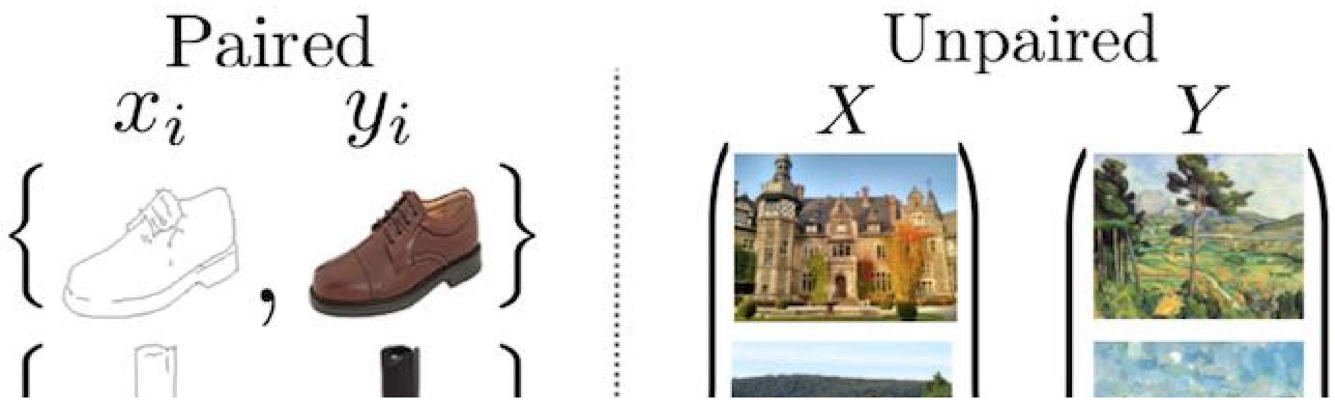
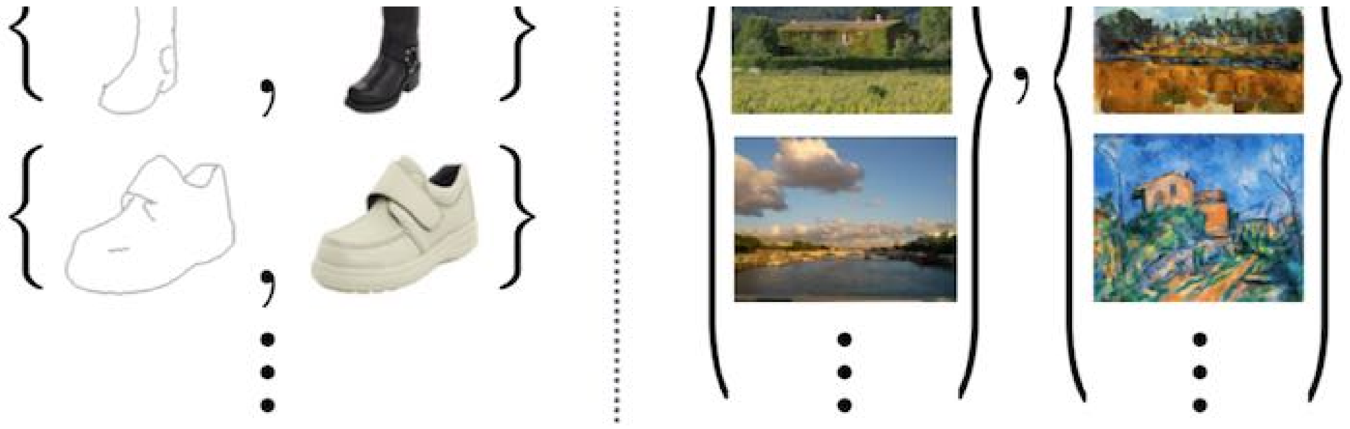
CycleGAN: Learning to Translate Images (Without Paired Training Data)

Sarah Wolf Follow
Nov 19, 2018 · 7 min read

An image of zebras translated to horses, using a CycleGAN

Image-to-image translation is the task of transforming an image from one domain (e.g., images of zebras), to another (e.g., images of horses). Ideally, other features of the image — anything not directly related to either domain, such as the background — should stay recognizably the same. As we might imagine, a good image-to-image translation system could have an almost unlimited number of applications. Changing art styles, going from sketch to photo, or changing the season of the landscape in a photo are just a few examples.




Examples of paired and unpaired data. *Image taken from the paper.

While there has been a great deal of research into this task, most of it has utilized *supervised* training, where we have access to (x, y) pairs of corresponding images from the two domains we want to learn to translate between. CycleGAN was introduced in the now well-known 2017 paper out of Berkeley, Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. It was interesting because it did *not* require paired training data — while an x and y set of images are still required, they do not need to directly correspond to each other. In other words, if you wanted to translate between sketches and photos, you still need to train on a bunch of sketches and a bunch of photos, but the sketches would not need to be *of* the exact photos in your dataset.

Since paired data is harder to find in most domains, and not even possible in some, the unsupervised training capabilities of CycleGAN are quite useful.

How does it work?

A Tale of Two Generators

CycleGAN is a Generative Adversarial Network (GAN) that uses *two* generators and *two* discriminators. (Note: If you are not familiar with GANs, you may want to read up about them before continuing).

We call one generator G, and have it convert images from the X domain to the Y domain. The other generator is called F, and converts images from Y to X.

$$G : X \rightarrow Y$$
$$F : Y \rightarrow X$$

Both G and F are generators that take an image from one domain and translate it to another. G maps from X to Y, whereas F goes in the opposite direction, mapping Y to X.

Each generator has a corresponding discriminator, which attempts to tell apart its synthesized images from real ones.

$$D_y : \text{Distinguishes y from G(x)}$$
$$D_x : \text{Distinguishes x from F(y)}$$

One discriminator provides adversarial training for G, and the other does the same for F.

The Objective Function

There are two components to the CycleGAN objective function, an *adversarial loss* and a *cycle consistency loss*. Both are essential to getting good results.

If you are familiar with GANs, the adversarial loss should come as no surprise. Both generators are attempting to “fool” their corresponding discriminator into being less able to distinguish their generated images from the real versions. We use the least squares loss (found by Mao et al to be more effective than the typical log likelihood loss) to capture this.

$$Loss_{adv}(G, D_y, X) = \frac{1}{m}\sum_{i=1}^{m}(1 - D_y(G(x_i)))^2$$

$$Loss_{adv}(F, D_x, Y) = \frac{1}{m}\sum_{i=1}^{m}(1 - D_x(F(y_i)))^2$$

However, the adversarial loss alone is not sufficient to produce good images, as it leaves the model *under-constrained*. It enforces that the generated output be of the appropriate domain, but does *not* enforce that the input and output are recognizably the same. For example, a generator that output an image y that was an excellent example of that domain, but looked nothing like x, would do well by the standard of the adversarial loss, despite not giving us what we really want.

The *cycle consistency loss* addresses this issue. It relies on the expectation that if you convert an image to the other domain and back again, by successively feeding it through both generators, you should get back something similar to what you put in. It enforces that $F(G(x)) \approx x$ and $G(F(y)) \approx y$.

$$Loss_{cyc}(G, F, X, Y) = \frac{1}{m}\sum^{m}[F(G(x_i)) - x_i] + [G(F(y_i)) - y_i]$$

$$m \; {}_{i=1}$$

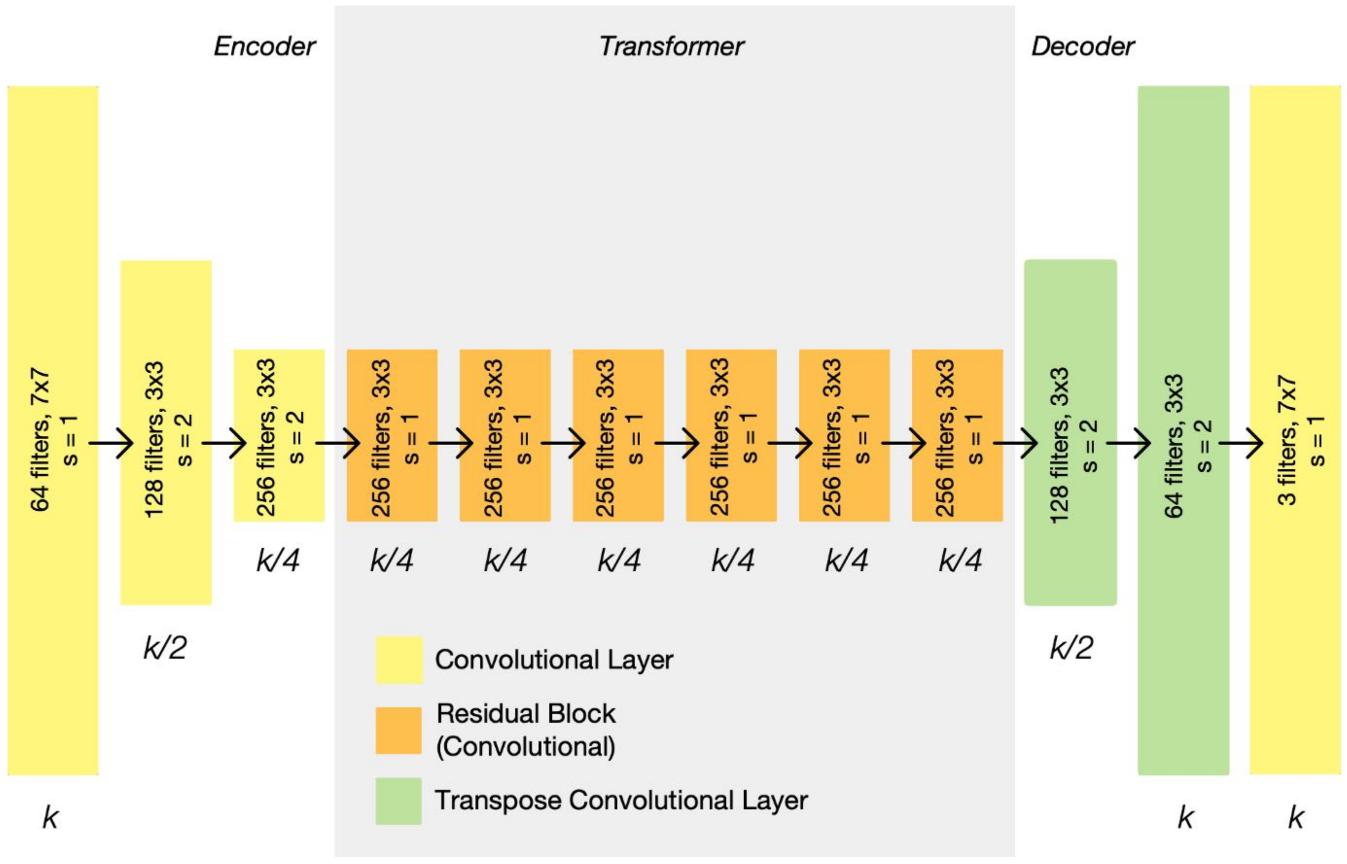
We can create the full objective function by putting these loss terms together, and weighting the cycle consistency loss by a hyperparameter λ. We suggest setting λ = 10.

$$Loss_{full} = Loss_{adv} + \lambda Loss_{cyc}$$

Generator Architecture

Each CycleGAN generator has three sections: an *encoder*, a *transformer*, and a *decoder*. The input image is fed directly into the encoder, which shrinks the representation size while increasing the number of channels. The encoder is composed of three convolution layers. The resulting activation is then passed to the transformer, a series of six residual blocks. It is then expanded again by the decoder, which uses two transpose convolutions to enlarge the representation size, and one output layer to produce the final image in RGB.

You can see the details in the figure below. Please note that each layer is followed by an instance normalization and a ReLU layer, but these have been omitted for simplicity.



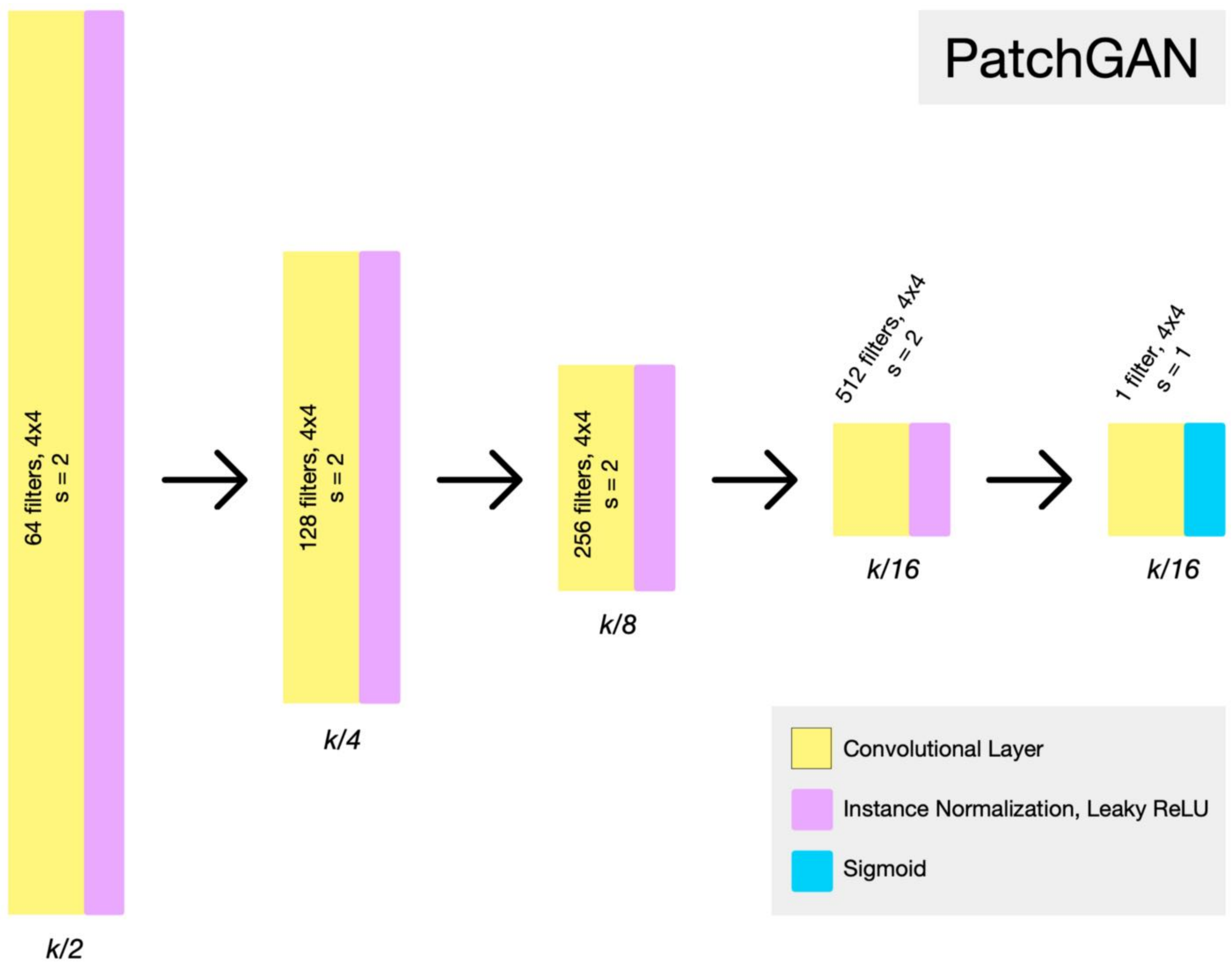
An architecture for a CycleGAN generator. As you can see above, the representation size shrinks in the encoder phase, stays constant in the transformer phase, and expands again in the decoder phase. The representation size that each layer outputs is listed below it, in terms of the input image size, k. On each layer is listed the number of filters, the size of those filters, and the stride. Each layer is followed by an instance normalization and ReLU activation.

Since the generators' architecture is fully convolutional, they can handle arbitrarily large input once trained.

Discriminator Architecture

The discriminators are PatchGANs, fully convolutional neural networks that look at a "patch" of the input image, and output the probability of the patch being "real". This is both more computationally efficient than trying to look at the entire input image, and is also more effective — it allows the discriminator to focus on more surface-level features, like texture, which is often the sort of thing being changed in an image translation task.

If you've read about other image-to-image translation systems, you may already be familiar with PatchGAN. By the time of the CycleGAN paper, a version of PatchGAN had already been successfully used in paired image-to-image translation by Isola et al in Image-to-Image Translation with Conditional Adversarial Nets.



An example architecture for a PatchGAN discriminator. PatchGAN is a fully convolutional network, that takes in an image, and produces a matrix of probabilities, each referring to the probability of the corresponding "patch" of the image being "real" (as opposed to generated). The representation size that each layer outputs is listed below it, in terms of the input image size, k. On each layer is listed the number of filters, the size of those filters, and the stride.

As you can see in the example architecture above, the PatchGAN halves the representation size and doubles the number of channels until the desired output size is reached. In this case, it was most effective to have the PatchGAN evaluate 70x70 sized patches of the input.

Reducing Model Oscillation

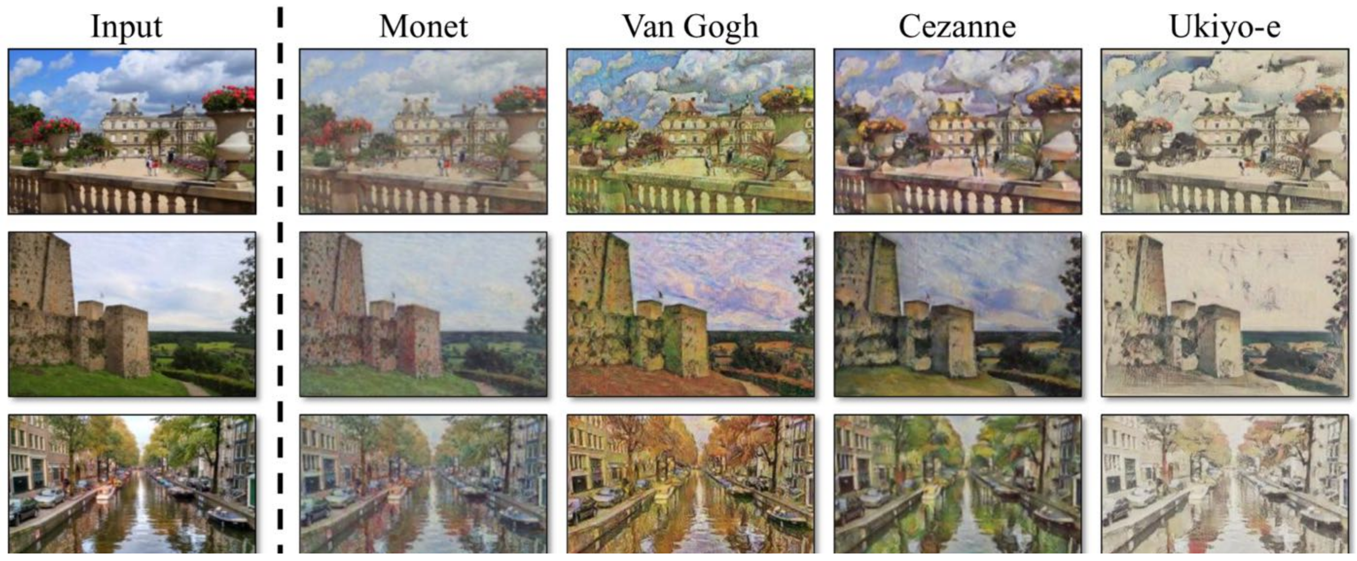
To prevent the model from changing drastically from iteration to iteration, the discriminators were fed a *history* of generated images, rather than just the ones produced by the latest versions of the generators. To do this, we keep a pool to store the 50 most recently generated images. This technique of reducing model oscillation was pioneered by Shrivastava et al. in Learning from Simulated and Unsupervised Images through Adversarial Training.

Other Training Details

The training approach was fairly typical for an image-to-image translation task. The Adam optimizer, a common variant of gradient descent, was used to make training more stable and efficient. The learning rate was set to 0.0002 for the first half of training, and then linearly reduced to zero over the remaining iterations. The batch size was set to 1, which is why we refer to *instance normalization*, rather than *batch normalization*, in the architecture diagrams above.

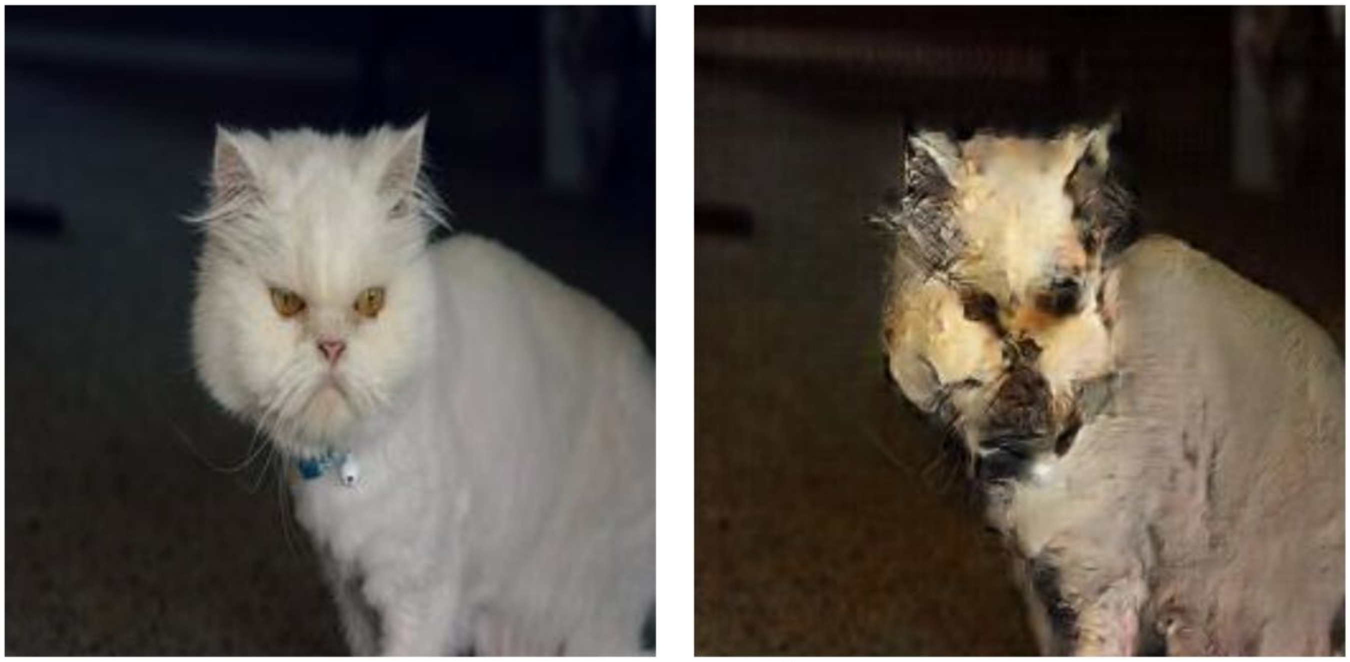
Strengths and Limitations

Overall, the results produced by CycleGAN are very good — image quality approaches that of paired image-to-image translation on many tasks. This is impressive, because paired translation tasks are a form of fully supervised learning, and this is not. When the CycleGAN paper came out, it handily surpassed other unsupervised image translation techniques available at the time. In "real vs fake" experiments, humans were unable to distinguish the synthesized image from the real one about 25% of the time.



CycleGAN can be used for collection style transfer, where the entire works of an artist are used to train the model. *Image taken from paper.

If you are planning to use CycleGAN for a practical application, it is important to be aware of its strengths and limitations. It works well on tasks that involve color or texture changes, like day-to-night photo translations, or photo-to-painting tasks like *collection style transfer* (see above). However, tasks that require substantial geometric changes to the image, such as cat-to-dog translations, usually fail.



A very unimpressive attempt at a cat-to-dog image translation. Don't try to use a CycleGAN for this. *Image taken from paper.

Translations on the training data often look substantially better than those done on test data.

Conclusion

Thanks for reading! I hope this was a useful overview. If you would like to see more implementation details, there are some great public implementations out there you can refer to. Please leave a comment if you have questions, corrections, or suggestions for improving this post.

Machine Learning Gans Computer Vision Towards Data Science