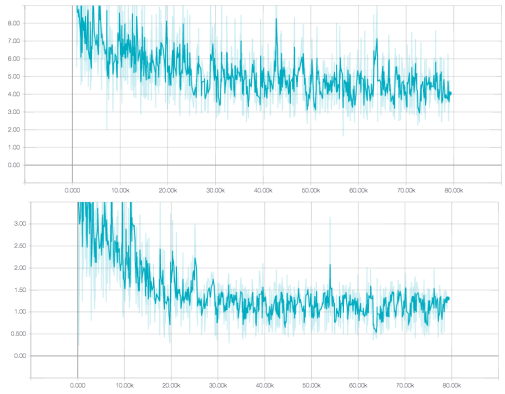


Figure 14: Stage 1 losses. Top: generator loss. Bottom: discriminator loss.

left") on 64x64 images. This model takes a sketch as an input and outputs 10 view images.

Our model as able to learn the right orientation to output for each of the 10 views. All 10 output views also share color and object style.

4.7. Loss

Both our Stage 1 and Stage 2 generator losses decrease gradually (see Fig.14) and flatten out towards the end. Discriminator losses go down a bit faster than generator losses

Figure 15: Stage 2 losses. Top: generator loss. Bottom: discriminator loss.

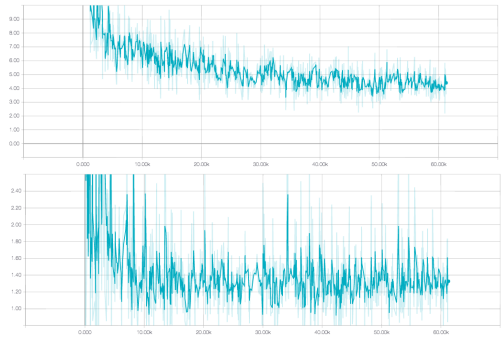


Figure 16: Losses for our "Multi-View Architecture II" that outputs all views for a single sketch. Top: generator loss. Bottom: discriminator loss.

(Fig. 15).

Also, both generator and discriminator losses oscillate as they compete against each other. Whenever generator might produce an image that doesn't look like anything discriminator has seen so far, discriminator loss might go up. Similarly, as soon as discriminator learns to classify new type of images correctly as "fake", generator loss will go up.

Generator loss for our 2nd multi-view architecture seems to be still decreasing slightly towards the end of the graph 16. It is possible it hasn't converged yet and might benefit from more iterations.

4.8. Sketch Results

We tried running our StackGAN model on actual sketches instead of edge images. Several outputs are shown in Fig. 17. Most sketch-based outputs look flat unlike our outputs for edge images.

Figure 17: Results obtained by applying our StackGAN architecture to sketches instead of edge images. From right to left: input sketch, Stage I output, Stage II output.

Figure 18: Failure cases of our StackGAN architecture. From left to right: edge input, ground truth, our output. Most of the outputs of StackGAN look good. Occasionally outputs might be blurry and lack details.

4.9. Failure Cases

Several failure cases are highlighted in Fig. 18, Fig. 19 and Fig. 20.

5. Conclusion

Our StackGAN model achieves compelling results for edge inputs. The model is able to add 3D-looking shading, color and detail to the input edges. However, our model did not generalize very well to sketches. A lot of sketch outputs look flat. We are likely to get better results during test time if we train on actual human sketches.

We extended our architecture to produce other views of the input sketch. For simplicity, we just ran Stage I for our Multi-View model. Our Multi-View model successfully captured general shape based on the input edge image and was able to produce 10 views from a single edge input. However, our multi-view outputs often lack smoothness and detail that we can see in our single view outputs. Further experiments such as using an L2 instead of L1 norm for a smoother image and tuning hyperparameters might yield better results.

To build on our Multi-View results, we could explore taking an edge image and constructing actual 3D shapes by outputting a voxel grid instead of a set of images.

Figure 19: Failure cases of Multi-View Architecture I model. From left to right: edge input, ground truth, our output. Common failure cases include: asymmetric "back" view outputs, missing details and patch artifacts causing outputs that don't look smooth.

Figure 20: Failure cases of Multi-View Architecture II model. Left: sketch input. All other images are the output. The Multi-View model fails to produce a clean output for a sketch input.

Contribution

Both team members studied various papers and implemented the initial model and pipeline, while Yifei Feng fo-

cused on StackGAN model and Anna Revinskaya focused on multi-view model. Both members drafted the report and poster together.

Acknowledgments

We would like to thank the CS231n course staff including instructors and TAs for their instruction and guidance throughout the course!

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] J. Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[3] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. Shapenet: An information-rich 3d model repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[4] 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*, 39(4):692–705, 2017.

[5] I. Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 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 *Advances in neural information processing systems*, pages 2672–2680, 2014.

[7] G. Hinton, A. Krizhevsky, and S. Wang. Transforming auto-encoders. *Artificial Neural Networks and Machine Learning–ICANN 2011*, pages 44–51, 2011.

[8] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *arxiv*, 2016.

[9] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint arXiv:1609.04802*, 2016.

[10] A. Odena, V. Dumoulin, and C. Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[11] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2536–2544, 2016.

[12] Y. Qu, T.-T. Wong, and P.-A. Heng. Manga colorization. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 1214–1220. ACM, 2006.

[13] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[14] 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 Machine Learning*, volume 3, 2016.

[15] C. Vondrick, H. Pirsiavash, and A. Torralba. Generating videos with scene dynamics. In *Advances In Neural Information Processing Systems*, pages 613–621, 2016.

[16] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *arXiv preprint arXiv:1612.03242*, 2016.