Generating Animated Variants of Real-World Images using Generative Adversarial Nets

Anjali Yadav, Prerit Gupta, Rajarshi Saha, Ritwika Chowdhury and Yash Agarwal
Indian Institute of Technology Kharagpur
West Bengal, India
{anjaliy119, gprerit96, rajarshisaha95, ritwikachowdhury1995, yashaga98}@gmail.com

Abstract

Generative Adversarial Networks (GANs) have been extensively deployed for generating samples from a particular probability distribution. In this work, we present a specific application of image-to-image translation using GANs, namely converting a real-world image to its animated variant. Our work has potential applications in character creation in animated movies, generating parody-images corresponding to a real-word scenario, etc. The choice of GANs to do this job lies in its innate capability of imitating probability distributions rather than perform a one-to-one mapping between pairs of images. This is primarily crucial for our application because of the unavailability of paired datasets for training. In our proposed approach, we resort to training our GAN architecture using unpaired image datasets. Moreover, we appropriately modify our cost functions to take into account the subtle nuances of our application domain. The detailed implementation and architecture have been presented, along with generated sample anime images.

1. Introduction

Generating animated versions of real-world images is not only an extremely interesting research problem but also finds applications in digital media such as computer graphics for characters in animated movies, creating parodies and comical printed media, use in story-telling for education and a host of other applications. The real world images could possibly be constituted of genuine political scenarios and could be used to generate mocking depictions of the same. These depictions could be used in advertisements or newspapers so as to be more effective in creating awareness about any agenda. Apart from all, these animated representations can provide inspirations to artists to create more refined versions.

Today, generating animated characters for computer graphics take up a lot of time. To put thing things into a little perspective, on an average, it takes about a week to create even a 30 second animated clip. Blockbuster movies like *Toy Story* took years of effort to do the final rendering. It becomes essential to capture the subtle intricacies like facial characteristics while expression of emotions, the way the hair of a character blows in the wind, and several other nuances which are not really of any concern when a real-life movie is produced. It would give a major boost to the animation industry if it were possible to capture these details by converting image frames in real-world videos to animated ones. Our work is an approach to demonstrate that Generated Adversarial Networks might possibly be employed for this application.

One of the major issues that comes into picture while applying deep learning methods for generating animated variants of real-world pictures is the unavailability of any dataset with paired images to train our model. It is unreasonable to expect that such a dataset would be available because creation of any character is a work of art and even generating such a dataset can be prohibitive. Generative Adversarial Networks naturally comes into picture in such a scenario because of their innate capability of imitating distributions, rather than trying to model an input-output mapping which is generally done while training conventional neural networks. Moreover, Convolutional Deep Networks (CNNs) have shown immense success in image processing applications primarily because of their local activations. Hence, GANs which have their generator and discriminator networks comprised of deep CNNs are well suited for image-to-image translation tasks. As has been elaborated in the literature survey, a lot of work has already been done for image translation tasks and a lot of our architecture is motivated by them.

The report is organized as follows. In Section (2), we discuss the existing work that has been done in related applications of image-to-image translation tasks. In section (3), we discuss the relevant theoretical background of our implementation. We also discuss our proposed modifica-

tions to the loss function and justify how they influence our output results. In Section (4) we discuss in detail how we implemented our GAN architecture along with its salient features. In Section (**??**), we explain our training and experimentation methodology. In Section (6), we show some of the test output results of our trained model. Finally, in Section (7), we conclude our report with suggestions on how our model could be improved to provide directions for future work.

2. Literature Survey

Several previous works have addressed the problem of animated and comical image generation. Hawkins et al. [2] developed an approach for creating computer-generated humorous comics. But, the task of transforming a real world image to its animated variant is recently explored with the advent of deep learning techniques.

Chen et al. [1] presented a dedicated Convolutional Neural Network (CNN) to transform photos to comics which works by transferring style from a comic image to the content image. However, this approach fails to learn the complete comic style as one single image cannot capture the complete pixel distribution of a comic style. Hence, a better style transfer technique is needed which transfers style from a distribution learned from a large set of images.

In 2014, Goodfellow et al. [3] presented the basic architecture of Generative Adversarial Networks in which input of the generator network takes random noise and converts it into the target distribution. Since then, GANs have shown remarkable results in generating a particular distribution and its variants have been used in numerous applications.

Zhu et al. [9] proposed an approach to translate an image from source domain to target domain in the absence of paired examples using Cycle-Consistent Adversarial Networks. Although such an approach solves the problem of obtaining paired dataset, but the obtained results show that this method severely fails on tasks requiring geometrical changes.

Yifan Liu et al. [6] considered the problem of coloring black and white sketches using conditional generative adversarial nets in a way so that the choice of colors chosen gives the sketches an appearance of being an animated image. It is pretty convincing to expect that real-world images and animated images have starkly different color schemes, with animated images having a distinct contrast between colors of adjacent regions with minimal light-and-shade effects. Such details were taken into account in their GAN architecture and Loss functions, which even we have considered in our problem.

Philip Isola et al [7] addressed a more generic problem of image-to-image translation using conditional GANs where the application domain is pretty generic. They have shown that the GAN architecture can implicitly learn the loss functions themselves without having to tailor them specific to each application which requires expert knowledge. However, we realized that such a generic framework is not really the best option for us because the results in this case can depend a lot on the quality of the data set available, and in any case, results can definitely be improved by suitably customizing the loss functions specific to our application. Hence, even though we have a lot of the architecture for generator and discriminator common to theirs, we have also made extra finer contributions particular to our problem of generating "animated" variants.

3. Theoretical Background

3.1. Generative Adversarial Nets

Generative Adversarial Nets (GANs) is a recently proposed architecture for estimating generative models via a two-player minimax game, in which we train two models: a Generative model (G) that captures the data distribution, and a Discriminative model (D) that estimates the probability that a sample came from the training data rather than G. Mathematically, this is represented as:

$$\begin{aligned}\min_G \max_D V(G,D) &= \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] \\ &\quad +\mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(1-D(G(\mathbf{z})))]\end{aligned}$$

In the optimal solution of this game, G recovers the training data distribution and D outputs 0.5 for any input. In other words, the discriminator is effectively fooled into mistaking the fake samples from the generator as the ones coming from the real data set.

The RHS of the equation above is the *objective function* of the two-player game, which the discriminator tries to maximize and the generator tries to minimize. The theoretically optimal solution is a saddle point which is maxima w.r.t. the discriminator weights and minima w.r.t. the generator weights. Alternatively stated, we perform a *stochastic descent* with respect to the generator weights and a *stochastic ascent* with respect to the discriminator weights. The same problem can be reformulated by re-defining our loss functions separately for the Generator (ζ_G) and Discriminator (ζ_D) so that it is a minimization problem w.r.t. both of them. They are re-defined as below:

$$\zeta_G = V(G,D) \tag{1}$$

$$\zeta_D = -V(G,D) \tag{2}$$

Here, $V(G,D)$ denotes the objective function. Note that only the signs have been flipped. Maximizing something is the same as minimizing its negative.

A common variant of this vanilla GAN architecture are the Conditional Adversarial Nets (cGAN), in which both the generator and discriminator are conditioned on some extra information. In our case these are the images from the real-world image dataset. The probability distribution now being estimated by the generator is conditional distribution (conditioned on the inputs). This is relevant to our application because we actually want to transform an image into its animated variant, rather than just generate arbitrary animated images. So there must be a good deal of structural similarity between the input real-word image and the output animated variant. This structure is captured by the *conditional* version of the probability distributions being imitated. The min-max game is modified as below:

$$\min_G \max_D V(G,D) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x}|\mathbf{y})] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z}|\mathbf{y})))]$$

In our setting, x represents an image drawn from the *animated* manifold, y is the conditioning input which denotes an image drawn from the *real-world* manifold. Finally, z is random noise which must necessarily be inputted to the generator in any GAN architecture so as to ensure stochasticity in its outputs.

In our work, the task of the generator is not just to fool the discriminator, but also to take into account subtle nuances of animated images (such as predominant edges) as well as ensure that the generated animated image is "close" to the conditioning real-world image. Another benefit of the separate formulations of loss functions for generator and discriminator is in visualizing how each of them can be individually modified to consider the above-mentioned minute details. This is explained in the next subsection.

3.2. Variants to the Vanilla Loss functions

We consider several variants of the vanilla loss functions for generator and discriminator to take into account the nuances of our problem. In their expressions, **x** is a sample image from the animated dataset. **y** is a sample image from the real-world dataset, and **z** is random noise. The losses are detailed as below:

3.2.1 Feature/Content (Coarse) Loss

It is introduced to retain high-level information present in the images, which are extracted using VGGNet. The subscript l denotes the layer of the VGGNet from which features are extracted. Expectation is taken over $x, y \sim p_{data}(x, y), z \sim p(z)$. Contributing to generator loss again, it is expressed as:

$$\zeta_F = \mathbb{E}[\parallel VGG_l(\mathbf{y}) - VGG_l(G(\mathbf{x}, \mathbf{z})) \parallel_1] \quad (3)$$

3.3. VGG-16 Feature Embedding for Content Loss

The animated images generated through our network should preserve the semantic content of the actual input images, which is the main reason of using the coarse loss described in section (3.2.1). Pre-trained VGG-16 is used with our network to encode the complex semantic information as well as perceptual similarity between the input images and their corresponding generated animated images in feature form.

VGGNet was preferred for extracting high-level information because it has proved to be pretty effective in image recognition and labelling tasks, for which high-level information is pretty relevant. Conv 5-3 latent features embedding is used to calculate the l_1 Coarse loss function between animated images and generated variants of real images instead of pixel-wise losses.

The VGG-16 network is a 16 layered architecture consisting of convolutional layers with filter size of 3x3, max pooling layers with filter size of 2x2, and 3 fully connected layers at the end. Feature embeddings of layer Conv 5_3 are taken for content loss calculation. As a standard VGG-16 network takes input an image of size 224x224, the original images and the animated images are resized before feeding them to the VGG network.

3.3.1 Edge-Enhancing (Fine) Loss

This is introduced to ensure the predominance of sharp boundaries in animated images. Contributing to discriminator loss, it is expressed as below. Here, **e**(**x**) is the edge-smoothed version of animated image **x**.

$$\zeta_E = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log(1 - D(\mathbf{e}(\mathbf{x})))] \quad (4)$$

The primary intuition behind this is that in the absence of sharp distinctive edges, we want the the discriminator to classify the image as "*fake*", i.e. not belonging to an animated manifold. Hence, from our dataset of animated images, we synthetically create another dataset of "*edge-smoothened*" images, which is denoted as $e(x)$ in the equation, and we consider the form $(1 - D(e(x)))$ within log. This pre-processing which is elaborated in the next subsection (3.3.2) is mainly a *Data Augmentation* technique.

3.3.2 Pre-Processing: Edge Smoothening

In order to obtain the fine loss as described in Section (3.3.1), the edge-smoothened versions of the animated images are obtained and stored as a separate dataset as shown in Fig (1). Canny Edge Detection algorithm is used on gray-scale Animated Images to detect edges for creating edge-smoothed versions. After identifying the edges, the binary image with detected edges is dilated. Gaussian filter

smoothing is applied with a kernel size of 5x5 on the pixels of original image which were detected by dilated edges.

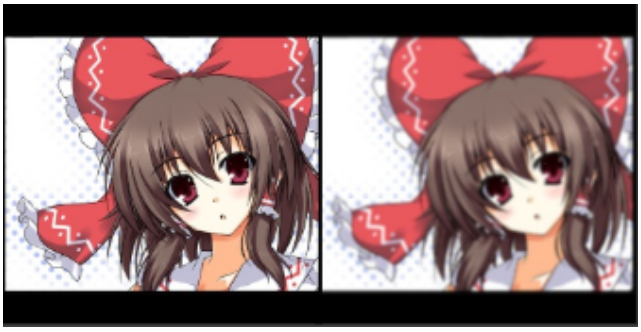

Figure 1: Edge Smoothing of Animated Image where original animated image is on the left and its edge-smoothed version on the right

The major steps in Canny Edge Detection include application of Gaussian filter to smooth the image in order to remove noise. The intensity gradients in the image are then computed and a non-maximal suppression step follows to get rid of spurious response in the detected edges. This is followed by a double-thresholding step to detect true images. The edge is tracked by the corresponding hysteresis loop and the detection of edges is finalized by suppressing all the other edges that are weak and are not connected to strong edges.

3.3.3 Final Loss Expressions

The final loss expressions for generator and discriminator are weighted combinations of the above introduced losses. The *Generator loss* is:

$$\zeta'_G = \omega_0 \zeta_G + \omega_F \zeta_F \tag{5}$$

Similarly, the *Discriminator loss* is:

$$\zeta'_D = \omega_1 \zeta_D + \omega_E \zeta_E \tag{6}$$

The training objective is to minimize both of them. The individual contribution weights are hyper-parameters that need to be tuned.

4. Detailed Implementation

4.1. Animated GAN Architecture

The architecture for Animated GAN is inspired from the GAN architecture used for super-resolution task [5] by Ledig et al and the work of Isola et al on Image-to-Image translation [7]. Refer to Figure 2. The basic architecture used consists of two parts: a) The Generator Network G and b) The Discriminator Network D. We have incorporated the changes required in the architecture to train it for our purpose. The input images are fed to G which then outputs the corresponding animated images. These images are fed to the VGG-16 network and the coarse loss is calculated for training the Generator Network. The animated image generated by G, original input animated image and the edge-smoothed animated image are all fed as inputs to the Discriminator Network one by one. The discriminator D is used to give a probability value where 1 indicates that the generated animated image is a real animated image (not a edge smoothed animated image). These probabilities are used to estimate the fine loss which is further used to train the network using back propagation.

G begins with a convolutional layer having filter size of 7x7 with stride of 1 followed by another convolutional layer having a filter size of 3x3 and stride of 2. This layer helps to encode the local features in the image and transform it to a lower dimension. The core of the architecture consists of 5 residual blocks. The final layers of G consists of up-sampling (or using an abuse of terminology as is generally done, *de-convolution*) blocks with stride of 1/2 which is used to reconstruct a similar input dimension image from the features encoded by the previous layers.

D consists of a series of flat convolution layers with a stride of 1 stacked with down-convolutional layers with stride of 2. Batch normalization layers and Leaky ReLu layers with a negative slope of 0.01 are added at the end of the convolution stages. The salient features of our architecture are:

Residual blocks: The 5 residual blocks use skip connections to modify the function to (F(x) + x) form, which mainly solves the vanishing gradient problem of deep networks and high training error due to more layers.

Unpaired images for training: It is very difficult to collect datasets of paired real life images and animated images, so we have ensured the training of our network with unpaired images by using particular loss functions and architecture. Unpairing of dataset is feasible in GANs as they learn mapping between two different distributions rather than the transformation between two specific images.

Skip Connections: The Generator architecture is largely based on *U-Net* architecture, a salient feature of which is long-range skip connections (apart from those already present in the residual blocks). While upsampling in the CNN after the bottleneck layer, the subsequent layers are stacked along with their mirror images from the input side. In other words, there is a skip connection from the i^{th} layer to $(n-i)^{th}$ layer. In a standard Encoder-Decoder network, much of low-level information is lost while everything goes entirely from the input to the output and through the bottleneck layer. Apart from tackling the

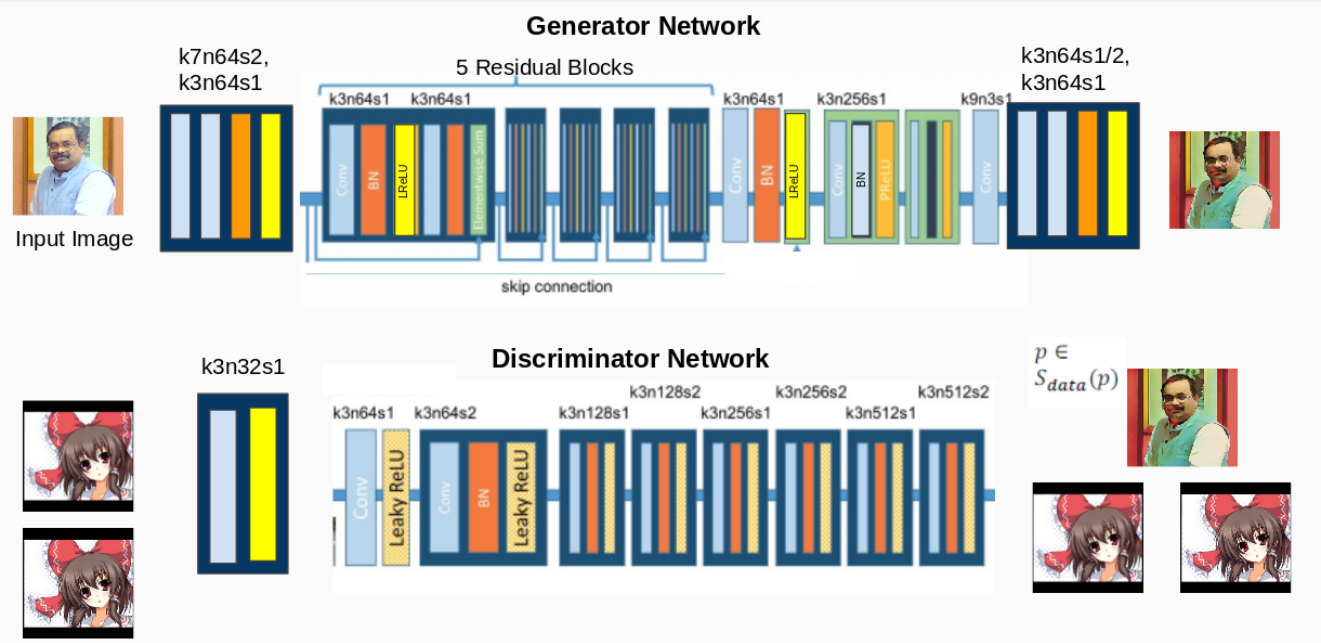

Figure 2: Proposed Architectures of generator and discriminator network in which

vanishing gradient problem, these skip connections shuttle the low-level information directly across the net, which plays a major role in faster training.

PatchGAN Discriminator: The *PatchGAN discriminator* architecture classifies the image as being *real* or *fake* for each patch of the image, and the results are finally averaged. A new hyper-parameter: the *patch size* comes into picture here.

5. Experiments

5.1. Datasets

Datasets are taken from two different sources and images are resized to 256 x 256 dimension:

– *Real World Images*: 6K images are randomly sampled for training from the freely available Celebrity Attributes Dataset(CelebA) which contains more than 200K images of large-scale face attributes of the dimension 178 x 218

– *Animated Images*: 6K images are randomly sampled from the freely available *Danbooru-2017* dataset which contains around 300K images of the dimension 512 x 512 from a large-scale crowd sourced data.

The above datasets are carefully selected to ensure that the images are of higher dimensions, so that re-sizing them to 256 x 256 doesn't lead to blurring. For style transfer learning, getting paired dataset is a challenging task and also requires expensive manual work. However, the advantage of our network is that paired images aren't required. This makes dataset selection for training easier than previous techniques.

A distinguishing feature of animated images is that they have sharp edges. So it is important that our network should be able to differentiate the images based on this feature.To ensure this, we have also created a dataset of images with smooth edges by pre-processing our anime dataset as described in Section (3.3.2). These boundary-smooth versions of animated images form the third dataset. So now while training our network we will use all three sets of data.

5.2. Training Parameters

We have implemented our architecture in Tensorflow and trained it on Nvidia Quadro M4000 in Computer Vision Lab server. The model has been trained by backpropagation on 30 epochs using Adam Optimizer with a learning rate of 0.0001, fine loss weight of 10 and a batch size of 1(due to limited GPU memory). The model parameters have been saved after every 1000 iterations.

6. Test Results

The Fig. 4 shows the generated image after the GAN architecture is trained for 100 epochs which corresponds to the real image shown in Fig. 3. The fine edges which is required in an animated image are retained well as well as the content of the image is preserved. Although some color changes in some localized regions are dominating like be-

low the chin, moustache and hair, but overall the network is successfully able to generate qualitative animated images which can be clearly seen through visual analysis. A bunch of other images from different places at II Kharagpur campus as shown in Fig. 5 are passed through the trained generator and the output images shown in Fig. 6 are good animated representations of their real analogues.

6.1. Comparison with state-of-the-art

There have been some work on real-life photo to animated image conversion already existing in literature. Several of them have already been mentioned in section (2). Although the results obtained by those approaches are also pretty impressive, there are some distinguishing features in our approach.

Figure 3: Input Real Images to the trained Generator

A company *BeFunky Inc* [8] in 2007 (Note: GAN was proposed much later in 2014), launched with a single service in which the customer would mail them their photo, and one of their artists would turn the photo into an animated image by hand and mail it back to the customers. They typically used photo-editing software like Adobe Photoshop, etc. It became so popular that they automated their service and made it into an online software. However, their conversion time is noticeably significant in comparison to ours. Their conversion style is hard-coded. Their conversion rules have been programmed with years of effort of those artists and cannot adapt itself if we wanted a different style. In our approach, simply by retraining our model with another *animated dataset* of appropriate style, the translation model can be adapted. Although training time is pretty high, it is nothing in comparison to years of effort put into hard-coding the image conversion rules.

Figure 4: Generated Image (Portrait) at epoch 50

Yanghua et al [4] have addressed the problem of anime character generation using GANs and they have demonstrated promising progress in this area. They have included several steps and loss functions which can definitely also be applicable in our context as well. However, they only considered generating anime faces and moreover, they don't consider the conversion from a real-world image. In that respect our application is different form theirs as it captures scenes other than portraits too.

7. Conclusion and Future Work

To conclude our report, we would like to point out that the primary takeaway from our project is the fact that Conditional Generative Adversarial Nets with suitably modified loss functions can prove pretty effective in generating animated variants of real-world images. The testing is seen to be instantaneous. It can possibly be employed in converting each frame of real-life video to convert it into an animated video. This can play a significant role in speeding-up computer graphics application.

As is evident, our proposed architecture is a pretty naive and generic one, with very little modifications. A lot more work can be done to suitably customize our architecture. For instance, more pre-processing can be done with available database to train different facial-features in animated characters. In our work, we have just considered the edge-enhancing loss which is specific to our problem. Several more loss functions can be suitably defined to penalize light-and-shade variations and choose a suitable color contrast. It seems that in certain cases, the pixel values saturate and they give undesirable artifacts. This needs to be handled as well. Finally, speculating far ahead, we can also

Figure 5: Input Real Images to the trained Generator

do further post-processing, drawing ideas from neural style transfer and related work to improve our results. Moreover, right now, we are just judging the quality of our results by visual inspection. We need to come up with suitable performance metrics which can accurately judge the quality of our results.

References

[1] Y. Chen, Y. Lai, and Y. Liu. Transforming photos to comics using convolutional neural networks. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 2010–2014, Sept 2017.

[2] P. P. Daniel Hawkins, Devin Cook. Comedy53: An approach for generating computer-generated humorous comics. *Proceedings 21st International Symposium on Electronic Art*, 2015.

[3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *ArXiv e-prints*, June 2014.

[4] Y. Jin, J. Zhang, M. Li, Y. Tian, H. Zhu, and Z. Fang. Towards the automatic anime characters creation with generative adversarial networks. *CoRR*, abs/1708.05509, 2017.

[5] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. P. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802, 2016.

[6] Y. Liu, Z. Qin, Z. Luo, and H. Wang. Auto-painter: Cartoon image generation from sketch by using conditional generative adversarial networks. *CoRR*, abs/1705.01908, 2017.

[7] T. Z. A. A. E. Phillip Isola, Jun-Yan Zhu. Image-to-image translation with conditional adversarial networks. *Proceedings 21st International Symposium on Electronic Art*, arxiv, 2017.

Figure 6: Generated Synthetic Images from the trained Generator

[8] T. Tatar. Befunky inc. 2007.

[9] J. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017.