ACV Report

Word Count: 1435

Introduction

CycleGAN [4] style deep learning models can be used to map the style of a domain onto an input image. We attempt to tackle the problem of stylistic transfer between the domains of movies and games by utilising our own deep learning model and analyse the performance of the model on face-to-face and frame-to-frame problems.

Preprocessing (1.1, 3.1)

For our frame dataset, we extract images from each video randomly across the whole video, checking we never extract the same frame twice. This ensures that we have data from a wide range of different scenes, settings, and environments. We then split the images into three sets: train, validation, and test, at an 8:1:1 ratio. We use the train and validation sets to train the model – using the validation set to ensure that the model is not overfitting – and the test set to evaluate the success of the model. As the frame-to-frame task is difficult, we extract a total of 1500 frames per domain – 50 seconds worth of video.

We employ a similar method for the face dataset. We lower the faces per domain to 1000 – as this task is easier to train for and the process of face extraction is more computationally expensive.

For both frames and faces, we use 128x128 pixels as our input size. We chose this value for two reasons. Firstly, the size is a nice balance between maintaining resolution and reducing computation cost (memory and processor cycles). Secondly, one of our loss functions requires a square image (therefore we cannot preserve the original aspect ratio), which we explore later.

Model Architecture (2.1, 3.2)

GANimorph [1] is the inspiration for our model architecture. In its unaltered state, the GANimorph model consists of (at least) 23 convolutional layers within the generator – a network that is far too deep for our dataset size and hardware. Therefore, we reduce the number of layers by lowering the number of convolutions in the residual blocks to 2 and overall make the model shallower, as depicted in Figure 1.

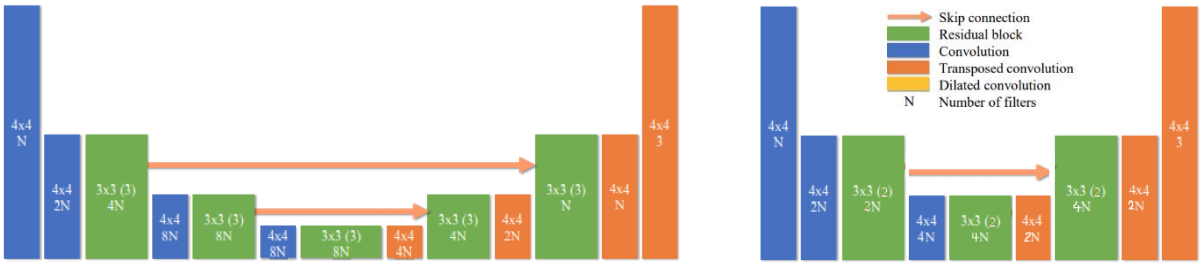

Figure 1: GANimorph generator model [1] (Left, N=64), our generator model (Right, N=64)

To shallow the discriminator, we remove a single convolutional layer and the 8-dilation layer from the discriminator, then reduce the number of features/filters from 64 to 32. This significantly reduces the number of trainable parameters in the discriminator (2.8 million) to be in line with the generator (5.7 million), as at N=64 the parameters are in excess of 10 million. The overall changes are highlighted in Figure 2.

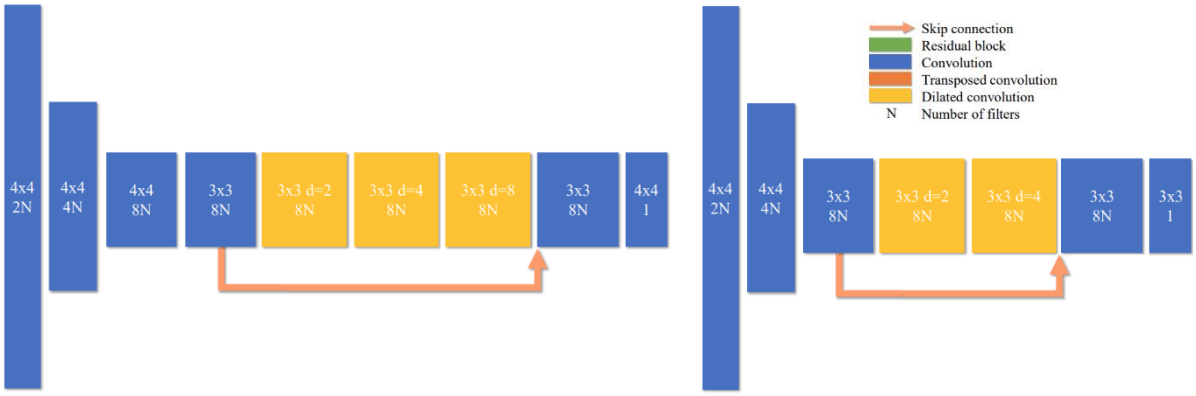

Figure 2: GANimorph discriminator model [1] (Left, N=64), our discriminator model (Right N=32)

Our loss functions significantly differ to the ones in GANimorph. We chose to incorporate 3 losses for our generator: adversary loss, cycle loss and identity loss. From our testing, the discriminator was strong enough with standard GAN loss (BCE), therefore, we do not add any additional loss terms there.

Adversary loss is standard GAN BinaryCrossEntropy (BCE) loss based on discriminator output. Adversary loss rational follows traditional GAN ideas, we learn by trying to beat the discriminator.

Cycle loss is the similarity the input image has to the output of running the input cyclically through the two generators. To do this, we calculate Multiscale-Structural Similarity Index Measure (MS-SSIM) [5] between the input and the cycle image, both of which must be square. As MS-SSIM is a similarity measure, we adapt it for use as loss by using 1 minus the MS-SSIM output. This loss allows us to minimise the features lost via style transfer by ensuring cycled images are similar.

Our final loss is identity loss, defined as the L1 Loss between the input and the generator output. The idea here is to ensure the input and output have a similar shape.

We introduce a set of hyperparameters for our losses in the form of a weight λ for each loss, to allow us to tune what the generator bases improvements on. We bound the weights via Equation 1.

$$\lambda_{adversary} + \lambda_{\text{cycle}} + \lambda_{\text{identity}} = 1$$

Equation 1: Generator Loss Weight bounding term

We chose the number of epochs based on training time – we aim that no training runs longer than 45 mins, though typical training times were 20-40 mins. All our final hyperparameters are summarised in Table 1.

	Frame-to-Frame Model		Face-to-Face Model	
Hyperparameter	Generator	Discriminator	Generator	Discriminator
Learning Rate	0.001	0.0001	0.001	0.0001
# of Features	64	32	64	32
$\lambda_{\text{adversary}}$	0.5	-	0.7	-
λ_{cycle}	0.3	-	0.2	-
$\lambda_{\text{identity}}$	0.2	-	0.1	-
Input Size	128x128		128x128	
# of Epochs	40		36	
Batch Size	1		1	
Dataset Size (domain)	1500		1000	

Table 1: Model Hyperparameters

Unconventionally, we chose the batch size for our model to be 1. We utilise InstanceNorm in our model, which is invariant to mean/features within single images, helping with brightness invariance. If we increase our batch size significantly, transition to BatchNorm may be required, which is only invariant to batch level mean/features.

Frame-To-Frame Performance (2.1)

Overall, frame-to-frame performance is poor. The model failed to converge on any overall meaningful output by our epoch limit. The outputs in Figure 3 are typical across all variants of the frame-to-frame model, in both directions.

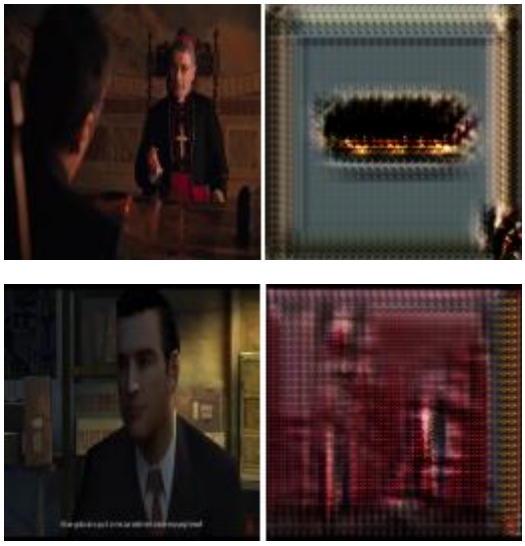

Figure 3: Frame-to-Frame Model Showing Noise
[Truth (Left) Generated (Right)]

However, this is not to say that the model cannot achieve a good result. Figure 4 shows an outputs (from middling epochs during training) which show the model is learning image structure instead of just unstructured noise. The style transferred colours are also emerging, with the brown colour not far from the orange tint we expect from the movie domain. With further hyperparameter tweaking, additional data, and longer training runs, we may see a model output that is acceptable.

Figure 4: Frame-to-Frame Model Output Showing Structure
[Game Truth (Left) Movie Generated (Right) Epoch ~21/40]

Face-To-Face Performance (3.2, 3.3, 3.4)

Figures 5 and 6 shows a good model fit in both directions, with the main facial features being preserved and the correct colour tones for each domain being applied. The output is superior to those shown in Figure 4, which only manage to capture the basic structure of the scene. This may be attributed to that faces have a defined structure, allowing the model to learn the features of a face with far less difficulty than the features of a full scene, which can significantly vary in shape, lighting etc.

The movie generated faces suffer from a loss of quality due to lack of invariance to light, the images in Figure 5 show dark patches in places where there should not be any. Furthermore, all generated images suffer from checkboard artifacts. To combat this, we attempted to use Resize-Convolutions [6]. This dampened the artifacts (Figure 7), but the model failed to fit to task, so we discarded the method moving forward.

Figure 5: Non-Augmented Dataset on Face Model (Game to Movie)

Figure 6: Non-Augmented Dataset on Face Model (Movie to Game)

Figure 7: Face Model with Resize Convolution [6]

We then applied data augmentation to the dataset, as described in Table 2. These methods allow the model to become robust to different situations and overall improve model quality.

Augmentation Type	Probability	Justification
None	0.2	Standard Feature learning
Horizontal Flip	0.2	Y-Axis mirror invariance
Vertical Flip	0.2	X-Axis mirror invariance
Gaussian Blur (k= 3)	0.2	Noise invariance
ColorJitter [12]	0.2	Lighting invariance

Table 2: Data Augmentation methods and rates

Figure 8: Augmented Dataset on Face Model (Movie to Game)

Figure 9: Augmented Dataset on Face Model (Game to Movie)

Compared to Figure 5, Figure 9 shows much better lighting due to the data augmentations. However, the blur may have resulted in loss of features for movie-to-game, the faces in Figure 6 are more defined than Figure 8 – though further training and a lower augmentation probability may mitigate this.

In terms of realism, the model is strong at mimicking the main features of the face, especially contours, but loses fine features such as the eyes and hair (blurred together). At this stage in training, the general features have been learnt as the discriminator can distinguish shape well, however, the generator can get away with excluding finer features, and there may not be sufficient data to learn the nuances of the face.

Real-world application (4.1, 4.2)

Remarkably, the data-augmented model applies itself well to a "full" size 720x720 image input [7] for transforming a series of frames into video [8]. It seems by training on smaller images, visual errors are minimised as opposed to increased when using larger input. Figure 10 and 11 show the model on 128x128 inputs, where we see significant blurring and loss of fine features. In contrast, overall video [8] quality is almost similar to the input and the learned tones of the movie domain have been successfully transferred. However, heavy pixilation/blur occurs in areas where large movements occur, as the model is not trained in temporal robustness.

Figure 10: Full Frame on Face Model (Movie to Game)

Figure 11: Full Frame on Face Model (Game to Movie)

To create a model for video inputs, we can utilise methods such as vid2vid [9]. While our current model uses single, isolated frames as training data, a video input model makes temporal estimations to create good outputs. Consequently, a [9] style model could take time into account and consider both neighbour frames (back/forward) of input data in order to make robust interpolations. A shallow enough model could allow for real-time prediction, though input size may have to be reduced.

Models such as SAGAN [10], AttentionGAN [11], utilise masks to focus the style transfer process. Given the rendering pipeline, we can create masks for individual scene elements (lighting, meshes etc) and create a model that uses the masks to inform the stylisation.

Conclusion

We have successfully implemented an image style transfer model between the movie and game domains. The overall quality of the system is good, though further work is required to combat blurring and define finer features. The model can be used on video inputs but temporal inconsistencies may appear.

References

[1] Gokaslan, A., Ramanujan, V., Ritchie, D., Kim, K.I. and Tompkin, J., (2018). Improving shape deformation in unsupervised image-to-image translation. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 649-665).

[2] Wang, T. C., Liu, M. Y., Zhu, J. Y., Liu, G., Tao, A., Kautz, J., & Catanzaro, B. (2018). Video-to-video synthesis. *arXiv preprint arXiv:1808.06601*.

[3] https://github.com/advaitrane/GANimorph_pytorch

[4] Zhu, J. Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision* (pp. 2223-2232).

[5] Z. Wang, E. P. Simoncelli and A. C. Bovik, (2003). Multiscale structural similarity for image quality assessment. The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, Vol.2. doi: 10.1109/ACSSC.2003.1292216. (pp. 1398-1402).

[6] Odena, et al. (2016). Deconvolution and Checkerboard Artifacts. Distill. http://doi.org/10.23915/distill.000

[7] original_vid.avi (Found in submission zip)

[8] transformed_vid.avi (Found in submission zip)

[9] Wang, T. C., Liu, M. Y., Zhu, J. Y., Liu, G., Tao, A., Kautz, J., & Catanzaro, B. (2018). Video-to-video synthesis. *arXiv preprint arXiv:1808.06601*.

[10] Zhang, H., Goodfellow, I., Metaxas, D., & Odena, A. (2019, May). Self-attention generative adversarial networks. In *International conference on machine learning* (pp. 7354-7363). PMLR.

[11] Tang, H., Liu, H., Xu, D., Torr, P. H., & Sebe, N. (2019). Attentiongan: Unpaired image-to-image translation using attention-guided generative adversarial networks. *arXiv preprint arXiv:1911.11897*.

[12] https://pytorch.org/vision/stable/transforms.html#torchvision.transforms.ColorJitter

Disclaimers

Training Hardware Specification

GPU: Tesla V100-SXM2-16GB

CPU: 2x Intel(R) Xeon(R) CPU

External Code

[3] was used to help define the initial GANimorph model in PyTorch, though we deviate significantly from this codebase.

Logging

Comet.ml was used to log losses and testing images (some are used in the report) throughout the training process. Please feel free to explore the loss graphs and epoch outputs for the models used in this report. Note: the duration metric is NOT an indicator of training time, the logger does not always start or finish at train time.

https://www.comet.ml/nikesh/style-transfer?shareable=tKSlQW6J1GDnW5vVrYYFgleqJ