Exploring CycleGAN and Its Application to Font Transfer

Asif Iqbal
Ryerson University
Toronto, ON, Canada
asif1.iqbal@ryerson.ca

Abstract

Unpaired image to image translation has gained quite a bit of attention with the advent of Cycle-Consistent Generative Adversarial Networks (CycleGANs). Translation domains like $horse \leftrightarrow zebra$, $apple \leftrightarrow orange$, $photo \leftrightarrow painting$, $summer \leftrightarrow winter$ and several others have been explored in the original work. In this paper, we plan to regenerate some of the works done in CycleGAN, try out a couple of the already tried domains on our own, and finally apply the CycleGAN concept to font style transfer. Specifically, we try it out on Arial to Times New Roman black fonts for single uppercase characters and also on lowercase multi-character words, and demonstrate that it might be a promising direction. Although it is not at all hard to get paired data for text fonts, we hope that our approach can in the future be extended to font image transfer tasks where paired data might indeed be hard to attain. The code has been made open source in the Github repository https://github.com/asif31iqbal/cycle-gan-pytorch.

1. Introduction

Image-to-image translation is a class of vision and graphics problems where the goal is to learn the mapping between an input image and an output image using a training set of aligned image pairs [28]. Image to image [28]. The field of image-to-image translation has been studied to quite an extent over the last couple of years. This problem can be more broadly described as converting an image from one representation of a given scene, x, to another, y, e.g., grayscale to color, image to semantic labels, edge-map to photograph [10, 28]. Years of research in computer vision, image processing, computational photography, and graphics have produced powerful translation systems in the supervised setting, where example image pairs $\{x_i, y_i\}_{i=1}^{N}$ are available [3, 4, 9, 11, 13, 16, 19, 23, 24, 27]. However, obtaining paired data for many tasks can be difficult and expensive. Obtaining input-output pairs for graphics tasks like artistic stylization can be even more difficult since the desired output is highly complex, typically requiring artistic authoring [28]. Let's say we want to transfer a particular summer scene into a winter one and vice versa. We can easily imagine how the corresponding winter version of a summer scene or a summer version of a winter scene might look like even though we might have never seen a summer and winter version of the same scene side by side. Based on this insight, the authors of CycleGAN [28] came up with the algorithm that can learn to translate between domains without paired input-output examples, assuming that there is some underlying relationship between the domains for example, that they are two different renderings of the same underlying scene and seek to learn that relationship. Although the algorithm lacks supervision in the form of paired examples, it can exploit supervision at the level of sets: we are given one set of images in domain X and a different set in domain Y. We may train a mapping $G : X \leftarrow Y$ such that the output $\hat{y} = G(x), x \in X$, is indistinguishable from images $y \in Y$ by an adversary trained to classify $\hat{y}$ apart from y [28]. However, as discussed in [28] that there could be infinitely many mappings G that will induce the same distribution over $\hat{y}$. Also, there is the problem of mode collapse [7], where all input images map to the same output image and the optimization fails to make progress.

To tackle these problems, the CycleGAN [28] authors leverage the notion of *cycle consistency*, in the sense that if we transfer the font style of a character from Arial to Times New Roman, and then translate it back from Times New Roman to Arial, we should get back the original character. Mathematically, if we have a translator $G : X \leftarrow Y$ and another translator $F : Y \leftarrow X$, then G and F should be inverses of each other, and both mappings should be bijections. We apply this structural assumption by training both the mapping G and F simultaneously, and adding a cycle consistency loss [**?**] that encourages $F(G(x)) \approx x$ and $G(F(y)) \approx y$.

The authors [28] have applied this idea to a wide range of applications, like collection style transfer, object transfiguration, season transfer and photo enhancement. In this paper, we first attempt to regenerate their work and

network architecture from scratch, apply it to couple of domains that they have already tried out, namely season ($summer \leftrightarrow winter$) transfer and object style transfer ($apple \leftrightarrow orange$). Next, we apply it to the domain of font style transfer. We limit ourselves to just two fonts - Arial and Times New Roman. We try it on single uppercase black English characters and on lowercase black English words. Although it is not difficult to get paired data for this sort of font style transfers for well known fonts which are widely available, there are unknown fonts, text and calligraphy styles that are available in the wild for which it is not easy to get paired data and applying the CycleGAN concept might be a good idea. The attempt with known fonts in this paper is a baby step towards the possible applicability of unknown font style transfers using cycle consistency.

2. Related Work

Over the last couple of years, Generative Adversarial Networks (GANs) [7, 8] have achieved quite a bit of success in image generation. The key idea behind GAN's success is the *adversarial loss* that forces the generated image to be indistinguishable from the input image. In the CycleGAN case, the adversarial loss has been adopted in such a way that the generated images are indistinguishable from the images in the target domain.

Much of the related work regarding unpaired image-to-image translation have been mentioned in the original paper [28]. Approaches like [9, 16, 18, 12] and **Pix2Pix** work on paired training examples, as opposed to the unpaired training concept that CycleGAN relies on.

There has been a few works on unpaired image-to-image translation as well. Works like [1, 14, 15] uses a weight sharing strategy between domains. Another group of work like [2, 20, 25] encourages the input and output to share specific content features even though they may differ in style. Unlike these approaches, the CycleGAN concept does not rely on any task specific, predefined similarity measurements. It's more of a general purpose framework.

As mentioned in the original paper, the idea of cycle consistency also has quite a bit of a history. Of these works, [6], [26] and [**?**] are the ones that are conceptually most similar to CycleGANs.

Neural Style Transfer [11, 5, 21] is another family of work for image to image translation, which synthesizes an image by combining the content of one image with the style of another image. Again, this is a paired training concept while CycleGAN is unpaired.

The primary focus of CycleGAN and hence also of this paper, is learning the mapping between two two image collection, rather than between two specific images, by trying to capture correspondence between higher-level appearance structures. We try to apply the same idea in case of font style transfer.

3. Problem Formulation

We formulate our problem in the same way the original authors [28] did, where the goal is to learn mapping between two domains A and B given training samples $\{a_i\}_{i=1}^{N}$ where $a_i \in A$ and $\{b_j\}_{j=1}^{M}$ where $b_j \in B$. The problem formulation has been depicted broadly in Figure 1.

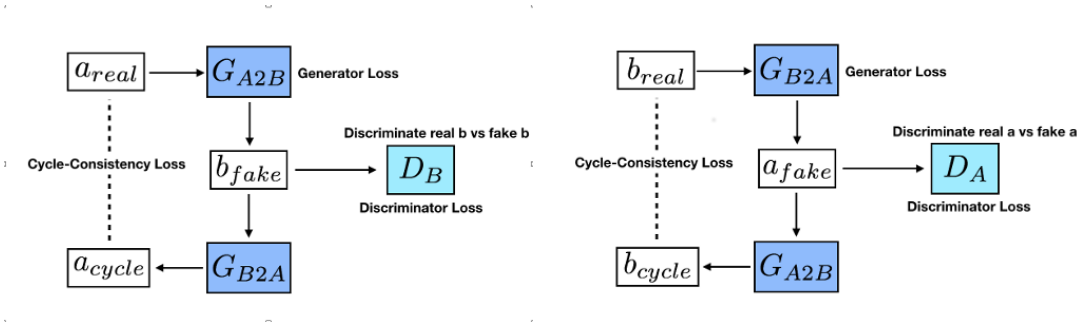
We have two generators G_A and G_B, and two discriminators D_A and D_B. G_{A2B} takes a real image a_{real} from domain A and generates a fake image b_{fake} in domain B, while G_{B2A} takes a real image b_{real} from domain B and generates a fake image a_{fake} in domain A. Discriminator D_A tries to discriminate between the generated image a_{fake} and real images in domain A, while discriminator D_B tries to discriminate between the generated image b_fake and real images in domain B. The generated fake image b_{fake} is then fed back to G_{B2A} to generate an image a_{cycle} in domain A, while the generated fake image a_{fake} is then fed back to G_{A2B} to generate an image b_{cycle} in domain B.

3.1. Loss Functions

As can be seen from Figure 1, there are broadly 2 sorts of losses - *adversarial loss* (generator loss and discriminator loss) and *cycle-consistency loss*. The adversarial loss comprises of the losses incurred from the generators trying to fool the discriminators to take fake images as real in their corresponding domain, and from the discriminators trying to distinguish fake images from real ones. As discussed in [28], with large enough capacity, a network can map the same set of input images to any random permutation of images in the target domain, where any of the learned mappings can induce an output distribution that matches the target distribution. Thus, adversarial losses alone cannot guarantee that the learned function can map an individual input a_{real} to a desired output y_{fake}. This is where the *cycle-consistency* loss kicks in - the image a_{cycle} should be the same as image a_{real}, and the image b_{cycle} should be the same as image b_{real}.

To be complete, the loss functions can be broken down into the following components:

1. D_A must approve all the original images a_{real} of the domain A
2. D_A must reject all the images b_{fake} which are generated by G_{B2A} to fool it
3. G_{B2A} must make D_A approve all the generated images b_{fake}, so as to fool it
4. Image b_{cycle} must retain the property of original image b_{real}
5. D_B must approve all the original images b_{real} of the domain B



(a) Generator G_{A2B} takes an image a_{real} from domain A and outputs a_{fake} which Discriminator D_B tries to distinguish from actual images in domain B. Image a_{fake} is then passed onto generator G_{B2A} which generates a_{cycle}. This is used for calculating the *cycle-consistency* loss.

(b) Generator G_{B2A} takes an image b_{real} from domain B and outputs a_{fake} which Discriminator D_A tries to distinguish from actual images in domain A. Image b_{fake} is then passed onto generator G_{A2B} which generates b_{cycle}. This is used for calculating the *cycle-consistency* loss.

Figure 1: Problem Formulation

6. D_B must reject all the images a_{fake} which are generated by G_{A2B} to fool it

7. G_{A2B} must make D_B approve all the generated images a_{fake}, so as to fool it

8. Image a_{cycle} must retain the property of original image a_{real}

In the above list, items $1, 2, 3, 5, 6$ and 7 are adversarial components of the loss, while items 4 and 8 a re the *cycle-consistency* components.

The original authors used L_2 (MSE) loss for the adversarial components, and L_1 loss for the *cycle-consistency* component since it earned them better results. We adhere to the same principle for our work. The loss equations can be mathematically written as follows:

$$\mathcal{L}_{disc} = \|D_A(a_{real}) - 1\|_2 + \|D_B(b_{real}) - 1\|_2 + \|D_A(a_{fake}) - 0\|_2 + \|D_B(b_{fake}) - 0\|_2$$

This captures 1, 2, 5 and 6 above.

$$\mathcal{L}_{gen} = \|D_A(a_{fake}) - 0\|_2 + \|D_B(b_{fake}) - 0\|_2$$

This captures 3 and 7 above.

$$\mathcal{L}_{cycle} = \|a_{real} - a_{cycle}\|_1 + \|b_{real} - b_{cycle}\|_1$$

This captures 4 and 8 above.

So the total loss comes down to:
$\mathcal{L}_{total} = \mathcal{L}_{disc} + \mathcal{L}_{gen} + \lambda\mathcal{L}_{cycle}$

where λ is a parameter to control how much weight we want to put on the *cycle-consistency* as opposed to the adversarial behaviour.

In addition, for certain domains, the original authors [28] also used an *Identity* loss, to ensure that passing an image in domain A through generator G_{B2A} produces the same image and passing an image in domain B through generator G_{A2B} produces the same image.

$$\mathcal{L}_{identity} = \|G_{B2A}(a_{real}) - a_{real}\|_1 + \|G_{A2B}(b_{real}) - b_{real}\|_1$$

We can control this loss using another parameter β. In that case, the total loss comes down to:

$\mathcal{L}_{total} = \mathcal{L}_{disc} + \mathcal{L}_{gen} + \lambda\mathcal{L}_{cycle} + \beta\mathcal{L}_{identity}$

The original paper uses a λ value of 10 and a β value of 5 where used. We keep the β value as 5 (where used) but vary the λ value as 5 and 10 for performing ablation discussed later.

4. Data Collection

For the $summer \leftrightarrow winter$ and $apple \leftrightarrow orange$ transformations, we collect the datasets from the original authors' source data [17]. The *summer2winter* dataset contains 1231 train and 309 test images for summer and 962 train and 238 test images for winter. The *apple2orange* dataset contains 995 train and 266 test images for apple and 1019 train and 248 test images for orange.

For the font style transfer, we have written python programs (included in the github repository) to generate images of single uppercase English characters and lowercase words. We obtain the words vocabulary from [22]. For

the single uppercase character scenario, we generated 11 train images and 10 test images for each character with random scaling and translation, so in total we got 286 train images and 260 test images. For the word scenario, we randomly sampled 1981 words from [22] to create four disjoint datasets (sizes 500, 493, 496, 492) in a way that each of them has an approximately equal number of words starting with a certain character, to avoid bias as much as possible. We used the first 2 of these sets to generate images with Arial font and use those for training and testing respectively. Similarly, we used the latter 2 sets to generate training and testing images for Times New Roman font. We also applied random scaling and translation to the words before generating the words.

5. Network Architecture

We essentially rebuilt the same network architecture as the original authors [28] have used. The architecture is shown in Figure 2. The numbers of channels at all intermediate stage have been shown in red, as well as the kernel, stride and padding size in blue. If the type of padding is Reflection padding, it has been used indicated in the figure in the corresponding blocks, otherwise padding is done with 0 values everywhere else.

5.1. Generator

The generator has 3 main segments - Encoding, Transformation an Decoding. Figure 2a The encoding part has 3 general convolutional layers each of which is a convolution followed by instance normalization [28] and Relu.b The transformation part is a series of 9 residual blocks (shown in more detail in Figure 2b). Finally, the decoding segment is a couple of general deconvolutional layer followed by a convolution+tanh(kernel size 7). Each general deconvolutional layer is a deconvolution followed by instance normalization and Relu. The details have benn shown in Figure 2a.

5.2. Discriminator

For the discriminator, we use the same 70×70 PatchGAN concept used in [28]. It basically has 4 general convolutional layers. The first one is a convolution plus Leaky Relu, while the other three are convolution plus instance normalization plus Leaky Relu. These 4 layers are then followed by one single convolution only layer. The details have been shown in red in Figure 2c.

Notice that both the generator and the discriminator networks are end-to-end fully convolutional. Before feeding our images to these networks, we resize them to 256×256. The output size of the generator network would be of the same size as it's input ($256 \times 256 \times 3$). The output of the discriminator is just a one single channel 30×30 feature map, which is compared against a 30×30 all 1's or all 0's tensor while calculating discriminator loss.

6. Experimentation

6.1. Training

We train our networks on a GeForce 1080 GPU using the training data collected for $summer2winter$ and $apple2orange$, and also using the generated training data for our font style transfer. The original authors [28] used 200 epochs and a learning rate of 0.0002, exponentially decaying the rate after first 100 epochs. We found after quite a bit of experimentation that for the $apple \leftrightarrow orange$ and $summer \leftrightarrow winter$ cases, the networks effectively cease to learn after 50 - 60 epochs, so we train these for 50 epochs only, exponentially decaying the learning rate after 25 epochs. We do the same for the font style transfer for full words. However, for the single character font style transfer, given that we have less training data, we use 100 epochs, exponentially decaying the learning rate after 50 epochs. We use Adam optimizer and a batch size of 1. Table 1 shows a summary of the training times.

Domain	Epochs	Training Time
$Apple \leftrightarrow Orange$	50	5 hours
$Summer \leftrightarrow Winter$	50	5 hours
$Arial \leftrightarrow Times$	100	3 hours
$Arial\,word \leftrightarrow Times\,word$	50	3.5 hours

Table 1: Training Times for different domains

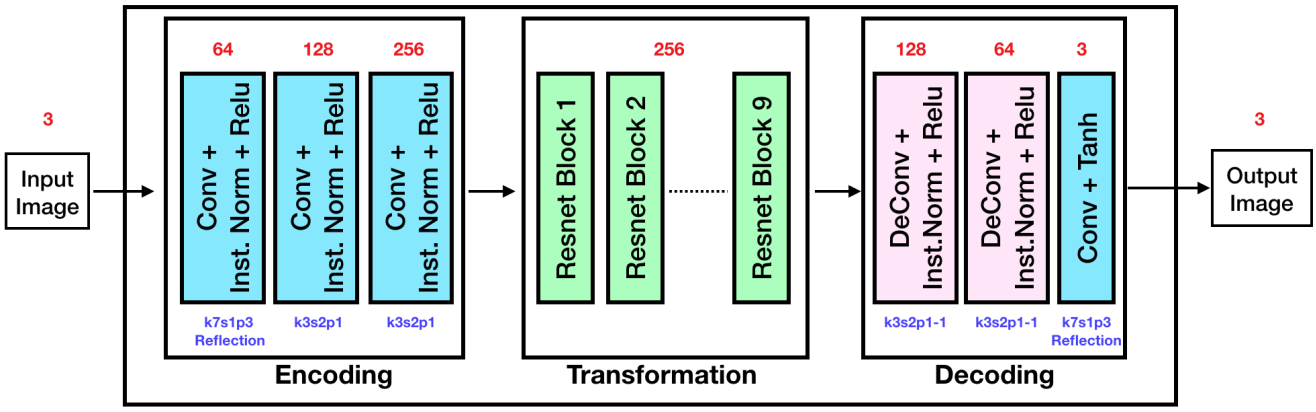
6.2. Evaluation

This is a kind of problem where the visual perception is a gold standard for evaluating the quality of the translated images. Although a rigorous way of evaluating the quality of our images could have been using Amazon Mechanical Turk (AMT), due to our time constraints, we leave the quality evaluation to the visual perception of ourselves and the readers. Figures 3a and 4 show some sample results of the $apple \leftrightarrow orange$ and $summer \leftrightarrow winter$ transfers. While the results are not extremely impressive, they do give the impression of real transfers to some extent.

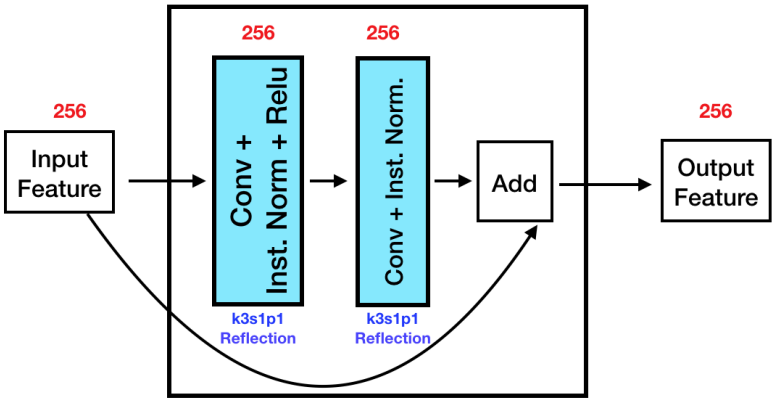
We show some of our single character font style transfer results in Figure 5 and some of our full word font style transfer results in Figure 6a. We see that although according to the original paper [28], the generators of CycleGAN was engineered for good performance on the appearance changes and not the shape changes in particular, it is capturing the shape changes of the fonts reasonably nicely.

References

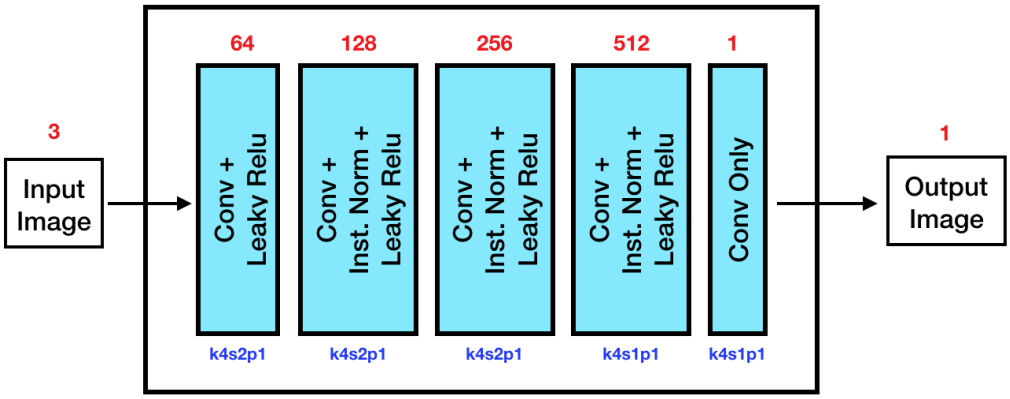
[1] Y. Aytar, L. Castrejon, C. Vondrick, H. Pirsiavash, and A. Torralba. Cross-modal scene networks. *PAMI*, 2016.



(a) Generator



(b) Resnet Block in Detail



(c) Discriminator

Figure 2: Network Architecture. The numbers in red refers to the number of output channels of each block. The letter-number combination in blue in form kxsypz beneath the blocks corresponds to the kernel size (x), stride (y) and padding (z). For example, k4s2p1 would mean a kernel size of 4×4, stride of 1 and padding of 1. If the padding is form $z - z$, that means that a padding of z is applied to the output as well (in addition to the input padding of z). The use of reflection padding has been indicated by the word Reflection in blue

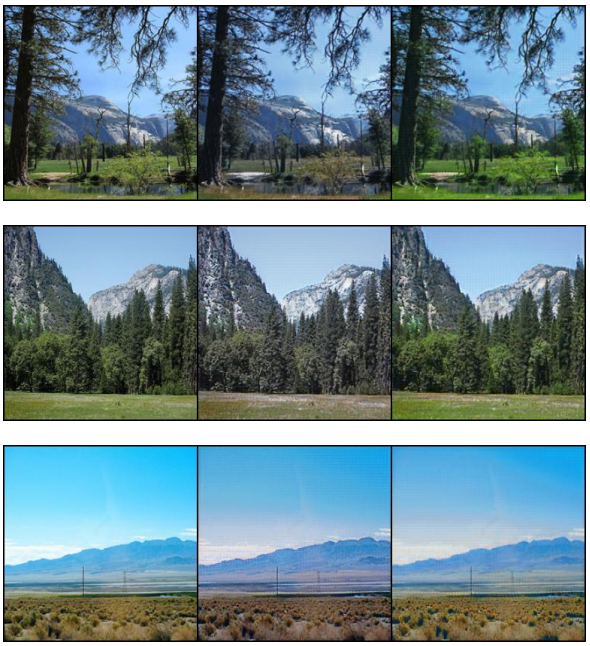
[2] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. *CVPR*, 2017.

[3] D. Eigen and R. Fergus. Predicting depth, surface normal and

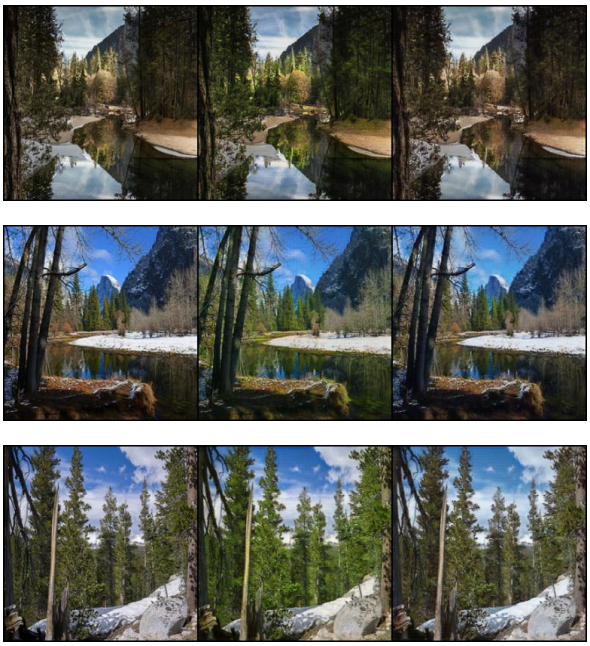
(a) $Apple \rightarrow Orange \rightarrow Apple$

(b) $Orange \rightarrow Apple \rightarrow Orange$

Figure 3: $Apple \leftrightarrow Orange$



(a) $Summer \rightarrow Winter \rightarrow Summer$



(b) $Winter \rightarrow Summer \rightarrow Winter$

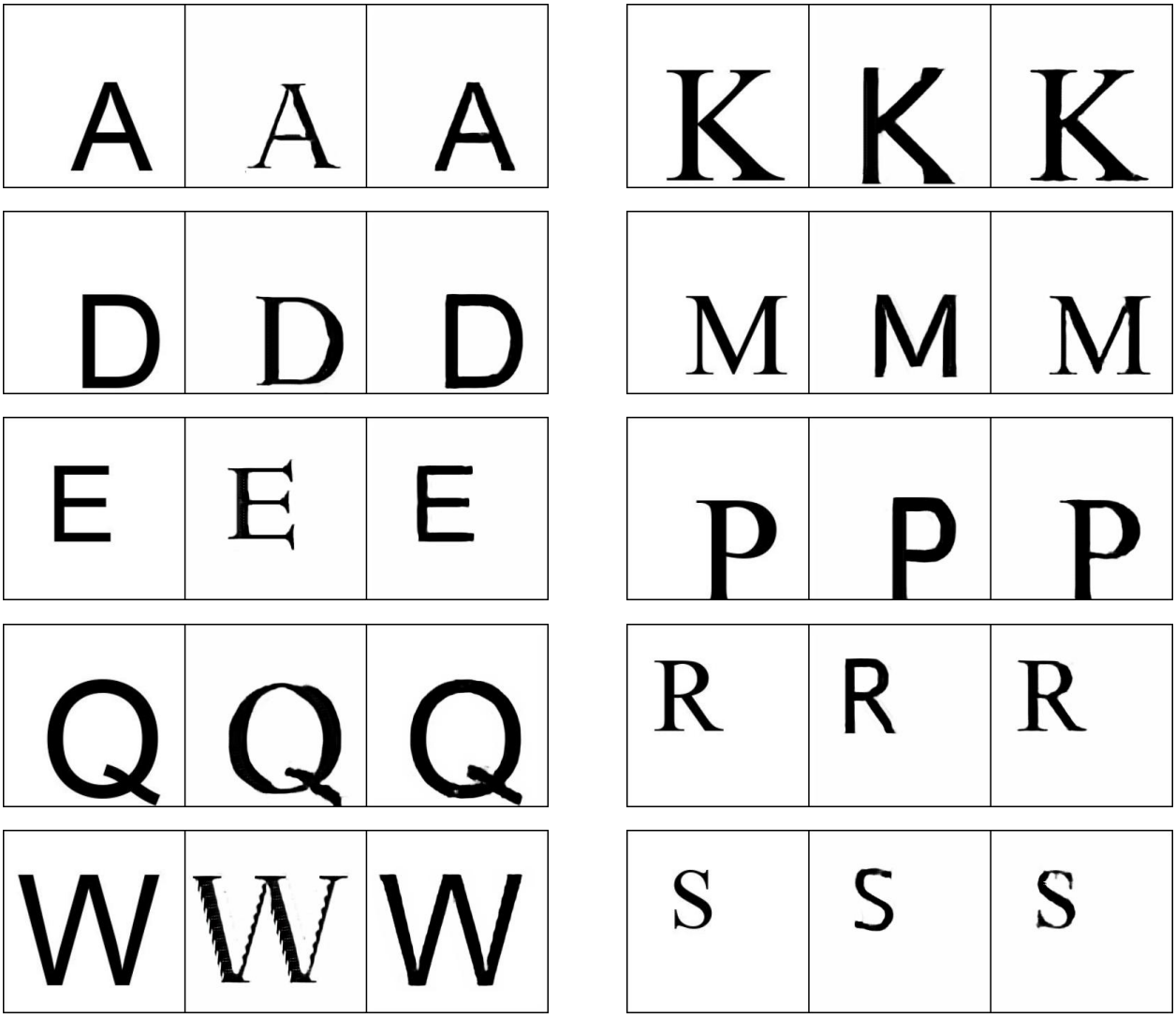
Figure 4: $Summer \leftrightarrow Winter$

semantic labels with a common multi-scale. *ICCV*, 2015.

[4] D. Eigen and R. Fergus. Predicting depth, surface normal and semantic labels with a common multi-scale. *ICCV*, 2015.

[5] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. *CVPR*, 2016.

[6] C. Godard, O. M. Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. *CVPR*, 2017.



(a) $Arial \rightarrow Times \rightarrow Arial$ (b) $Times \rightarrow Arial \rightarrow Times$

Figure 5: $Arial \leftrightarrow Times$

[7] I. Goodfellow. Nips 2016 tutorial: Generative adversarial. *arXiv preprint arXiv:1701.00160*, 2016.

[8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *NIPS*, 2014.

[9] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. *SIGGRAPH*, 2001.

[10] P. Isola, J. Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017.

[11] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. *ECCV*, 2016.

[12] L. Karacan, Z. Akata, A. Erdem, and E. Erdem. Learning to generate images of outdoor scenes from attributes and semantic layouts. *arXiv preprint arXiv:1612.00215*, 2016.

[13] P.-Y. Laffont, Z. Ren, X. Tao, C. Qian, and J. Hays. Transient attributes for high-level understanding and editing of outdoor scenes. *ACM TOG, 33(4):149*, 2014.

[14] M.-Y. Liu, T. Breuel, and J. Kautz. Unsupervised image-to-image translation networks. *NIPS*, 2017.

[15] M.-Y. Liu and O. Tuzel. Coupled generative adversarial. *NIPS*, 2016.

[16] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CVPR*, 2015.

[17] T. Park. Cyclegan dataset. `https://people.eecs.berkeley.edu/~taesung_park/CycleGAN/datasets`.

[18] P. Sangkloy, J. Lu, C. Fang, F. Yu, and J. Hays. Scribbler:controlling deep image synthesis with sketch and color. *CVPR*, 2017.

(a) *Arial* word → *Times* word→ *Arial* word

(b) *Times* word → *Arial* word→ *Times* word

Figure 6: *Arial* word ↔ *Times* word

[19] Y. Shih, S. Paris, F. Durand, and W. T. Freeman. Datadriven hallucination of different times of day from a single outdoor photo. *ACM TOG, 32(6):200*, 2013.

[20] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from simulated and unsupervised images through adversarial training. *CVPR*, 2017.

[21] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. *ICML*, 2016.

[22] I. University of California. Uci bag-of-words vocabulary. `https://archive.ics.uci.edu/ml/machine-learning-databases/bag-of-words/vocab.kos.txt`.

[23] X. Wang and A. Gupta. Generative image modeling using style and structure adversarial networks. *ECCV*, 2016.

[24] S. Xie and Z. Tu. Holistically-nested edge detection. *ICCV*, 2015.

[25] A. P. Y. Taigman and L. Wolf. Unsupervised cross-domain image generation. *ICLR*, 2017.

[26] Z. Yi, H. Zhang, T. Gong, Tan, and M. Gong. Unsupervised dual learning for image-to-image translation. *ICCV*, 2017.

[27] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. *ECCV*, 2016.

[28] J. Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CVPR*, 2017.