SIMPSONSGAN: MODYFYING CYCLEGAN FOR UNPAIRED REAL LIFE-TO-SIMPSONS STYLE IMAGE TRANSLATION

DEEP LEARNING FOR VISUAL RECOGNITION

Alex Krogh Smythe
Department of Computer Science
Aarhus University, Denmark
alex.aarhus@gmail.com

Aron Eli Baldvinson
Department of Computer Science
Aarhus University, Denmark
aronbaldo@gmail.com

Christian Nordstrøm Rasmussen
Department of Computer Science
Aarhus University, Denmark
nordstroem92@gmail.com

December, 2021

ABSTRACT

Keywords Image-to-Image · Cycle-GAN · Deep Learning · Translation

The follow website grants access to our **Weights and Biases project**, where all the training data has been saved. The Google Colab document is included in the ZIP file.

1 Abstract

This paper tackles the unique problem of translation unpaired city images from a real-life domain to a cartoon domain in the style of 'the Simpsons'. The project is based on the official PyTorch implementation of CycleGAN, and attempts to improve upon this model by techniques of preprocessing, Total Variation Denoising and Edge-Promoting optimization. Results are presented throughout and compared to the baseline model - CycleGAN trained on two unmodified sets of training images. Both Gaussian Blur, Total Variation Denoising and Edge-Promoting loss reveal interesting results in their own rights, however all implementations struggle with generalizing to validation data, which is probably due to the size and quality of the training datasets.

2 Introduction

This paper aims to use unpaired image-to-image translation to turn real-life images of different architecture around the world into the style of the TV cartoon show 'The Simpsons'. The Simpson has a unique style, characterized by simple and abstracted elements, pronounced edge-strokes and a vibrant color scheme. This makes it a unique challenge and allows for easy perceptual evaluation of the results.

The project is built from a PyTorch implementation (Kang [2018]) of the CycleGAN model proposed by Zhu et al. [2017] from which the baseline results are derived from training the model on the two unpaired data sets: Respectively images from the Places data set and Simpons images scraped from a Google Images search. Throughout the rest of the paper, multiple steps are taken in order to increase the accuracy of the image translation. Gaussian Blur is introduced as a preprocessing technique for all training images. After that the training data is augmented using different TRS (translations, rotation, scaling) techniques. We introduce Total Variation Loss to the generator networks of the GAN in order to reduce noise in generated images. Lastly, an edge promoting loss function by Chen et al. [2018] is implemented in the generator network to optimize for cartoonish edges on the images. Results from both training and validations sets are presented and discussed in order to reflect on how they succeed in the image translation tasks.

3 Related Work

This section will present the related work that serve as theoretical foundation for this paper. It covers research on the GAN models, Total Variation Denoising and GAN-specific preprocessing.

3.1 CycleGAN

Zhu et al. [2017] propose the cycleGAN model for achieving unpaired image-to-image translations, i.e. the problem of mapping input and output images in the absence of paired image data sets. Their GAN consists of two generator networks G: X $\rightarrow$ Y and F: Y $\rightarrow$ X, and two discriminators networks D_y and D_x. The role of the generator networks is to transform images from their own domain to the target domain, while the role of the discriminator is to distinguish between real and generated images. Both types of networks have an adversarial loss function for which the generators try to minimize the loss, while the discriminators try to maximize the loss. A generator will learn to produce various fake images from real ones, that are good enough to cheat the discriminator. To ensure that the generators are consistent in how they generate fake images, meaning that images are transformed similarly every time, the Cycle Consistency Loss is introduced. The aim here is to minimize the difference between a real images and one that has been through both generators, e.g. x $\rightarrow$ G(x) $\rightarrow$ F(G(x)) $\approx$ x, as seen in Figure .

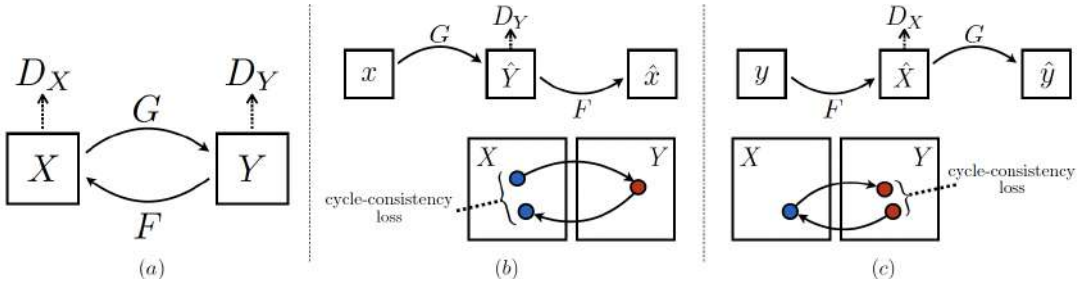

Figure 1: Illustration of Cycle Consistency Loss.

The experiments performed in this paper will take offset in the CycleGAN model, and try to improve its architecture in order to create a more convincing style transfer from real images to Simpsons-style images.

3.2 Total Variation Denoising

In 1992, Rudin et al. [1992] proposed Total Variation Denoising which can create a denoised image from a very noisy image, by smoothing out overall variation between adjacent pixels in an image. According to Wikipedia [2021], Total Variation loss is based on the assumption that random noise in an image can be reduced by minimizing the integral of the absolute image gradient of an image. Total variation denoising has the advantage that it preserves edges well compared to simpler noise reduction techniques, such as linear smoothing or median filtering. In this paper, we will experiment with implementing Total Variation Regularization into the loss function of the generator networks to create a more simple and abstracted output.

3.3 CartoonGAN

In Chen et al. [2018], CartoonGAN is presented to tackle the unique problem of cartoonization, which presents two major challenged to existing GANs: the problem of generating high-level simplifications and abstractions from an input image, and the problem of maintaining clear edges, color shadings and textures. The paper proposes two solutions to address these challenges: a Content Loss function to derive high-level features from the image, and an Edge-Promoting Adversarial Loss to preserve edges. In our paper, we will experiment with the edge promoting loss function in the context of turning real images into Simpsons-style images.

3.4 Learning to Cartoonize

Work done by [Wang and Yu, 2020] presents an approach for image cartoonization in a GAN. The authors approach was identified by observing painting and artist behaviours. By doing so [Wang and Yu, 2020] proposed three steps to decompose images to extract useful information for training a GAN framework to learn cartoonization from real-life images. As seen in Figure 2 the first step proposed is to extract a surface representation of an image, this step ensures

that surfaces are smoothed yet still maintaining global semantic structure. The second step is to extract a structure representation, this step uses sparse color blocks, emulates global content and creates clear boundaries between color changes. Lastly the texture representation implements a single channel for colored images, retaining textures and decreasing influence of color and luminance. Combining these pre-processing steps with a GAN framework training showed promising results in improving cartoonization of real-life images compared to previous techniques. For our project, inspiration was taken from the proposed surface representation and used in the pre-processing step, further details about this step is described in Section 4.3.

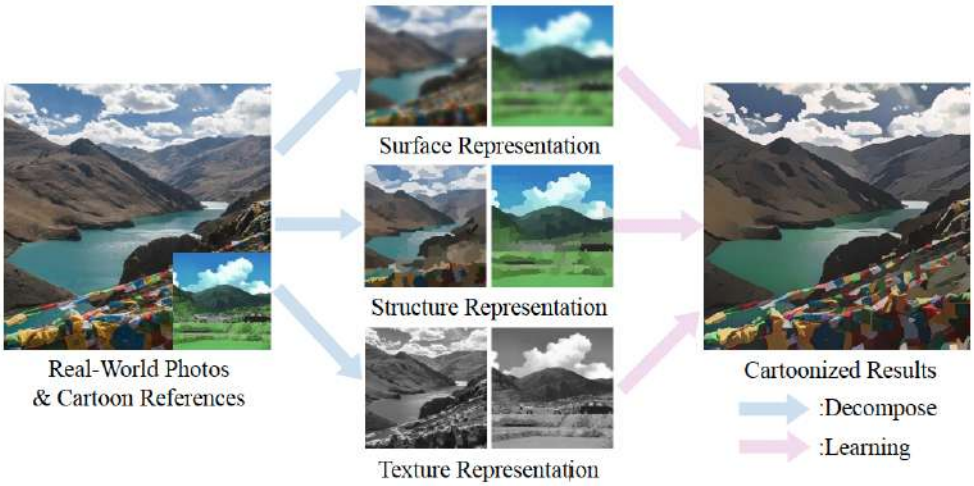

Figure 2: Illustration of how [Wang and Yu, 2020]

4 Methods

This section will present the methods used to achieve image-transformation from the domain of architecture to the domain of Simpsons-stylization. It will contain both theoretical and practical explanations.

4.1 Data set

For this project two data sets are used. The first data set is a is the buidling facade class from the Places data set by Bolei Zhou [2017]. The building facade class consist of 5000 images. The second data set is a collection of Simpsons related buildings and architectural images scraped from Google Images. A total of 794 images were scraped. These images are scaled to a uniform size of 256x256 to both match the size of the places building facade data set and based on the requirement for using the CycleGAN model. The scaling operation used on the Simpsons images result in a varied quality loss, some images suffer more than others. We used a total of 794 images for both the building facade class and Simpsons images. For the training set a total of 1288 images where used - 644 images in each domain. The remaining 150 images in each domain were used for validation, amounting to approximately a 5/1 split. The data set was then further expanded using data augmentation, which will be discussed in further detail in Section 4.4.

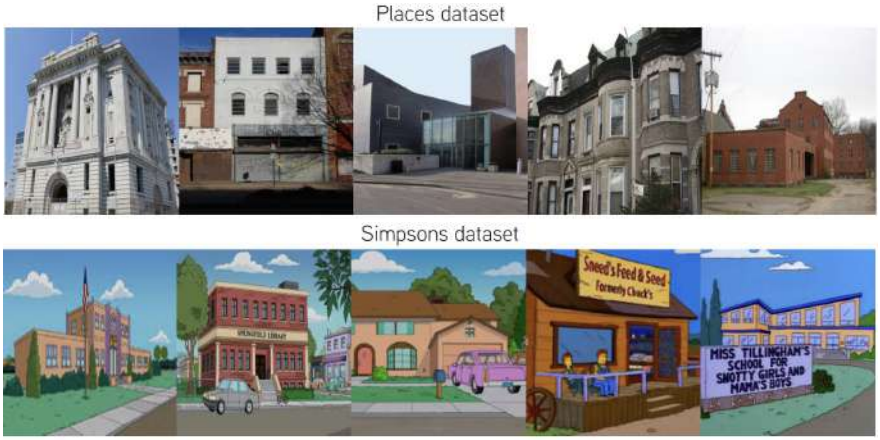

Figure 3: A collection random images taken from the original data sets used for training the model

4.2 Out of the box CycleGAN

The offset for the project was based off Kang [2018], which is the official github project for the CycleGAN paper by Zhu et al. [2017]. The github project is implemented in the PyTorch machine learning framework for Python. The project contains a tutorial in Goolge Colab Notebook, which makes it easy to use pretrained models or train new models and adjusting hyperparameter settings using simple python commands. In the very first succesful run, the pretrained horse2zebra model was mistakenly used, which produces some rather comical results that were discarded. After the initial error, the commands were altered to a new train model on the correct data. This resulted in our first model which was used as a benchmark for future comparisons, which can be seen in Section 5.1.

Since using the Google Colab Notebook for training was a 'black box'-approach, that allowed no modification of the CycleGAN model itself, the github project was later forked, and changes were made to the networks.py and cycle_gan_model.py files in the models directory. The github repositories and branches used in this project can be accessed here `https://github.com/nordstroem92/pytorch-CycleGAN-and-pix2pix`.

4.3 Gaussian Blur

The initial training of CycleGAN revealed the noisy output images, that did not converge to anything more coherent over the 150 epochs of training. For this reason, we introduced gaussian blur as a preprocessing technique for our images. Gaussian blur is commonly used for reducing noise and high frequency details in images, by running a smoothing filter over images Lahiri [2020]. The intuition behind using this technique in our case, was, that by preventing the model from adapting to the high-frequency details, we could get a more general model, that focused more on structural features of the images. Since the cartoonish style is generally characterized by simple and abstracted elements, this

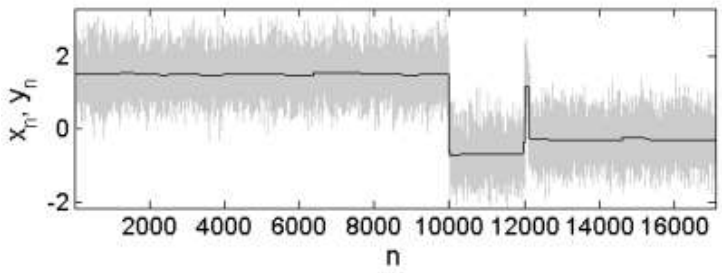

Figure 4: Example of how the total variation regularization denoises input signals
source by:`https://en.wikipedia.org/wiki/Total_variation_denoising`

further supported the choice of removing high-frequency details from training images. First, a Gaussian Blur filter size of 5x5 was used. Since this did not change the output significantly, the filter size was increased to 11x11.

4.4 Data Augmentation

After using Gaussian Blur on the data set, we introduced data augmentation to improve the results by creating new alternate images for the model to train on. Two data augmentation techniques were tested to create more training images: vertical flipping and cropping. Other techniques like horizontal flipping and rotation were excluded, since both data sets had a clear directionality, meaning that we did not want to train a model with houses and architecture being upside down. Augmenting the data set with vertical flipping revealed better results in training, but cropping was left out in later runs, as it down-sampled the resolution of the input images, that were already small (256 x 256px).

4.5 Total Variation Regularization

We used an implementation of total variation regularization that was retrieved from Lee [2017] and integrated into the backward propagation function for the generator losses. The differentiable version of the Total Variation Regularization function can be described as such:

$$V_{aniso}(y) = \sum_{i,j} |y_{i+1,j} - y_{i,j}| + |y_{i,j+1} - y_{i,j}|$$

Where the pytorch implementation looked as following:

```
x = image[:,:,1:,:] - image[:,:,:-1,:]
y = image[:,:,:,1:] - image[:,:,:,:-1]
loss = torch.sum(torch.abs(x)) + torch.sum(torch.abs(y))
```

So when an image has a shape of [batch, depth, height, width], or [1, 3, 256, 256] this means that for each indice x_i or y_j in the tensors x and y, we subtract the value of $x_i + 1$ or $y_j + 1$ with the value of x_i or y_j. When calculating the loss itself we then add the sum of the absolute value of x with the sum of the absolute value of y using torch.sum(torch.abs(x)) + torch.sum(torch.abs(y)).

Using this Total Variation Loss function we aimed to achieve denoising of real-life images, converting the broad spectrum of color-variation that they exhibit and replace that spectrum with a much simpler color variation. Essentially replacing complex colours of large surfaces with single colours much like showed in Figure 4

4.6 CartoonGAN

Chen et al. [2018] propose an Edge-Promoting Loss functions and an Content Loss function to tackle the problem of cartoonization. In our experiments, we will only focus on implementing the Edge-Promoting Loss Function. The reason for this focus is, that the combination of gaussian noise and Total Variation Loss will likely produce a similar semantic coherency to that of the that of the Content Loss function, I.e. the Guassian Blur will produce a less detailed image, and the Total Variation Loss will decrease complexity over the spatial variation.

The Adversarial Loss presented in CartoonGAN is defined as:
$L_{adv}(G, D) = \mathbb{E}_{ci} S_{data}(c)[logD(c_i)] + \mathbb{E}_{ej} S_{data}(e)[log(1 - D(e_j))] + \mathbb{E}_{pk} S_{data}(p)[log(1 - D(G(p_k)))]$
Where D is the discriminator function and G the generator function, while c is the cartoon-manifold, e is a cartoon-manifold where the edges have been blurred out in the preprocessing, and p is the real photo manifold.

In practice, this meant that a new branch of data sets needed to be created: A trainA directory holding real images, a trainB directory holding Simpsons images, and a trainC directory holding Simpsons images in which the edges had been blurred out using a script provided by Kang [2018]. Once the code was modified to the new data, the loss function could be implemented in a few lines of code:

```
loss_D_real = self.criterionGAN(pred_real, True)
loss_D_fake = self.criterionGAN(pred_fake, False)
loss_D_smooth = self.criterionGAN(pred_smooth, False)

loss_D = loss_D_real + loss_D_fake + loss_D_smooth
```

5 Results

For the structure of this chapter, we have chosen to display both a comparison of the true images to the training images as well as another pair of true images to the validation images. This is because the training images seem to be much more detailed and adopt the image translation to a much higher degree, while the validation images still show how well the model has trained itself to generalize new images. This also enables us to discuss and compare the training and validation images across the different runs. This has been further elaborated upon in Section 6

5.1 CycleGAN

Figure 5: Images taken from the initial run using CycleGAN. On the left are images taken from the results of the training set and on the right are images taken from the validation set.

The results of using out-of-the-box CycleGAN were surprisingly good. The most recognizable and convincing features about the translation are the colors. Based on Figure 5, we can see that the colors have been shifted from a slightly dull real look to a much more bright and cartoonish color palette on both the training and validation set.
On the other hand, we can also see that the colors have started to bleed into other areas of the images with patches of seemingly random changes across individual surfaces. What is very interesting here is how the model has created the characteristic blue sky with completely white clouds in all of the images. Looking at the results of the validation set and comparing them to the training set, we can see that there is not a significant differences between them, which might indicate, that the model is general enough to process the training and validation images the same.

5.2 Gaussian Blur

Figure 6: Results of training the model with Gaussian blurred images using a 11x11 filter. Images taken out of the training run (left) and the validation run. (right).

Looking at images taken out of the training set, we can see that the model successfully has disregarded some of the high-frequency details of the real images. This has resulted in less noise in surfaces and less distortion in the generated training images giving them a more uniform and cartoon-ish look, meaning that areas that previously had more details and therefore varying colors, now have less details and larger areas with the uniform colors.
The validation set however suffers even more from the spontaneous patches occurring throughout the images. One way to argue for this is the fact that while the inner details of the architecture may have been blurred out, some of the critical edges that define areas suffer from the same effect of the blur, making it more difficult to determine where a surfaces begins and where it ends.

5.3 Data Augmentation

Figure 7 shows the results of the model using Gaussian blur and data augmentation. As seen on the figure, the images made during training have achieved a look that is very comparable to the architectural impression of The Simpsons. The model has added grass, sky with clouds and avoided all of the random color patching throughout the training images.
What is interesting is how the validation images have significantly reduced the amount of color patching and made the images look much more uniform compared to the validation set using only Gaussian blur. This might indicate, that our initial data set of 794 training images was not sufficient, where as the data set after augmentation containing twice the amount of images, has made a significant improvement to the validation images. We have further discussed this in Section 6.1

Figure 7: Images taken out of the training run (left) and the validation run (right).

5.4 Total Variation Regularization

From the run where Total Variation Loss was implemented, we see a significant difference from the previous runs, specifically in the generated training results. The most meaningful change is the how colours have been significantly denoised by the loss function. Large surfaces have been uniformly colored, which leads to more desired cartoonish look and creates sharper contrasts between distinct shapes, as seen in Figure 8. Another interesting result from the training set is that the model has created large grass surfaces in front of buildings, where no grass or similar features are on the original image. This can be explained by the fact that large grass surfaces are a common theme for our Simpsons data set as seen in Figure 9. However the results of the validation run show obvious signs of over-fitting, generating results that very messy. How we would overcome this issue is discussed further in Section 6.1

Figure 8: Results of training with Total variation loss combined with Data augmentation and Gaussian Blur

Figure 9: Source: Simpsons data-set, Large green surfaces in foreground of buildings

5.5 Edge preserving loss

The initial run with the implementation of the edge-promoting loss function did not lead to any satisfactory results. While the distinct objects of the output images did appear slightly more separated from each other, the details and the colors were noisy, similar to the results without Gaussian Blur preprocessing. To make up for this, we attempted another run, in which Gaussian Blur had been applied to the trainA and trainB images. The the trainC directory was left untouched, as its purpose was to contain images in which only the edges were blurred. The generated training images mostly looked like they had been through a saturation filter and it was hard to see the Total Variance denoising working. On the validation set, the results had clearer edges and at least shared a bit of the color palette that would be expected from a Simpsons-style image. An interesting feature of this was, that the Edge-Promoting Loss was able to undo some of the blur that would otherwise be carried on to the validation images after training on images with Gaussian Blur.

The lack of uniformity in the generated surfaces is hard to explain since it had been previously achieved by the combination of Gaussian Blur and Total Variation Loss in previous runs. A reason may simply be, that edge preserving loss works better in combination with the semantic Content Loss function presented in Chen et al. [2018]. The CartoonGAN model also use a different generator model than CycleGAN that aims at only generating images with only high-level features, which probably serves as a better base for the loss functions to improve upon. Since the scope of this project was to improve upon the CycleGAN model in order to produce Simpsons images from unpaired data sets, we did not attempt to implement entirely new generator functions into the model.

Figure 10: On many of the generated images during training the colors were overly saturated while the surfaces were not as well separated as the previous training run with TV loss. The validation set hovewer had clearer edges than all previous runs, even though the surfaces were not as uniform.

6 Discussion

6.1 Difference between results of the training and validation set

It was clear from the validation images, that the models had overfitted to the training images. We speculate that the reason for this is, that the models trained for too many epochs on too little training data: 644 images - 1288 images before and after data augmentation. One of the practical obstacles was, that we did not have enough computational power to increase the training size significantly without lengthening the training time by a lot, e.g. one epoch with 1250 training images took $\sim$ 380 seconds. Another problem that was first recognized near the end of the project had to do with the image scraping from Google Images. Unfortunately, the images that had been scraped were thumbnail-quality and not the actual quality of the online images. Since training and validation images were already scaled down to 256x256, it would have been more beneficial to take offset in images of better quality. The conclusion to this is that the models could have benefited from more images of higher initial quality, along with access to more computational power in order to train the models.

6.2 Evaluation of results

Generative Adversarial Network are notorious for not having an objective way to evaluate the quality of the output. This is because there is no loss function readily available and developers are relying on the work of the discriminator to force the generator to generate the optimal quality images.

In this report, we have used a subjective visual evaluation of the images. This is because we are all very familiar with the TV show The Simpsons and therefore are able to assess the quality of the generated images by comparing them to our subjective impressions of the style. This simple technique is also one of the most used according to the quote from Borji [2018]: “Visual examination of samples by humans is one of the common and most intuitive ways to evaluate GANs.”.

This does however place us as authors in a dilemma, since we have to choose subjectively between which images to highlight throughout the report. We have tried to select a broad spectrum of images that show different results through the report, while keeping the same images in the validation set in order to show progress of the model.

7 Conclusion

In this paper we have attempted to improve on the CycleGAN model in the case of transforming unpaired images from buildings to simpsons stylization. To this end we have built upon the official PyTorch implementation from the CycleGAN paper, which also served as the benchmark run for later comparisons of results. Gaussian blur was introduced for the model to adapt to more high-level details of the training data. Data augmentation was further introduced to reduce overfitting as a result of the model learning to remember training images. Total Variation Loss was implemented in the generator networks in order to optimize for reduced noise and smooth coloring within distinct objects. Lastly, an edge promoting loss function was implemented in the discriminator network to optimize for clear edges. Overall, the models overfitted, likely due to the data set being too small. Otherwise, the best results for training images were achieved using a combination of Gaussian blur (11x11), data augmentation and TV loss, as the elements were clearly separated and uniform in color. This did however not carry over to the validation data. The model in which edge promoting loss had been implemented produced the best validation result, as the edges were pronounced with clear colored surfaces.

References

Hyeonwoo Kang. pytorch-cartoongan, december 2018. URL https://github.com/znxlwm/pytorch-CartoonGAN.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017. URL http://arxiv.org/abs/1703.10593.

Yang Chen, Yu-Kun Lai, and Yong-Jin Liu. Cartoongan: Generative adversarial networks for photo cartoonization. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9465–9474, 2018. doi:10.1109/CVPR.2018.00986.

Leonid I. Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1):259–268, 1992. ISSN 0167-2789. doi:https://doi.org/10.1016/0167-2789(92)90242-F. URL https://www.sciencedirect.com/science/article/pii/016727899290242F.

Wikipedia. Total variation denoising, September 2021. URL https://en.wikipedia.org/wiki/Total_variation_denoising.

Xinrui Wang and Jinze Yu. Learning to cartoonize using white-box cartoon representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8090–8099, 2020.

Aditya Khosla Antonio Torralba Aude Oliva Bolei Zhou, Agata Lapedriza. Places, September 2017. URL http://places2.csail.mit.edu/.

Rajarshi Lahiri. Gaussian blurring and it's importance in image processing, july 2020. URL https://medium.com/@rlahiri/gaussian-blurring-and-its-importance-in-image-processing-4be8915b85ec.

Ceshine Lee. Pytorch implementation of perceptual losses for real-time style transfer, July 2017. URL https://towardsdatascience.com/pytorch-implementation-of-perceptual-losses-for-real-time-style-transfer-8d608e2e9902.

Ali Borji. Pros and cons of GAN evaluation measures. *CoRR*, abs/1802.03446, 2018. URL http://arxiv.org/abs/1802.03446.