Cycle-Wasserstein-GAN for CelebA-Bitmoji and SVHN-MNIST

Papers followed: Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks (Zhu et al.) and Wasserstein GAN (Arjovsky et al.)
Datasets Used: CelebA - https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html, Bitmoji - https://www.kaggle.com/datasets/romaingraux/bitmojis
Framework Used: PyTorch Lightning (1.6.4), PyTorch (1.11.0), TorchVision (0.12.0)

1) Code Details:

a) **CBCycleWGAN** – Class for Cycle Wasserstein GAN model for CelebA and Bitmoji datasets
   1. It has 2 critics – CriticC and CriticB, for CelebA and Bitmoji images respectively. Both the critics have 5 down-sampling layers and a linear layer at last to output the score for the image. Each layer is followed by PReLU activation function, a max pooling layer and a BatchNorm layer to act as a regulariser.
   2. Also has 2 generators – genC2B and genB2C, for translating images from CelebA to Bitmoji and from Bitmoji to CelebA respectively. Both the generators have 5 down-sampling convolution and 5 up-sampling deconvolution layers, with variable number of channels etc to be consistent with the required data dimensions. Each layer is followed by PReLU activation function and a BatchNorm layer to act as a regulariser.
   3. The model takes ncritic, ngen, cycle_weight and batch_size as parameters. ncritic dictates the frequency with which the critic will be called. ngen dictates the frequency with which the generator will be called. cycle_weight indicates how much weightage to give for cycle consistency loss.
   4. During the critic update, both the critics were updated using the usual Wasserstein loss. During the generator update, both the generators were updated using the usual Wasserstein loss and the cycle consistency loss.
   5. RMSProp was used as the optimiser. 2 optimisers were used. One for both the critics and one for both the generators. Their parameters were chosen from the Wasserstein GAN paper.
   6. In the on_train_batch_end method, the weights of the critics were clipped between -0.01 and 0.01 as instructed in the Wasserstein GAN paper. This method is called after weight updates are done for that batch.
   7. During training, in the CelebA – Bitmoji CycleWGAN, additional transformations were applied to the input data i.e., random horizontal flipping. This is used as a method to regularise the model, as advised in the CycleGAN paper. We did not apply this for the SVHN – MNIST pair as flipping would change the semantics of the image.

b) **SMCycleWGAN** – Class for Cycle Wasserstein GAN model for SVHN and MNIST datasets. Everything is similar to the CBCycleWGAN class except some changes in CNN configurations etc. due to changes in number of dimensions of the datasets.

c) **Training Utilities**: train_and_test: General function which allows running different configurations. It takes parameters such as max_epochs, which GPUs to use etc. Even takes the model_class and the model_kwargs to instantiate the model inside the function. It uses the Trainer class of PyTorch Lightning to train, validate and test the model on the given dataset and the model. It logs everything into a tensorboard (created inside the function), which can then be used to view loss curves.

d) **Data related utilities**: All the data loaders were kept inside the model. TorchVision was used to load the datasets in batches. To allow for loading from different folders etc., CombinedDataLoader approach was used.
e) All randomness is controlled by using seeds which ensures reproducibility.
f) **Plots and Analysis Utilities**: plot_translations: View 50 image translations from one dataset to another in a 10 X 10 grid

2) Architecture Choices & Details:

1. PReLU was used as activation function because of the findings of this study: https://arxiv.org/abs/1502.01852v1. It is essentially LeakyReLU where the slope parameter is kept learnable. Kaiming normal initialisation was used for the layer preceding this activation function.
2. HardTanh was used as the activation function at the last layer as the image is converted from 0 to 255 range to -1 to 1 by using a normalisation technique. As opposed to tanh, this function can output -1's and 1's. Xavier normal initialisation was used for the later preceding this activation function.
3. CycleGAN authors use image pool to update the discriminator. That was required because GAN training (on the usual objective) was unstable. We are using WGAN and hopefully won't get into such issues. Hence, we skipped keeping the pool of images.
4. Cycle Consistency Loss: Ideally, L1 norm should be added across dimensions and mean-ed across samples. In their implementation, authors have mean-ed across dimensions too, so we kept implementation same as them.

3) Experiments and Results:

The models has many hyperparameters worthy of tuning. Hyperparameters such as number of CNN layers, kernel size etc which are related to CNNs were fixed for the assignment. We want to understand the effects of hyperparameters introduced in the GAN model. In that line of thought, two hyperparameters were explored: ratio of number of generator and critic update frequency and cycle loss weight. Only some of the results are mentioned here to keep the report concise.

CelebA – Bitmoji

Different variations were tried. Some of the variations are shown in the Figure 1. The translations are not that good and so it is difficult to quantify which combination is doing definitely better. For Bitmoji to CelebA, it looks like Ncritic=1 and Ngen= 5 and CycleWeight = 1 is doing better. It seems to be converting it into some space where the bitmoji's effect seems to be going away. For CelebA to Bitmoji, it looks like Ncritic = 5 and Ngen= 1 and CycleWeight = 1 is doing better. It seems to be capturing the white background of the bitmoji space and transforming the CelebA images to that space. Between CycleWeight = 1 and 10, there doesn't seem to be much of a difference, visually.

Fig 1: Image translation results between CelebA and Bitmoji. Left side panels are from Bitmoji to CelebA. Right side panels are from CelebA to Bitmoji. Row wise panels have the respective frequency of updates and cycle weights mentioned

SVHN – MNIST

Different variations were tried. Some of the variations are shown in the Figure 2. The translations are not that good and so it is difficult to quantify which combination is doing definitely better. It seems

like the MNIST to SVHN generators are not able to understand what colour etc to add. They keep the original black and white intact. The SVHN to MNIST generators give a black background and a white blob, unable to understand which number to present.

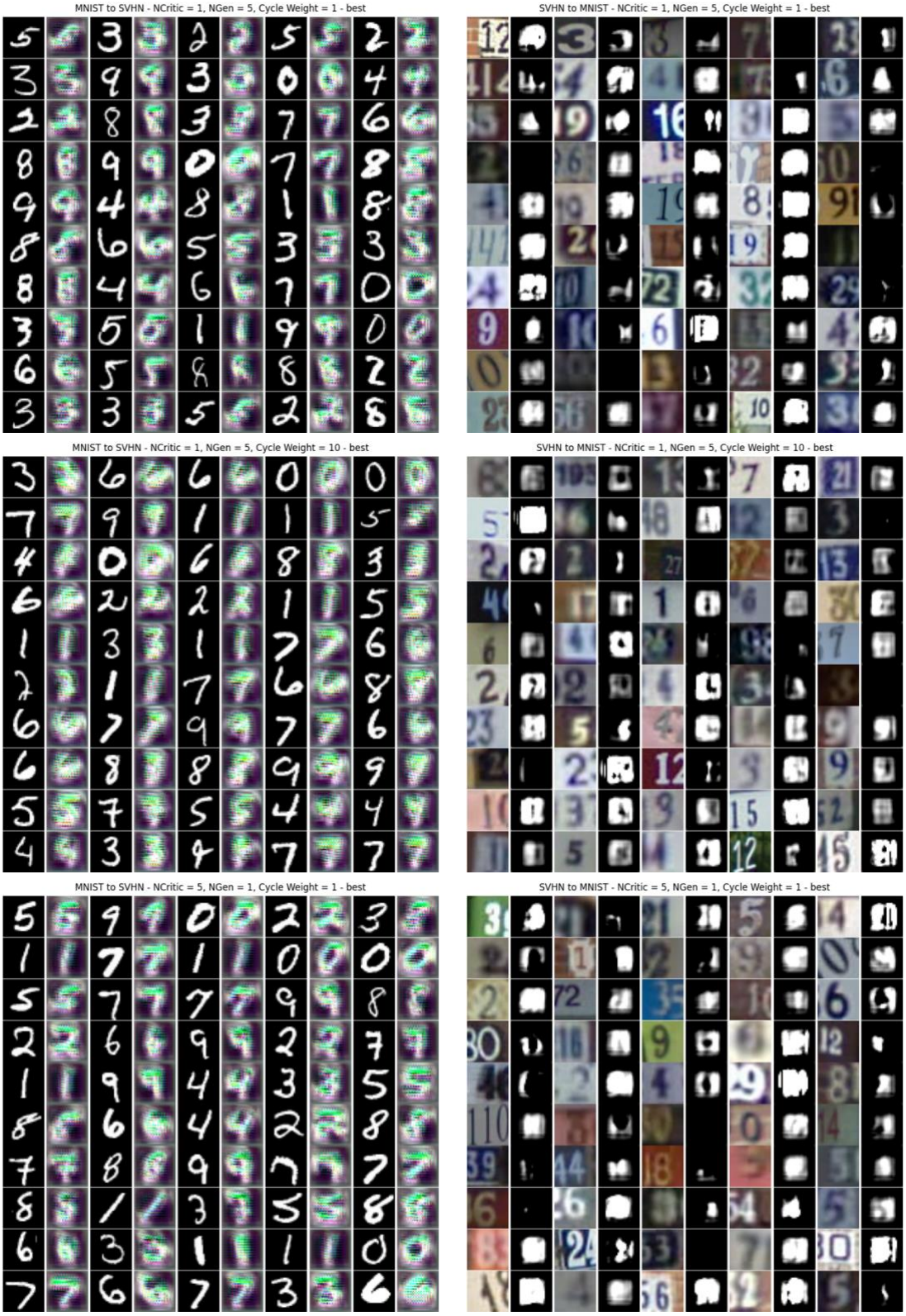

Fig 2: Image translation results between SVHN and MNIST. Left side panels are from MNIST to SVHN. Right side panels are from SVHN to MNIST. Row wise panels have the respective frequency of updates and cycle weights mentioned

The above results maybe because of one reason. The critics may be very powerful, causing generators to not learn much. We then investigated the loss curves and saw that critics loss decreases in the beginning itself and stays there while generators loss increases in the beginning and stays there. This could explain the results. A sample loss curves are shown in Figures 3 & 4.

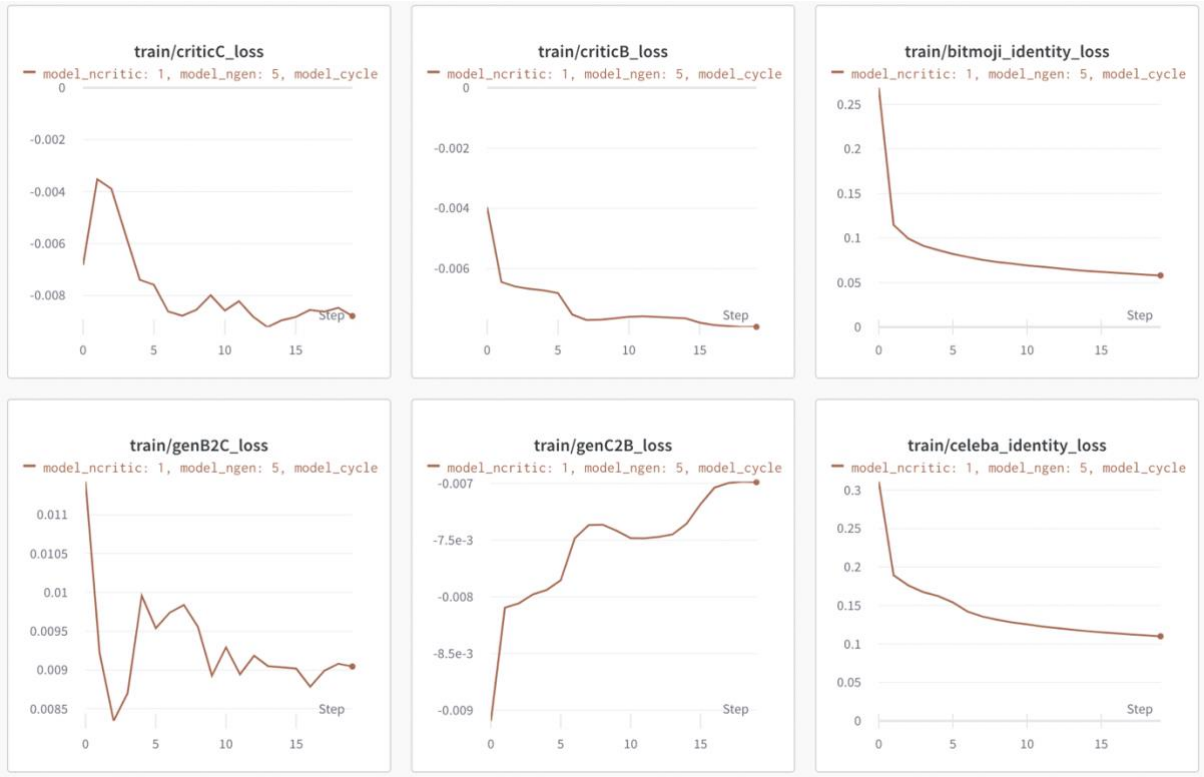

Fig. 3: Loss curves for the CelebA – Bitmoji dataset

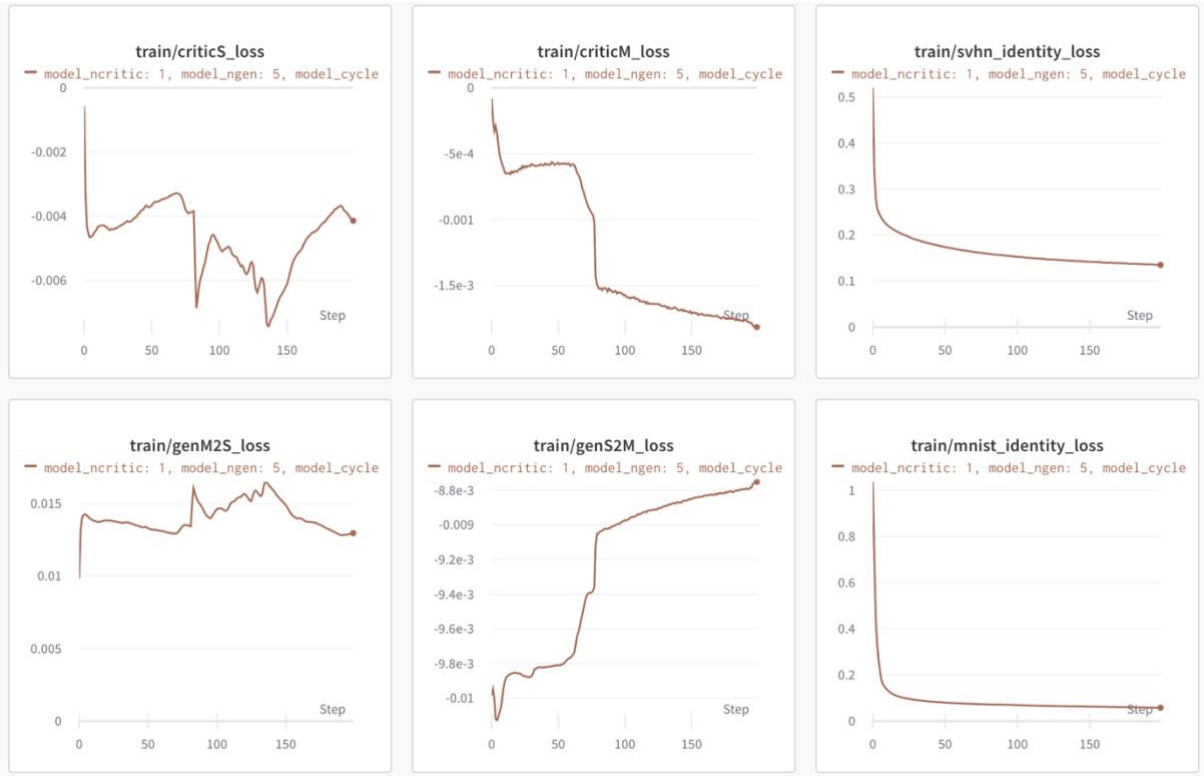

Fig. 4: Loss curves for the SVHN – MNIST dataset

4. Problems faced, solutions used:

1) Working with multiple datasets with large image sizes really pushed the need for high compute power. Under the computational constraints, we set the max epochs to 50 for the CelebA – Bitmoji CycleWGAN.
2) Many choices between CycleGAN and Wasserstein GAN papers were different. For example, the choice of optimisers. We used our best judgement in such confusions.

5. Conclusion & Future Work:

1) This question of the assignment was a heavy headscratcher as we had mix concepts from two very different problem statements and hence was a good exercise in helping us strengthen our concepts in both.
2) We will try this code with known and successful image translation datasets like painting or the horse-zebra one. If it doesn't work there, then it may point to some issue in the code!
3) We think we need to apply the ideas of general GAN training like making the discriminator weak etc. to get good translation results in these datasets
4) SVHN – MNIST combination though may seem translatable to the naked eye, it must be observed that SVHN has images which has multiple digits in them. Maybe working with images which have single digit may give expected performance.