Text-Adaptive Generative Adversarial Networks

Emma Qin, Will Schwarzer, Kyra Wilson, Orlando Zuniga

March 16, 2020

Abstract

In this study, we replicate the text-adaptive generative adversarial network proposed in [4]. We do this for two reasons: first, image modification with natural language is an interesting and challenging problem, and second, replicating studies can provide additional information about the vailidity and generalizability of the original authors' claims. Our replication followed the specifications of the network given in the published study and supplementary materials (and not the publicly available code provided by the authors) and we trained our network using the CUB [9] dataset. We evaluated our network using L2 error and informal qualitative measures and found that our results were significantly worse than the results from the original study, due to issues in accurately and comprehensively recreating the network.

Contents

1 Introduction

Image synthesis is a difficult problem in computer vision, but the use of Generative Adversarial Networks (GAN) has led to great progress being made towards generating realistic images [1]. GANs typically consist of a generator, which is able to generate images, and a discriminator, which is able to discern whether an image is real or made by a computer. These two parts of the network typically work against each other, with the generator trying to make more and more accurate images to fool the discriminator and the Discriminator trying to get better and better at predicting what kinds of images are real or not.

In their basic form, the images generated by GANs are not conditioned on any information other than random noise: the generator simply takes in a random noise vector as input, and generates a random image as output [1]. This means that the user is unable to control in any way what kind of image a GAN generates, aside from modifying the training data. Given this limitation, one of the most exciting extensions of the classical GAN paradigm is conditional GANs [1, 2], where the generator is instead given some input vector y as a starting point for its image generation, which could be anything from a class label for the image to be generated [2] to a free-form natural language description of the image [3]. When the discriminator is judging any given image, it then also learns to judge how well the image matches the condition it is associated with, based on the condition/image pairs in the training data. The feedback thus produced by the discriminator allows the generator to generate images that are not only lifelike, but also accurately depict the given input.

Another possible extension to the problem of image generation with corresponding natural language is instead modifying preexisting images in order to match a new text description. To do this, we will still feed in the text description to both the generator and the discriminator, and allow the discriminator to judge the generated images based on how well they match the description; however, we will also input a real image to the generator for it to base its generated image on. This approach is the subject of the paper we're replicating [4], as well as several previous papers [5, 11].

Although replication as a task may seem unimportant (especially since a successful replication might be simply redoing work that someone else has already done), it is actually very valuable and beneficial work that is becoming more necessary as problems surrounding replicability are brought to light. For example, one study has estimated that as few as 40% of foundational psychology studies have results that could be reproduced [12]. This is obviously problematic given the number of studies and experiments that have been conducted since then that were possibly operating on faulty premises.

Psychology is not the only discipline that suffers from a replicability problem. Machine learning also potentially faces a similar problem where original results cannot be reproduced, despite authors making code and network architectures publicly available. An additional problem for machine learning is that even if results can be properly reproduced, it is unclear how generalizable these results are because they are often dependent on very specific hyperparameters or datasets. In this project, we will be seeking to address this problem by replicating the text-adaptive generative adversarial network mentioned previously. Perhaps unsurprisingly, our results were not identical to those of the original authors, which was likely a consequence of difficulties in replicating machine learn papers in general.

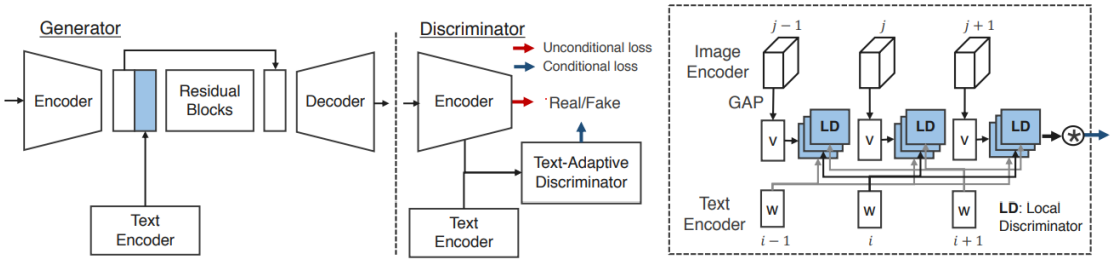
2 Methods

2.1 Network Architecture

Previous work to achieve natural language image modification is the Semantic Image Synthesis GAN (SISGAN) [5]. The network that we will be replicating in this project, the Text-Adaptive GAN[4], uses a slightly modified version of SISGAN [5]. Both of them has GAN structure, which contains a generator and a discriminator, where they work against each other to produce images that are modified according to the input text descriptions.

2.1.1 Generator

In the generator, the encoder is used to make vectorized representations of images and natural language text descriptions. For images, this is done using a convolutional neural network (CNN), while text encoding is done using



(a) GAN structure (b) Text-adaptive discriminator

Figure 1: The network architecture used in [4].

a bidirectional GRU (gated recurrent unit, a type of recurrent neural network (RNN)). "Bidirectional" means we encode the sentence both forward and backward. In this way, the beginning and the end of the sentence are equally important. GRU is a simplified LSTM with less parameter. The reason we use GRU is that RNNs suffer from short term memory and vanishing gradient problems. During back-propagation, the gradient shrinks through time and this is the value used to update the weight in the RNN. Hence, RNNshave difficulty remembering what has been seen in a longer sequence. GRUs are a great solution to these problems. For every unit, it carries in the past information so that it does not forget it. In SISGAN, the RNN is pretrained before being used with the data of interest, but the TAGAN RNN is trained from scratch, though its embeddings are supplemented with pretrained embeddings from fastText [6]. While the authors of [4] do not state why they chose not to pretrain their RNN as was done in previous studies, we hypothesize that one possible reason for this is that the language used in the image captions is specialized enough that pretraining the neural network on more general natural language may not provide any benefit for the task at hand, and may even worsen its performance.

Another alteration made for text encoding with TAGAN is that conditioning augmentation [7] is used in order to generate more text-image pairs and minimize any outlying data in the training set. Conditioning augmentation uses a similar principle to variational autoencoders (VAEs): rather than encoding each raw data point to the same vector every time, the data are instead first encoded into vectors of normal distributions, from which individual representations are then sampled. This serves to "smush" the representations out to fill more of the representational space, forcing the generator to process textual inputs more flexibly, thereby allowing it to interpolate to novel sentences more easily.

Once both the image and the text have been encoded, the encodings are concatenated to create a visual-semantic text representation to be used as input for the residual blocks. These blocks, which are the part of the generator that actually modifies the image representation to match the language representation, are sets of standard multilayer perceptrons (MLPs, aka basic feedforward neural networks) with one particular quirk: each residual block saves the input vector it is given, and adds it to the output vector that it produces. This means that what each residual block is actually learning is the difference (residual) between the input and the output - i.e. the modification to be made to the input.

Finally, the decoder returns the latent feature representation to an image of the desired size by passing it through several upsampling layers. A batch of such generated images is then passed to the discriminator, which determines the validity of the images (whether or not they're real, and whether or not they match their descriptions); in the same epoch, a batch of real images is similarly given to the discriminator for validity checking.

2.1.2 Discriminator

In its basic structure, the discriminator used in this paper parallels that of [5]. Like all GANs, part of the discriminator is unconditional, i.e. it ignores the natural language descriptions of each image: it simply encodes the image

(using a CNN, like the generator) and classifies it as either real or fake using unconditional loss. However, like all conditional GANs, it also has a conditional (or "text-adaptive") part to it, which classifies whether or not the image accurately depicts the description associated with it by conditional loss. This is where the GAN architecture employed by this paper diverges most significantly from previous work. Whereas the architecture in [5] simply encodes the description as a whole, then asks the discriminator to classify the combined image and text representations, this paper uses a much more complicated word-level discriminator. Besides two loss function mentioned above, reconstruction loss is also introduced by the paper to measure the difference between original images and generated images. It helps the generator to keep the non-manipulated features to stay the same, including background.

The basic idea behind this word-level discriminator is to consider each individual word of the description, determining the semantic fit between each word and the image in question. To do this, the paper uses a local discriminator, which takes in an image representation and a word representation as input, and returns a value representing whether or not the word in question (for example, "brown", or "bird") appears to be present in the image; this local discriminator is then used on the image with each individual word of the description, and the results of each of these trials are added together to determine the overall match between the image and the description. The goal of this finer-grained discriminator is to provide the generator with more precise feedback: rather than only being able to back-propagate the information that a given bird did not match its description, the word-level discriminator will be able to tell generator that the bird was not sufficiently black, or that its beak was too long.

That said, the above description glossed over a few details. First, the results of each local discrimination test are not just added together to get the overall result; instead, the network uses an attentional system to add together these results, allowing it to place more weight on words that it has learned are important. (For example, perhaps it has learned that it should really pay attention to whether or not the image matches the word "brown", but the word "the" tends to convey less information.) This turns the overall discrimination score into a weighted sum of the invidividual tests, where the weight for each word's local test is learned over time. We achieved this by creating a 1D sigmoid local discriminator f_{w_i} for each word vector:

$$f_{w_i} = \sigma(\mathrm{W}(w_i)) \cdot v + \mathrm{b}(w_i), \tag{1}$$

where $\mathrm{W}(w_i)$ and $\mathrm{b}(w_i)$ stands for the weight and bias of each word vector. v is an 1D image vector computed by applying global average pooling to the feature map of the image encoder. And attention is calculated by

$$\alpha_i = \frac{\exp(u^T w_i)}{\sum_i \exp(u^T w_i)}, \tag{2}$$

where u is a temporal average of w_i. Together, the local discriminator f_{w_i} and attention α_i contribute to calculating the conditional loss.

In addition, there is a bit of finesse in the exact image representations that the local discriminator takes as input: in fact, there are three different local discriminators, each taking in a different type of image representation! These different image representations represent different levels of detail in the image: "conv3" is the finest-grained (i.e. highest-dimensional), while "conv5" is the coarsest. These odd names stem from the details of how the representations are created by the network's CNN.

CNNs generally produce low-dimensional image representations from high-dimensional images by separating blocks of convolutional (sparsely connected) layers with pooling layers, which compress the dimensionality of the representation - often by a factor of 2 each time. (Specifically, they usually compress the width and height dimensions of the representation, while increasing its depth.) Thus, if conv3 is the last convolutional layer of the third convolutional block of the CNN, and if the image representation at this point has dimension (w, h, d), conv4 might have dimension $(\frac{w}{2}, \frac{h}{2}, 2d)$, and conv5 would be $(\frac{w}{4}, \frac{h}{4}, 4d)$. Generally, a CNN feeds an image through all of its convolutional blocks, before using several dense layers at the end to compress the representation fully to the desired size. However, you can also short-circuit the full process by instead converting the results of each convolutional block directly to a $(1, 1, *)$ representation, in which we do not care about the last dimension. This is done using global average pooling (GAP), where the values of each (w, h) slice of the representation are averaged together to get a single number, turning the (w, h, d) representation in conv3 into a $(1, 1, d)$ representation, while conv4 and

conv5 produce $(1,1,2d)$ and $(1,1,4d)$ representations, respectively [8]. Since CNNs generally produce more and more coarse-grained representations while moving through their layers, it is reasonable to assume that the representation taken from conv3 will have the best low-level detail, while conv5 will be a more high-level representation of the image, as desired. The authors confirm this in the ablation study shown in Figure 2.

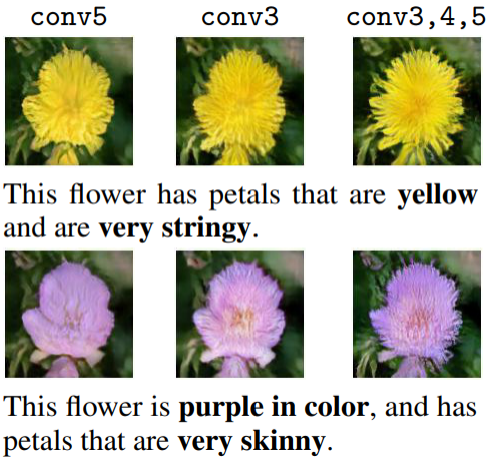


Figure 2: Results of only using certain image representations in the discriminator.

Once these different representations have been produced from the CNN, each of them is fed into its own corresponding local discriminator. This discriminator then attempts to determine whether or not the image at that particular scale matches the given word well: for example, conv3 might work well for determining whether or not a flower's petals are indeed "stringy", while conv5 might be good at determining whether or not the flower is purple. The results of these three local discriminators are then summed together attentionally, the final result of which is used as the value of the overall local discrimination test for that individual word. Once the discriminator has determined the total word-level fit of the description and the image, feedback on the reconstruction, conditional, and unconditional loss is provided to the generator, to allow it to produce more realistic and text-accurate images, while the discriminator receives feedback on whether or not the image was correctly classified as being real/fake and matching/not matching the description.

2.2 Datasets

The datasets used to train and test TAGAN are CUB[9] and Oxford-102[10] datasets. These are both fairly standard and commonly used image datasets for machine learning tasks and in particular the predecessors of TAGAN[5]. For this project, we have chosen to focus on replicating the results using CUB as our primary goal, so further discussion of datasets is limited to that one. The CUB dataset includes 11,788 images of birds in 200 different categories, as well as annotations containing 15 part locations, 312 binary attributes, and 1 bounding box for each image. Because this paper aims to alter images based on descriptions that are natural language rather than sets of features, the CUB dataset must also be supplemented with natural language captions. The captions the authors used (which we also intend to utilize) were created for a different investigation which also employed CUB and Oxford-102 datasets[3]. These researchers created 10 natural language captions for each image in the dataset according to the annotated features in CUB. The annotated images and text are then used as inputs for the generator. In our replication, 80% of the data will be devoted to training and 20% will be devoted to testing. Furthermore, the authors of both the present paper and Dong et al., 2017 used standard data augmentation techniques for images, namely, cropping, rotation, and flipping, which we will also replicate.

2.3 Implementation

We implemented our method using PyTorch. For the generator, we adopt the architecture of the generator from[5] to encode and decode 128 × 128 images. We additionally encode a text using a bidirectional GRU and the pre-trained fastText [14] word vectors. In the discriminator, we use conv3, conv4, and conv5 for the local discriminators. We trained our network up to 100 epochs using Adam optimizer [13] with the learning rate of 0.0002, the momentum of 0.5, and the batch size of 64. For data augmentation, we used random cropping, flipping, and rotation. We resized images to 136 × 136 and randomly cropped 128 × 128 patches. The random rotation ranged from -10 to 10 degrees. We set the weighting of conditional loss for discriminator (λ_1) to 10 and the weighting of reconstructive loss for generator (λ_2) to different values starting from 10 and going all the way up to 1000. A higher λ_2 value the closer it is to the original image. The original TAGAN researchers used a λ_2 value of 10. These values allow us to consider the visual quality and the training stability.

3 Results

3.1 First Set of Results

Experimental Setup

We evaluated our method on CUB dataset [9] which contains 11,788 bird images of 200 categories. Also, we used natural language captions provided in [3], where 10 individual sentences were annotated for each image. For evaluation, we compared our generated images to the images produced by the TAGAN researchers [4]. In particular, we adapted the code from SISGAN[5] to construct our generator. We fine-tuned the weight of the reconstruction loss to produce the best results. The original researchers used a value of 10 for the reconstructive loss. Once the generator was complete, we tested our image encoder and decoder. This was a test to determine if our implementation was correct. Once the generator was complete, we implemented the discriminator for the network using the supplementary materials provided in [4].

Figure 3: **TAGAN** (left) **Ours** (right)

Qualitative Results

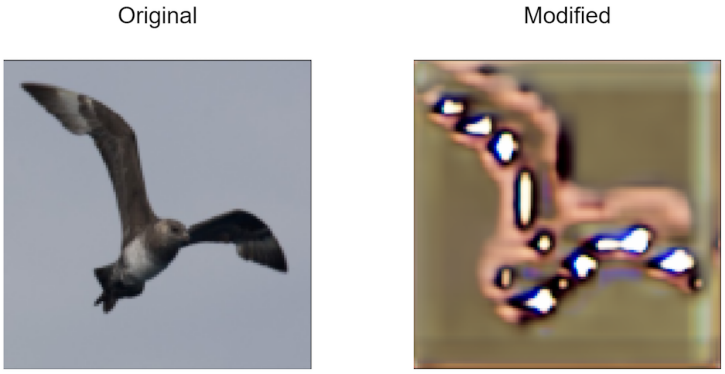
When we integrated both the generator and discriminator, we trained our network for 100 epochs and we used a reconstructive loss of 10. However, we were unable to train for 100 epochs because Google Colab would kick us out after a certain period of time. With a reconstructive loss of 10, we were able to get some image modification, but the images generated were very low quality. We experimented with a few values for our reconstructive loss, but we decided to use a value of 1000 to generated images that were closer to the original. We were able to train for 30 epochs. Figure 3 compares TAGAN to our implementation. The birds on the right look nearly identical, but the visual attributes of the generated bird image have not been modified. TAGAN trained their network for 600 epochs, and we trained our network for 30 epochs to produce the image in Figure 3. The high weighting of 1000 for the reconstructive loss also had an effect on the images produced. We trained our network for a maximum

of 70 epochs, but the images produced still did not have any changes to the visual attributes. This indicated that something was wrong in our implementation.

We discovered that one of our measures of loss that we use to train the network was coded incorrectly and this caused generated images to look very similar to the original images. Once this issue was fixed, we were only able to train the network for 10 epochs due to instability in our network. Our next goal was to understand why our network was unstable and to compare our implementation to the original. This would allow us to see what changes we needed to make in order to successfully modify bird images using natural language.

3.2 Results after changing loss

After removing the redundant sigmoid function in the unconditional discriminator and changing its loss from a sum of sigmoid outputs to binary cross-entropy, the discriminator's instability seemed to be largely fixed: its outputs no longer diverged to either extreme, at least not gradually. However, the discriminator still had its issues: during training, it would occasionally (without warning) start outputting NAN for every input, and would continue doing so indefinitely, meaning that training would have to be restarted from scratch (see discussion for more details). Combined with the fact that Colab's session timeouts often allowed us to only train a single-digit number of epochs at a time, this issue completely precluded the possibility of exactly replicating the original paper's 600-epoch training procedure in a short time span. Nevertheless, we eventually managed to train one model to approximately 150 epochs and another model to 100 epochs using the same hyperparameters as the paper (in particular, relatively low reconstructive loss weighting), and show a sample results from each in Figures 4-6.



This is a bird that has a black head and beak,
and a mix of black and dark brown feathers on its body and wings

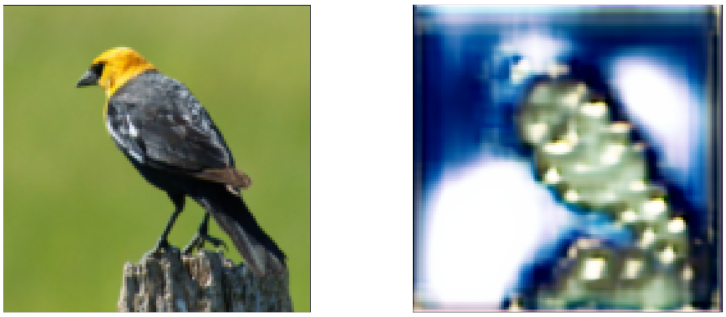
Figure 4: A representative result of one model after training for 100 epochs.

It is clear from these results that, at least by 100-150 epochs, the generator is still a long ways off from producing even realistic-looking images, much less accurate and natural image modification. However, it is not clear that this is an issue of training time. Figures 7 and 8 show images from each model created after only 10 epochs.

An eager observer might posit slight progress from these images to the later ones, but any such improvement is extremely slight. It would be nice to test the improvement, or lack thereof, of the model using quantitative measures such as loss; however, the training structure of a GAN makes such standard approaches somewhat problematic, as the loss of each part of a GAN does not necessarily go down with time, since the generator's loss is largely inversely related to the discriminator's loss. Thus, we must rely on qualitative measures to determine how well the GAN is training, and at present, the answer would appear to be: not well.

Given the seemingly reasonable hypothesis that it would expedite the training of both the generator and the discriminator for the generator to learn to produce at least realistic images quickly, even if they were not yet modified, we also tried training multiple models using much higher weighting (100-200) of the reconstructive loss; these models started outputting NANs after only 20-30 epochs, and by that point had reached a similar status to

Original Modified



Bird has brown body feathers, brown breast feathers, and brown beak

Figure 5: A representative result of another model after training for 100 epochs.

Original Modified

A bird with a slender black beak, white throat, with a black cap and cheek, and white eye ring and superciliary with pale yellow patches on wings and belly.

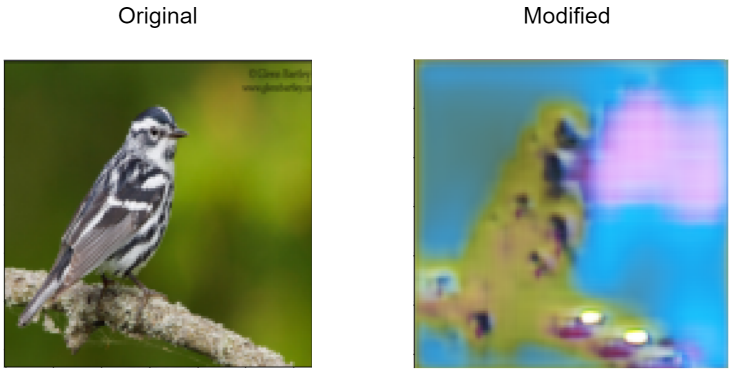
Figure 6: A representative result of one model after training for 150 epochs.

Original Modified

This particular bird has a belly that is white and tan on the breasts with black spots

Figure 7: A representative result of one model after training for 10 epochs.

the models described in the first section of the results, namely, reproducing images well, but not yet modifying them.



This tan-colored bird has a long tail, a small head with brown stripes running to the back of the head, and a small pointy beak.

Figure 8: A representative result of another model after training for 10 epochs.

4 Discussion

As noted in the previous section, the results we got were significantly different from the results of the paper. We were never able to fully replicate the study due to a number of issues that arose. While some issues were able to be corrected (such as the errors in loss functions mentioned earlier) we also had errors that we could not fully correct. One of these was that the conditional loss and unconditional loss began outputting values that were either too small or too large (also known as NAN, Not A Number) to have an effect on the network training. Despite retrieving and retraining the model when this occurred, we were not able to ever fully correct this error and train the network in the manner of the original paper. There are several possible reasons that could have led to the difference between the results of the original study and the results that we got (i.e. inability to train the network for an extended period of time), and we will discuss possible causes of and solutions to these issues (as well ways of improving the replication process) in the remainder of this section.

Of course, if the issues we have encountered are due to bugs in our code, then we could certainly solve them with more time and more debugging. It is therefore worthwhile to discuss one primary mistake our group committed during the replication process which significantly slowed the creation of an almost-usable model: the overall approach we took towards debugging. Towards the latter half of the project, as we designed most of the model, we generally did little to no testing of our code during the initial phases of coding, rather waiting until the entire model was complete to start debugging. The reason we took this approach was that we believed that any individual part of this model would be hard to test without the rest of the model finished, particularly including the residual blocks of the generator and each component of the discriminator. Hence, we had only tested the image encoder and decoder while coding and left all other parts untested until the model was finished. The general result of this process was a massive, monolithic debugging process towards the end of the project, where every team member had to debug not only their own code but also everyone else's - unsurprisingly, this fact increased the difficulty of debugging even simple dimensionality errors by an order of magnitude. In the future, this coding process could be improved by testing our model components for correct dimensionality while coding: we could generate random tensors with appropriate dimensionality so as to mimic the training process and eliminate dimension errors during the coding phase. This would allow each individual of the group to debug their own code, while it is still fresh in their minds, thus significantly speeding up the overall coding process.

Not all of the issues we encountered were of our own making, though; we also ran into issues with the paper that made reproducing the code and results significantly more difficult. These issues primarily took the form of simple typos, many of which were easily corrected. In equation (7) of the paper, for example, the authors suggest that the generator should use $\log D(\mathbf{x})$ as part of its loss, where D represents the unconditional discriminator and $\mathbf{x}$ represents an image input; however, there is no reason for the generator to concern itself with the unconditional discriminator's opinion of a given real image, rather than the discriminator's opinion of the generator's outputted

images. This error is fixed easily by changing $\mathbf{x}$ to $G(\mathbf{x}, \hat{\mathbf{t}})$, i.e. the generator's modified version of $\mathbf{x}$ given new text $\hat{\mathbf{t}}$.

There was one typo, though, that could not be fixed easily. The paper's supplementary materials suggest a softmax on the 1x1x1 output of the unconditional discriminator; however, using a softmax on a singleton tensor simply outputs 1 no matter the value of the tensor. This softmax would cause the unconditional discriminator to output 1 for every input. In this case, it is not clear what the authors' intent was. The singleton equivalent of a softmax is in many ways the sigmoid function (the former is generally used for categorical classification tasks, whereas the latter is generally used for binary classification tasks such as the discriminator's), and so we elected to replace the softmax with a sigmoid. As it turns out, this is indirectly what the authors did in their code; however, we only realized this after looking at their code, but we also had to slightly change the loss function to account for this sigmoid. This ended up representing a substantial stumbling block when training the model. If the authors had clearly identified their activation function from the beginning, we would have had more immediate success. Similar and even more extreme examples, regarding implementation details, likely exist in many of the machine learning papers. This presents a sobering prospect for would-be replicators in the future.

Our final attempt to find errors in our network involved substituting parts of the original network code into our code. The intention was to discover if the instability of our losses was caused by issues in our model or the original model (and if our model was at fault, which parts of the network specifically were causing problems). We found no significant differences in network architecture between our model and theirs that would've been likely to lead to the extreme instability of our model. We did find, however, that the original authors used a cross-entropy loss instead of a summation of raw logit values from the discriminator. While this detail is likely not the sole cause of our network instability, it is another example of an instance where the network description was either under-specified or entirely incorrect. This makes proper replication incredibly difficult and time-consuming to complete.

Because we were unable to ever successfully produce the results of similar quality to the original study's, we also attempted to simply run the code published by the authors. Due to hardware limitations, we had to use our own code to clean and organize data, but the architecture of their network was unchanged. Unfortunately, even when running their code, we were unable to reproduce the authors' results. We encountered runtime errors which impeded the network's functionality and could not be fixed without making significant changes to the network code. This was (obviously) beyond our goal of simply running the authors code. Hence, it was impossible for us to replicate the work from the paper even directly using the model given by the original authors. Thus, in terms of replicability of the study, it was not only extremely difficult to replicate the text-adaptive generative adversarial network from the paper the author published describing the work, it was also nearly impossible using the code produced by the authors themselves.

5 Conclusion

Our major goal of this project was to replicate the text-adaptive generative adversarial network proposed in [4]. We were only partially successful at this, as we could not create a network that was as capable of modifying images as the original paper. There were a variety of issues that led to our differing results, the most severe of which were incorrectly specified details of the network architecture in the original paper and difficulties in accessing the computational resources necessary to complete a full replication of the original training.

We were limited in our ability to fully replicate the original study, and future work should seek to address this. Ways in which this could be completed include the use of the Oxford-102 dataset to train and test the network and using additional quantitative and qualitative measures to measure the success of the network. We were limited in our ability to do this due to time and budget constraints, but these measures are still necessary for a full evaluation of TAGAN. Finally, and perhaps most importantly, we were also limited in our abilities to train our network as extensively as the original authors did, and future replication studies should address this as well.

In terms of replicability, our study has reiterated the importance of making machine learning studies more accessible and clearer to understand in addition to making code available publicly. Without being able to properly replicate the TAGAN network, we are unable to accurately evaluate the claims of the authors regarding the innovations of TAGAN. This makes further advances in GANs and image modification in general more difficult and costlier.

References

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In Proceedings of NIPS, pages 2672– 2680, 2014.

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

[3] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text-to-image synthesis," in ICML, 2016.

[4] S. Nam, Y. Kim, and S. J. Kim. Text-adaptive generative adversarial networks: manipulating images with natural language. In Advances in Neural Information Processing Systems, pages 42–51, 2018.

[5] H. Dong, S. Yu, C. Wu, and Y. Guo, "Semantic image synthesis via adversarial learning," in ICCV, Oct 2017.

[6] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," arXiv preprint arXiv:1607.04606, 2016.

[7] 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," in ICCV, 2017.

[8] A. Cook. (2017). Global Average Pooling Layers for Object Localization. [Blog] Available at: https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/ [Accessed 16 Oct. 2019].

[9] C. Wah, S. Branson, P.Welinder, P. Perona, and S. Belongie, "The Caltech-UCSD Birds-200-2011 Dataset," Tech. Rep. CNS-TR-2011-001, California Institute of Technology, 2011.

[10] M.E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in ICCVGIP, Dec 2008.

[11] T. Xu, P. Zhang, Q. Huang, H. Zhang, Z. Gan, X. Huang, and X. He. (2018). "Attngan: Fine-grained text to image generation with attentional generative adversarial networks," in Conference on Computer Vision and Pattern Recognition.'

[12] Open Science Collaboration. (2015). Estimating the reproducibility of psychological science. Science, 349(6251).

[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[14] E. Grave*, P. Bojanowski*, P. Gupta, A. Joulin, T. Mikolov, Learning Word Vectors for 157 Languages