CSCI 1430 Final Project Report: Lego-ization of Images using CycleGAN

LE(t's)GOOOOOOOOOOOOOOOOOO!!!: Caroline Zhang, Trey Wiedmann
TA name: Anh
Brown University

Abstract

Neural networks in deep learning have become a powerful tool in style and domain transformations, and have been especially successful with media such as images. Using two GAN models in a cyclic fashion has let us move samples from one model to another in an efficient manner to transfer styles, and without the need for paired data.We wanted to try out image style transfer where the the style domains relied upon learning 3D features such as shadows and perspective, and thought that a real world to Lego style transfer was well suited for that. However, our end results indicate to use that such a style translation requires a deeper knowledge of 3 dimensional imagery and geometry than could be learned by our model.

1. Introduction

Our goal was to train a model to take as its input a real-world object or scene and output an image of an equivalent Lego object or scene. We treated this as a image translation problem and decided to use a CycleGAN architecture as such a model does not require paired data. Many style transfer models are focused on very 2-D styles – for instance, taking a photo of a landscape and outputting what that landscape would look like if it were painted by van Gogh. In these, the output style tends to be more evenly or even randomly distributed across the image, and such a distribution can still result in a somewhat visually successful result. We were interested in seeing if the same architecture could be used to learn a more complicated 3-D style, where "style" could not be evenly or arbitrarily applied to the image but rather depended on the original input image's perspective.

2. Related Work

In neural style transfer, deep neural networks are applied to extract content from one image and style from a different image, and generate a new combined image [2]. Though these kinds of models have very promising results, we decided that "Lego" is more abstract and complex than a style one could extract from a single reference image.

The paper we mainly referred to in our project was "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks" [8]. The authors' CycleGAN architecture consists of two generator models and two discriminator models. Similar to traditional GAN architectures, generators are trained to minimize the discriminator's ability to differentiate real and generated images in the output domain, while discriminators are adversarially trained to maximize their ability to differentiate these real and faked images. CycleGAN adds in an "cycle loss" for the generators, which is the reconstruction loss of translating an image from domain A to domain B and then back to domain A. This enforces generators to retain the content in the original images, effectively preventing mode collapse. Based on the success the authors showed on various domains in their paper, we attempted to apply their method to our project. This fit ours needs better, as we could provided many images in order to provide our models a deeper understanding of "Lego" style.

Our project idea was partly inspired by the paper "GANcraft: Unsupervised 3D Neural Rendering of Minecraft Worlds", which rendered 3D digital block worlds as footage of realistic world scenes. While the authors of this paper used a GAN model, they also used neural rendering techniques that we were unfamiliar with [3].

3. Method

As mentioned in the introduction, we decided to use a CycleGAN architecture as it is a popular model for image translation problems and does not require paired data. We provide two trained models: one for real-world scene to Lego scene translation (scene-scene model), and the other for real-world object to Lego object translation (object-object model). For each model, we provide our own training and testing datasets, as there were no existing datasets that fit our needs.

For the object-object model, we scraped images of Legos and objects against clean white backgrounds from Flickr, Wikimedia Commons, and the Lego Website. However these

scraped images required a lot of manual cleaning. For instance, around 75 percent of the 3,000 images we first gathered from Flickr either had real world object in them or had people in them. This was despite filtering the Flickr images to be white while using Flickr's scraping API. Another issue we encountered was decided what types of real world objects we should obtain images of. We decided to gather images that we felt could be more easily converted to Legos, and that there existed Lego sets of, such as houses, cars, animals, and furniture. However we had a difficult time obtaining a large quantity of houses against white backgrounds (they often ended up being cartoon houses) so we also used a lot of images of gingerbread houses against white backgrounds. Another issue was with sizing. A lot of the images we scraped tended to be very large – this was especially the case with Flickr, as often the images were straight from people's cameras. We were concerned that resizing the image to 256x256 would blur our the more distinct features of Legos, so we also manually gathered a lot of Lego set images – around 1000 – via screenshotting. We decided to do a lot of the Lego data collection manually because we wanted the data to be as clean as possible so that the Lego features were the only features. Another reason this was done manually was to get images of Lego sets that were representative of all kinds of objects. For instance, a good amount of the usable Flickr-scraped Lego images were of cars. While we did try various webscraping approaches, they did not reliably get good results. When we scraped from various store websites, the saved images either were entirely not Lego (like various decorative images on the website) or they were not just of the Lego set, and instead featured the Lego set on a box, or someone's hand in the photo, etc. With webscraping it was also difficult to account for the distribution of types of lego sets. So instead we took many screenshots from different Lego categories. These images broadly fell into these categories of objects: City/House/Architecture, Animals, Vehicles, Plants, and Simple/Random structures. We made sure that the Lego sets we screenshotted were not too niche – that is, that the Lego set didn't feature any especially large round features, so that the model would hopefully learn the lines, corners, and small circles that we believe characterize Legos.

Our data collection process was quite similar for the scene-scene model. However, we were able to take advantage of the large image sizes of the Flickr Lego images by just cropping several 256x256 windows from Lego set images. We also began this process after we had already begun training our model on the object dataset, so we were aware of some of the issues our model was running into and how the 3-D nature of Legos could result in difficulty learning a consistent set of Lego features. To attempt to resolve this, we mostly used Lego scene images that were more from the top view, so that there were large areas patterned with small circles. We also tried to make the scale of these small circles more consistent across images.

Overall, we ended up with around 2400 images to use for training and testing the object-object model, and around 1300 images for the scene-scene model.

For preprocessing, we converted each image to a 256x256x3 numpy array. To preprocess the images in the object dataset, we padded the sides of the images with white so that they were square, and then resized them. We did this because we felt that the aspect ratio of the Legos was an important feature and didn't want those features to be too varied across the images. As for the images in the scenes dataset, if the image was bigger than 256x256, we cropped in the center.

We implemented the CycleGAN architecture as described in the original paper. We built two generators and two discriminators, and feeding into one another in a cyclic fashion. Our generators use convolutional layers to downsample images, resnet blocks to process features, and then transposed convolutional layers to upsample the images back to the original size [4]. Just as the CycleGAN paper, we implement our discriminator as a PatchGAN, which discriminates on 70x70 patches of the input images with convolutional layers [5].

During training, to aid the generator and prevent diminished gradients, we implement image pooling as in the CycleGAN paper, which uses a buffer to provide previous images to discriminators rather than the most recently generated ones. Additionally, following the CycleGAN paper, we use a least squares loss for the discriminators rather than sigmoid crossentropy loss [6]. This also leads to more stable training results. Along with cycle loss, we also used identity loss on the generators, which encourages output images that are similar in color and composition to the inputs. Both cycle and identity loss use L1 loss. We use a learning rate of .0002 with ADAM optimizers, as in the CycleGAN paper.

We also experimented with batch size, various weighings of losses, and number of convolutional layers and dimensions. The original paper used a batch size of 1, which we tried. We also tried a batch size of 2, 8, and 32. We were unable to fully train with a batch size of 32 since we ran out of memory on Colab. Our final implementation uses a batch size of 8. We found that the quality of output images did not vary at all that much with batch sizes of 1, 2, and 8.

We used Flickr's web-scraping API to scrape images from their site, and we used BeautifulSoup to scrape from Lego and Wikimedia. To preprocess our data, we used PIL, Scikit-Image, and NumPy. Our model was implemented in PyTorch and also used torcheval to evaluate loss metrics. We used the CycleGAN authors' PyTorch implementation as well as this and this Github repo as reference while writing our implementation [7] [1]. However, we experimented with various model hyperparameters such as number of convolutional dimensions and layers, weighing of identity loss, and batch size.

4. Results

Objective analysis of our results can be difficult, because of the nature of image translation with unpaired data. We qualitatively found that our object-object model was unable to produce Lego-like results given an input image of a real-world object. However, it was a bit more successful at converting an image of a Lego object into an image of a real-world object as it removed the Lego-like features such as the dark lines indicated the separation of bricks, and the small circles patterning the tops of the bricks (see Figure 2). Our scene-scene model was more successful at converting an input real-world scene image into the corresponding Lego scene image. We think that this is because the "style" was easier to learn, as the bulk of the Lego scene images featured repetitive raised circles (see Figure 1). However, our scene-scene model was unable to convert Lego scene images into a corresponding real world scene image. Instead, as the model trained for longer, the output images (given an input of a Lego scene) would lead to increasingly white images (See Figure 4). We believe this could be because of diminished gradients due to a strong discriminator.

We experimented with various weights of losses, such as increasing and decreasing the amount of identity loss used. However we found that decreasing identity loss didn't convincingly transfer an image of an object into the corresponding Lego object, and that we still needed to use at least a certain amount of identity loss to to preserve the recognizability of the input. We also experimented with various aspects of the model architecture, such as the number of convolutional layers and the convolutional channels. While these changes did lead to different results, we found that their quality was roughly the same, in that the model was still not learning and applying Lego features to the input image.

Another thing we found interesting was the changes the model made to the input Lego images (both Lego objects and Lego scenes) at each epoch. The progression of these changes was consistent across our experimentations with different weightings of identity loss and number of convolutional layers. Essentially, during the earlier epochs (up to around epoch 5), our model would output images featuring checkerboard artifacts which were a result of the upsampling our model used to re-generate an image. However, further training diminished the prominence of the checkerboard artifacts. What was surprising to us was that after the checkerboard artifacts diminished, further training didn't change the shape of the object in the output image. This was true across all both possible style transfer domains in both the scene-scene model and the object-object model. Images outputted by the model during epochs 5-20 (approximately) would retain the input photo's shape – the outlines of the original object would essentially be the same, as well as the outlines within the object. However there were also random clusters of neon pixels across the object, and these occurred most at the edges separating features (see Figure 3). Further training would decrease the amount of neon clusters, however it would not modify the edges of the features, which is of course integral to a Lego style transfer. We found that further training would often modify the colors in the input photo – for instance, some patches of the photo would get a bit darker. However this was not in accordance with any geometric properties of Legos, and the patches bore no consistent shape or placement across the input images.

Epoch	Cycle Loss	Avg Fake Acc	Avg Real Acc
1	0.6871	0.685	0.685
5	0.657	0.706	0.681
10	0.631	0.658	0.739
15	0.6305	0.689	0.728
25	0.5761	0.69	0.654
35	0.5723	0.689	0.691
50	0.5605	0.642	0.648

Table 1. Generator cycle losses and discriminator test accuracies for our final object-object model.

Epoch	Cycle Loss	Avg Fake Acc	Avg Real Acc
1	1116.67	0.835	0.94
5	1115.39	0.999	0.99
10	1115.17	1	0.98
15	1115.02	1	1
20	1114.99	1	1
25	1114.94	1	0.98
30	1114.95	1	1

Table 2. Cycle losses and discriminator test accuracies for our final scene-scene model.

We also found that our scene-scene model was struggling to decrease loss. We think that this is because the discriminator learned too quickly, despite implementing image pooling to help the generator stay ahead. This may have been because the images in the real-world datasets and the Lego datasets were already too distinct from one another, thus it was "easy" for the model to determine which images were fake and which images were real. The above figures feature the average cycle losses for both our scene-scene model and object-object model, as well as the average test accuracies of the discriminators. We find that our model "learned" a bit better when trained on the objects dataset, as the losses decrease more reliably, and that the discriminator did not converge nearly as quickly as that of the scene-scene model.

Figure 1. Results of our real world scene to Lego scene transfer. *Top left:* Input image, part of our testing dataset *Top right:* Output after 1 epoch *Bottom left* Output after 6 epochs *Bottom right* Output after 30 epochs

Figure 2. Results of our lego object to real world object transfer. *Left image:* Input image, part of our training dataset *Middle:* Output after 22 epochs *Right:* Output after 72 epochs

4.1. Technical Discussion

One question our project raises is the differences between 2D and 3D style and how that can be better accounted for in style transfer models. At least in our heads, it's easy to imagine what converting an image to Lego style is. We've had enough experience in the world to have a grasp of how something 3D in real life could translate into a convincing brick-like structure, so we can easily interpret "Lego" as being of its own style. When we look at an image of an object at a certain angle, we can pretty quickly imagine what the corresponding Lego object would look like at that angle – how the top might have small raised circles, and how the sides would have small but wide rectagular blocks of color. However those variations of Lego style have shown to be

Figure 3. Results of our real world object to lego object transfer. *Left:* Input image, part of our training dataset *Middle:* Output after 22 epoch *Right:* Output after 72 epochs

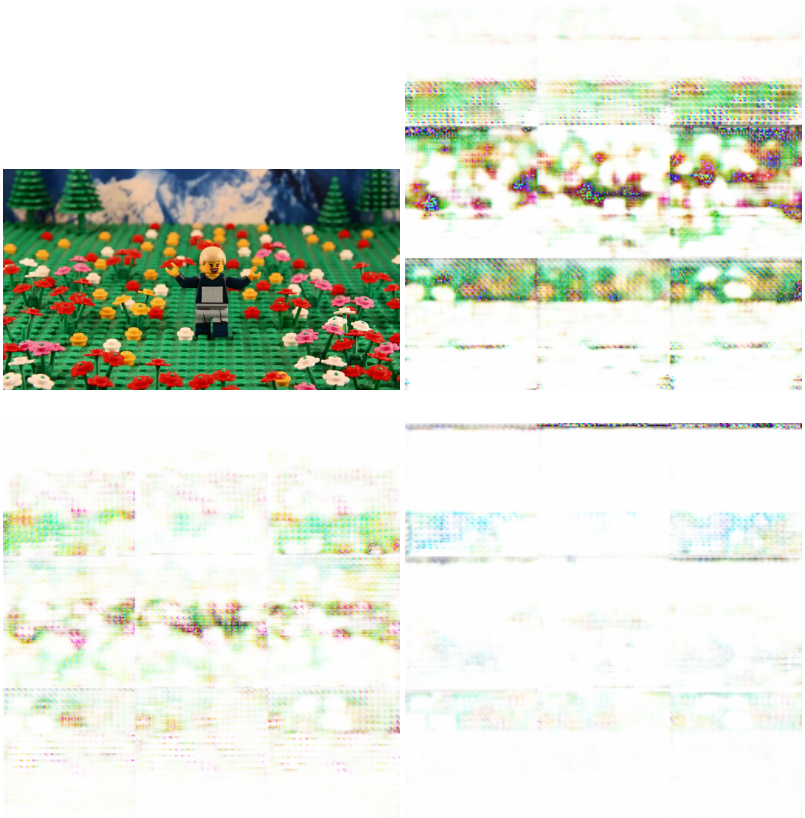

Figure 4. Results of our lego scene to real world scene style transfer. *Top left:* Input image, part of our testing dataset, although the model was also trained on square windows from that image *Top right:* Output after 1 epoch
Bottom left Output after 6 epochs *Bottom right* Output after 30 epochs

very difficult for the model to learn. The visual features that make Legos distinct – and Lego structures believable – are quite varied, and must be applied in very specific ways. I think that given the small size of our datasets the differences between the Lego images were likely too vast and so the model was unable to learn well.

If we were to further work off this project, there are two different approaches we think we could take to make our outputs better. We would like to compile paired data – for instance, images of a real tractor and images of a very similar Lego tractor, and at very similar angles. If possible we could also have associated data about the perspective or matching

features. However there is no existing dataset like this, at least not to our knowledge. This approach would help the model to learn perspective and how the style of Lego is dependent on that. Another approach we would take is creating a larger, and even more standardized set of Lego images. For instance, we could just use images of the top views of Legos, and have the images scaled so that across each image, the small circles at the top of the Lego bricks are roughly the same. However, this would require a lot of manual inspection as there is no existing dataset like this to our knowledge.
Finally, we think that the premise of our project would present an interesting, and likely better-suited, challenge to the field of neural rendering. A neural rendering approach could leverage knowledge of the specific geometric properties and constraints of Lego bricks to ensure that the resulting images conform to the desired Lego style.

4.2. Socially-responsible Computing Discussion via Proposal Swap

- Their first item in the proposal critique wasn't exactly a critique but rather a suggestion for datasets to use. We appreciated their suggestion, however the datasets they mentioned were images of separate, individual Lego bricks rather than Lego sets. The Github repos they mentioned were also related to Lego brick classification so the datasets they used did not suit our specific needs.

- If our model worked really really well and was able to actually Lego-ify images, then we might characterize our project more as a brick-ification of images rather than a Lego-ization of images to avoid any copyright issues. We would similarly then present the results as potential toy brick structures rather than potential Lego structures. However, given the model's outputs, we do not think our project poses copyright infringement threat to Lego.

- Since our model did not output convincing Lego images, we are unable to quantify how valid of a Lego structure it is (in terms of whether or not it would be possible to build) since we are unable to interpret much of our results as Lego structures. However we think that the question they ask would be a really interesting problem to try to solve – given an image of a structure, how might have it been built?

- If we had been able to create a paired dataset of Lego images to real-world objects/scenes, one SRC concern could be biases in which types of images/objects translated for convincingly into the Lego domain, as those differences might reflect the distribution of types of paired data images. For instance, the model might be better at Lego-ifying images of cars rather than trees since there are lots of Lego sets of cars.

5. Conclusion

Although the CycleGAN architecture shows promising results in many domains, we conclude that real-world to Lego translation requires a deeper understanding of 3 dimensional imagery than could be achieved with these techniques within the time-frame and computational limitations of our project. This project has also emphasized to us the importance of data collection, preprocessing, and standardization as it was quite a big limitation in our project.

References

[1] Symbolic music genre transfer with cyclegan for pytorch. https://github.com/Asthestarsfalll/Symbolic-Music-Genre-Transfer-with-CycleGAN-for-pytor 2022. 2

[2] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style, 2015. 1

[3] Zekun Hao, Arun Mallya, Serge Belongie, and Ming-Yu Liu. Gancraft: Unsupervised 3d neural rendering of minecraft worlds, 2021. 1

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 2

[5] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018. 2

[6] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks, 2017. 2

[7] Guan'an Wang. Pytorch image translation gans. https://github.com/wangguanan/Pytorch-Image-Translation-GANs, 2019. 2

[8] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020. 1

Appendix

Team contributions

Please describe in one paragraph per team member what each of you contributed to the project.

Caroline Zhang (czhan157) I collected and preprocessed the data, and helped write and train the model. I also created and printed the poster.

Trevor Wiedmann (twiedman) I researched model architecture and implemented our training and testing methods.