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 [30]. Image to image [30]. 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 [11, 30]. 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 [4, 5, 10, 12, 14, 17, 20, 24, 25, 28]. 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 [30]. 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 [30] 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 [30]. However, as discussed in [30] 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 [8], where all input images map to the same output image and the optimization fails to make progress.

To tackle these problems, the CycleGAN [30] 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 [29] that encourages $F(G(x)) \approx x$ and $G(F(y)) \approx y$.

The authors [30] 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 (like is posters or artistic drawings) 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) [8, 9] 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 [30]. Approaches like [10, 17, 19, 13] 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, 15, 16] uses a weight sharing strategy between domains. Another group of work like [3, 21, 26] 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, [7], [27] and [29] are the ones that are conceptually most similar to CycleGANs.

Neural Style Transfer [12, 6, 22] 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.

There has been use of GANs specifically in font style transfer as well. The most important one seems to be the use of multi-content GAN [2]. However, their main focus is taking partial observations of highly- stylized text and generalizing the observations to generate unobserved fonts. Our intent, however, in this paper is to apply the general framework of cycle consistency to font style transfer.

In the remainder of this paper, we discuss the problem formulation, data collection, network architecture and experimentation in order.

3. Problem Formulation

We formulate our problem in the same way the original authors [30] 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.

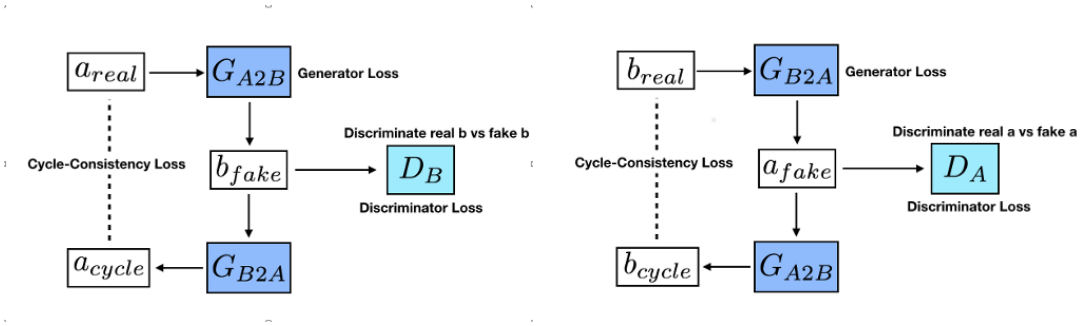
We have two generators G_A and G_B, and two discriminators D_A and D_B. G_{A2B} takes a real image a_{real} from domain A and generates a fake image b_{fake} in domain B, while G_{B2A} takes a real image b_{real} from domain B and generates a fake image a_{fake} in domain A. Discriminator D_A tries to 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 [30], 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



(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

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
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 [30] 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 [18]. 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 [23]. 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 [23] 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 [30] have used. The architecture is shown in Table 1. The generator architecture has been broken down into 3 segments - encoding, transformation and decoding. Note the use of reflection padding to reduce artifacts in the first and last layer of the generator network, as well as in the residual blocks in the middle. Instance normalization has been used just like the original paper.

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 [30] 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. We initialize the weights with a Normal distribution with mean 0 and standard deviation of 0.02. Table 2 shows a summary of the training times.

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 2a and 3 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 4 and some of our full word font style transfer results in Figure 5a. We see that although according to the original paper [30], 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.

6.3. Ablation

We perform ablation studies separately on different domains. The main parameters we use for our ablation studies are number of epochs, learning rate, weight of the *cycle-consistency loss* λ and use of the *identity loss*.

6.3.1 Apple $\leftrightarrow$ Orange

Number of epochs We tried varying number of epochs as 200, 100 and 50. We found that number of epochs more than 50 does not really help, and the networks seem to be not learning much after that. We decided to keep it to 50.

Learning Rate We found that using a rate of 0.0002 sometimes possibly gets trapped in a local minima and the network starts learning to invert the images for the first transformation, as if it is looking for a shortcut to get back to the original image via the cyclic transformation. Figure 6a demonstrates this.

Segment	Layer	Description	Channels
Encoding	Conv-Instance Norm-Relu	Kernel 7×7, Stride 1, **Reflection** Pad 3	In:3 Out:64
	Conv-Instance Norm-Relu	Kernel 3×3, Stride 2, Pad 1	In:64 Out:128
	Conv-Instance Norm-Relu	Kernel 3×3, Stride 2, Pad 1	In:128 Out:256
Transformation	9 Residual Blocks	See details in Table 1b	In:256 Out:256
Decoding	DeConv-Instance Norm-Relu	Kernel 3×3, Stride 2, Pad 1, Output Pad 1	In:256 Out:128
	DeConv-Instance Norm-Relu	Kernel 3×3, Stride 2, Pad 1, Output Pad 1	In:128 Out:64
	Conv-Tanh	Kernel 7×7, Stride 1, **Reflection** Pad 3	In:64 Out:3

(a) Generator Architecture

Layer	Description	Channels
Conv-Instance Norm-Relu	Kernel 3×3, Stride 1, **Reflection** Pad 1	In:256 Out:256
Conv-Instance Norm	Kernel 3×3, Stride 1, **Reflection** Pad 1	In:256 Out:256
Add with input		In:256 Out:256

(b) Details of each residual block

Layer	Description	Channels
Conv-LeakyRelu(0.2)	Kernel 4×4, Stride 2, Pad 1	In:256 Out:64
Conv-Instance Norm-LeakyRelu(0.2)	Kernel 4×4, Stride 2, Pad 1	In:64 Out:128
Conv-Instance Norm-LeakyRelu(0.2)	Kernel 4×4, Stride 2, Pad 1	In:128 Out: 256
Conv-Instance Norm-LeakyRelu(0.2)	Kernel 4×4, Stride 1, Pad 1	In:256 Out:512
Conv	Kernel 3×3, Stride 1, Pad 1	In:512 Out:1

(c) Discriminator Architecture

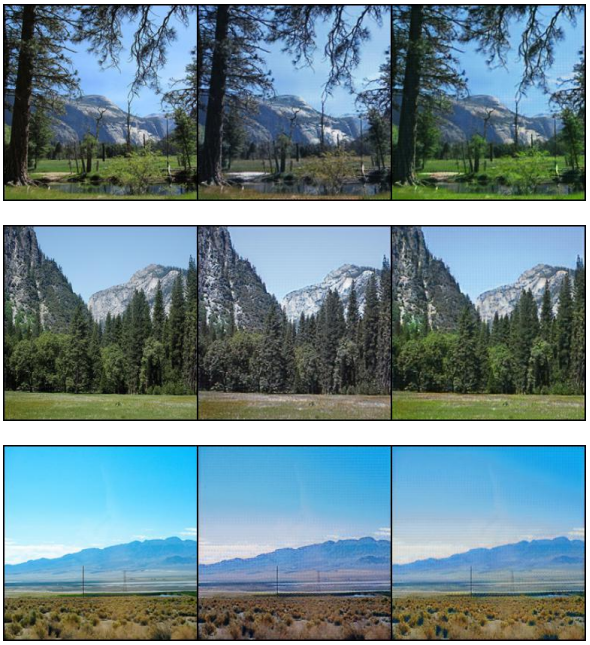
Table 1: Network Architecture Summary

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

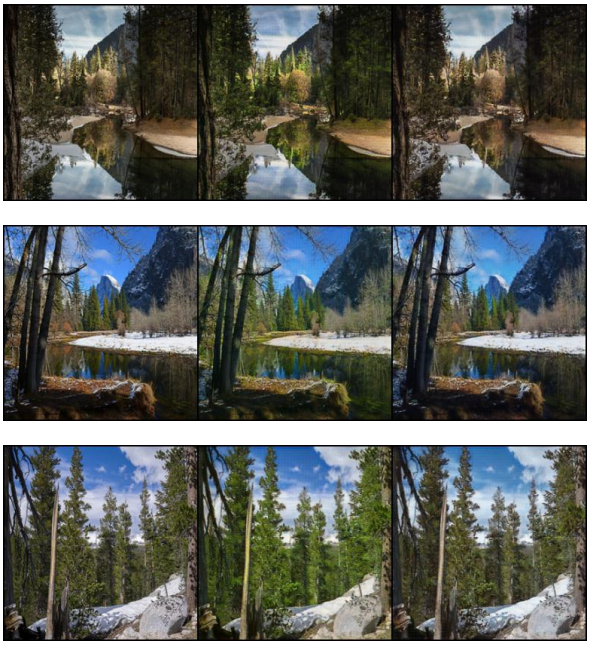
(b) $Orange \rightarrow Apple \rightarrow Orange$

Figure 2: $Apple \leftrightarrow Orange$

Cycle-consistency weight λ and Identity weight β We tried with λ values of 10 and 5, and for this case 5 turned out to work better. With 10, we get a similar local minima-like inverted phenomena that we discussed above for the low learning rate. For β, we tried with values 0 (exclude) and 1 (include), and 0 gave us better results.

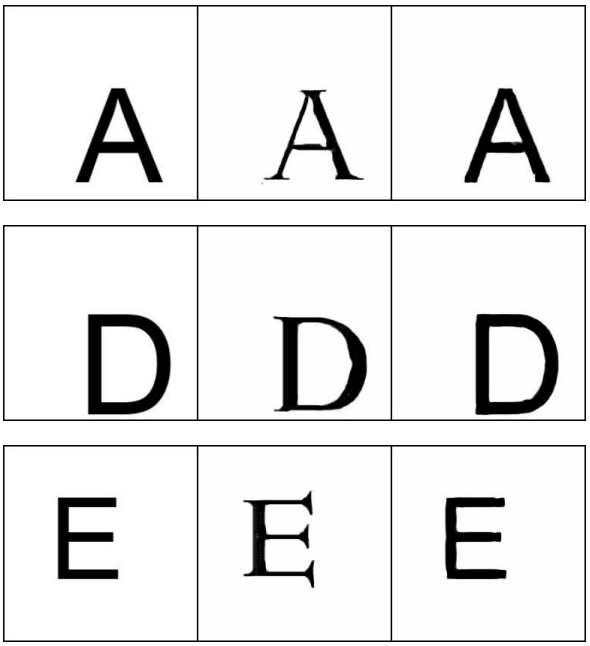


(a) $Summer \rightarrow Winter \rightarrow Summer$

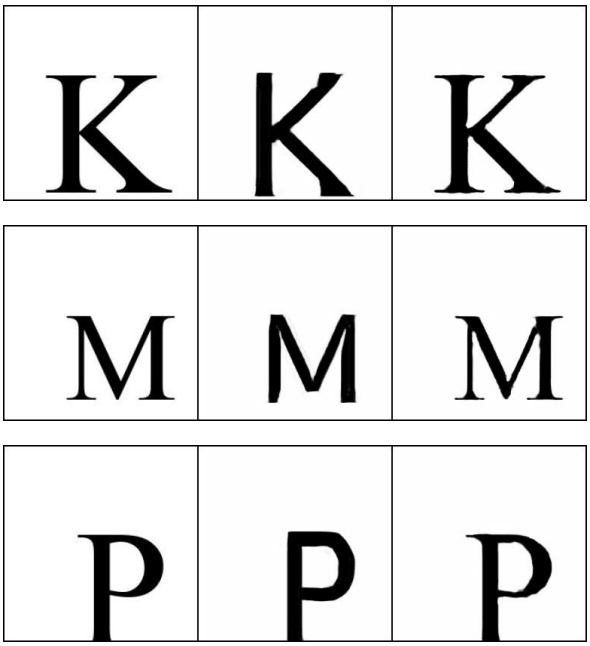


(b) $Winter \rightarrow Summer \rightarrow Winter$

Figure 3: $Summer \leftrightarrow Winter$



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



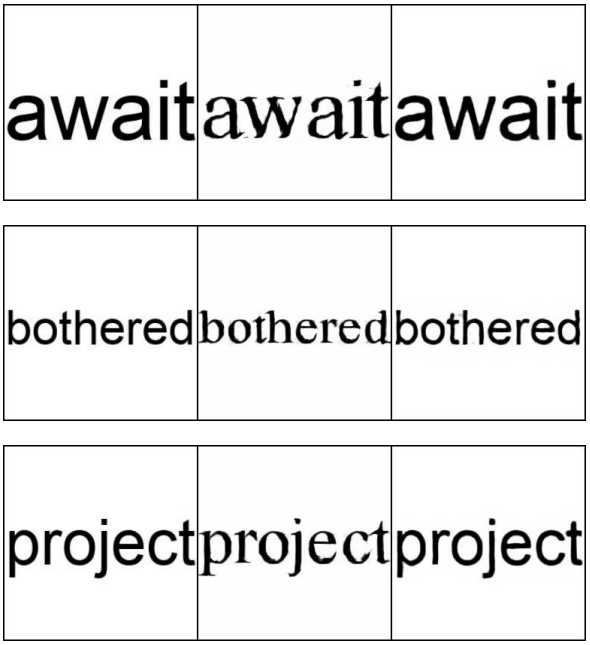
(b) $Times \rightarrow Arial \rightarrow Times$

Figure 4: $Arial \leftrightarrow Times$

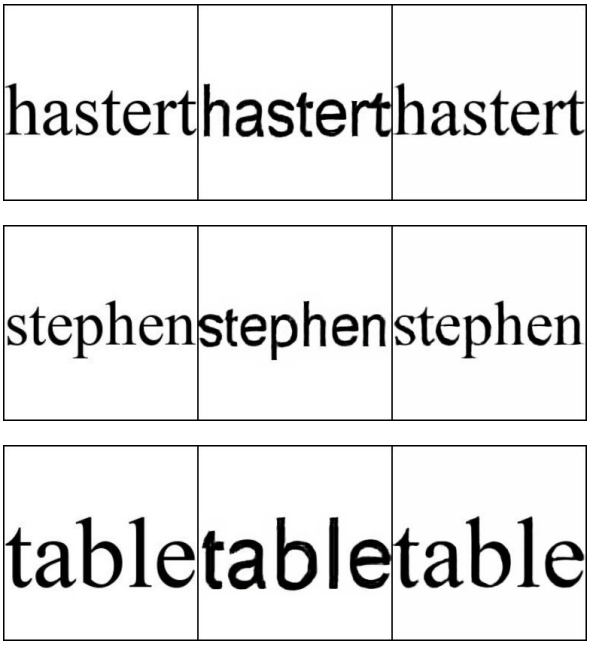
6.3.2 Summer ↔ Winter

Number of epochs Similarly for the $Apple \leftrightarrow Orange$ case, 50 turned out to be sufficient.
Learning Rate For this case, a learning rate of 0.0002 gave us slightly better results than 0.0004.

Cycle-consistency weight λ and Identity weight β Again, we tried with λ values of 10 and 5, and for this case 10 turned out to work better. For β, we tried with values 0



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



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

Figure 5: *Arial* word ↔ *Times* word

(a) Learning rate 0.0002 (above) and 0.0004 (below)

(b) Identity loss not included (above) and included (below)

Figure 6: Learning Rate and Identity loss ablation for *Apple* ↔ *Orange* and *Summer* ↔ *Winter*

Domain	Epochs	Training Time
Apple ↔ *Orange*	50	5 hours
Summer ↔ *Winter*	50	5 hours
Arial ↔ *Times*	100	3 hours
Arial word ↔ *Times word*	50	3.5 hours

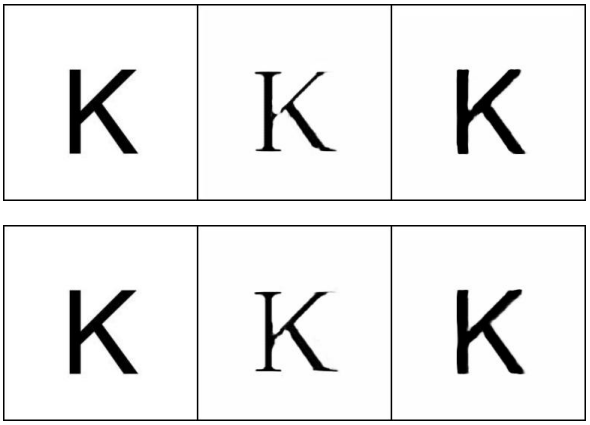
Table 2: Training Times for different domains

(exclude) and 1 (include), and 1 gave us better results. Just as the original authors [30] had noticed, without the identity loss it changes the tint of the image a little bit. Figure 6b demonstrates this.

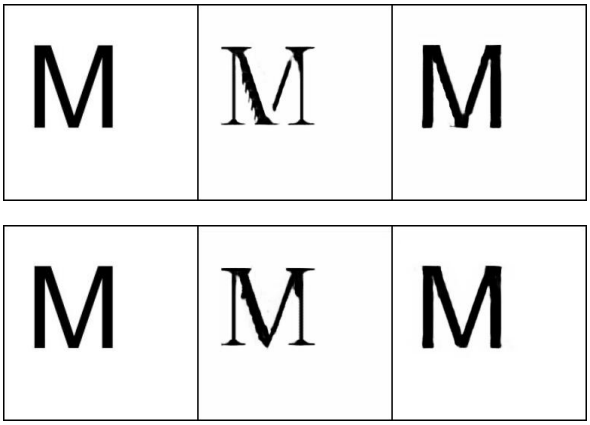
6.3.3 Font Style Transfer

Number of epochs For single character uppercase *Arial* ↔ *Times*, we settle for an epoch number of 100, while for the full word transfer, we settle for 50.

Learning Rate Again for this case, a learning rate of 0.0002 gave us slightly better results than 0.0004 as shown in Fig-

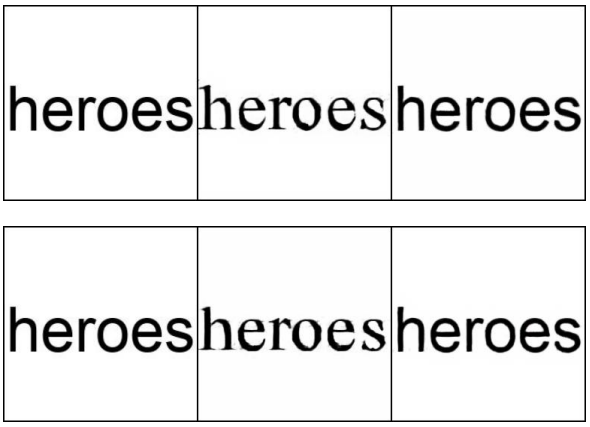


(a) Learning rate 0.0004 (above) and 0.0002 (below)



(b) Identity loss not included (above) and included (below)

Figure 7: Learning Rate and Identity loss ablation for $Arial \leftrightarrow Times$



(a) Learning rate 0.0004 (above) and 0.0002 (below)

(b) Identity loss not included (above) and included (below)

Figure 8: Learning Rate and Identity loss ablation for $Arial\, word \leftrightarrow Times\, word$

urer 7a and 8a.

Cycle-consistency weight λ and Identity weight β Again, we tried with λ values of 10 and 5, and for this case 10 turned out to work better. For β, including it (value 1) gave us slightly better results as demonstrated in Figures 7b and 8b.

7. Conclusion and Future Work

In this paper, we have tried to regenerate CycleGAN [30] from scratch, explore two of the already explored domains and finally apply it to font style transfer for single character and full words. We couldn't replicate all the possible domains and results that the original paper did because of time constraints. From the results we got, the attempt of applying it to font style transfer did look promising to some extent. However, there is still quite a lot of ablations and tunings that can be done - like possibly modifying the network architecture, using a different number of residual blocks (we used 9), trying out loss functions other than L_2 and L_1 norms, trying out batch and group normalizations instead of instance normalization etc. Also, our font and word images were all simple black ones with a plain white background, while real world posters and artistic drawings would contain fonts and backgrounds with varying colors and gradients, which is a definitely challenge to explore in the future.

Acknowledgements Dr. Kosta Derpanis and Matthew Kowal.

References

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

[2] S. Azadi, M. Fisher, V. Kim, Z. Wang, E. Shechtman, and T. Darrell. Multi-content gan for few-shot font style transfer. *CVPR*, 2014.

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

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

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

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

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

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

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

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

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

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

[13] 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.

[14] 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.

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

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

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

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

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

[20] 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.

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

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

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

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

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

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

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

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

[29] T. Zhou, P. Krahenbuhl, M. Aubry, Q. Huang, and A. A. Efros. Learning dense correspondence via 3d-guided cycle consistency. *CVPR*, 2016.

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