Training CycleGAN using AE-based Initialization

COSC 525 Deep Learning

1st Ankush Patel
Min H. Kao Electrical Engineering and Computer Science
University of Tennessee Knoxville
Knoxville, US
apatel79@vols.utk.edu

2nd Ravi Patel
Min H. Kao Electrical Engineering and Computer Science
University of Tennessee Knoxville
Knoxville, US
98ravipatel@gmail.com

***Abstract*—Preliminary results in initializing a CycleGAN using the weights of an unbalanced autoencoder (AE) look promising. Previous work has been done on common GAN architectures such as DCGAN, LSGAN, and WGAN; however, we plan on applying this method to a more complex model CycleGAN [1]. CycleGAN is a type of generative adversial network (GAN) that uses two GANs to cycle between two image classes for example apples and oranges, which is explored in this paper [3]. The AE is a simple encoder and decoder model designed in PyTorch that uses MSE as the reconstruction loss. We initialize both generators within the CycleGAN using a trained AE for each image class prior to training the CycleGAN. Results for the AE-based CycleGAN show higher SSIM and FID scores against the normal CycleGAN, which are both structural measurements of GAN-related images. Additionally, we show higher quality images produced with the AE-based CycleGAN against the normal CycleGAN.**

***Index Terms*—GAN, CycleGAN, AE, SSIM, FID**

I. Introduction

There is a very common reoccurring issue with GAN-based architectures. They usually require many epochs to get reasonable quality images. Additionally, many of them are prone to mode collapse, meaning they are only able to produce one singular output or a small set of generalized outputs. Our goal is to produce a wide variety of better quality images in a small number of epochs. We are planning on initializing an existing GAN architecture, namely CycleGAN, with a pre-trained decoder from an AE. There is an existing implementation of CycleGAN that we plan to use as a baseline with MNIST and augment this implementation with a pre-trained AE (unbalanced GAN). Ideally, we will be able to compare the training performance of both GANs over some epochs. The only similar implementation we found related to this was a paper trained a variational autoencoder (VAE) within a CycleGAN; however, they did not initialize the weights of the generator with a pre-trained AE like we are proposing. Originally, we planned on using a VAE to initialize the CycleGAN; however, we ran into a few issues that we discuss in the technical approach.

II. Previous Work

The motivation for this project came from an article which stated the "stable training of the generator by preventing the faster convergence of the discriminator at earlier epochs [2]." We were intrigued by this statement and further investigated how we could apply this to other projects. This method can be used for all GAN architectures as long as the generator portion of the GAN can be implemented as a decoder within a VAE. Furthermore the weights of the decoder within the AE must be able to be transferred to the weights within the generator for initialization. This can be quite useful in GAN applications where quick prototyping and stable training is of importance.

The article we examined is titled: Unbalanced GANs: Pre-training the Generator of Generative Adversarial Network using Variational Autoencoder from the Arxiv website [2]. The article explains the benefits of using VAE and provides examples of the performance improvements on datasets such as MNIST and the LSUN Bedroom dataset. The article also discusses the algorithm used to train the Unbalanced GANs, and explains the steps needed to apply the method to other projects.

Autoencoding Beyond Pixels Using a Learned Similarity Metric is an article which is very well known for combining AE and GAN [3]. This article uses a three network structure, encoder, decoder, and discriminator. The article is cited in the first article we discussed acknowledging that it was a similar attempt of using VAE on a GAN.

Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, contains information about how a GAN was used to generate an image which is in a different format than the original [5]. Some examples stated in the article include, changing images of Monet to photographs, zebras to horses, and summer to winter. We plan on modifying the structure they used in the article by pretraining the generator and comparing the performance differences. The article states the results were obtained after performing 200 epochs, and we will try to obtain similar results using less epochs by pretraining the generator using an AE.

III. Technical Approach

The problem we aimed to solve was the CycleGAN architectures are prone to mode collapse and are sometimes unable to produce quality images in a short period of time. The goal of a CycleGAN architecture is to transform images between two classes. For example, a very common data set used for a

CycleGAN is horses and zebras. The goal of the CycleGAN in that example is to convert original pictures of horses to zebras and vice versa. The problem which arises when doing this is mode collapse, which means the architecture is only able to produce a small set of similar outputs. For most GAN architectures, the goal is to generate a wide variety of pictures and not the same type of pictures. Additionally, some CycleGANs are not able to produce high quality transformation. The pictures generated are blurry and do not accurately resemble the target class. Our goal is to improve these problems, by introducing a VAE to the CycleGAN.

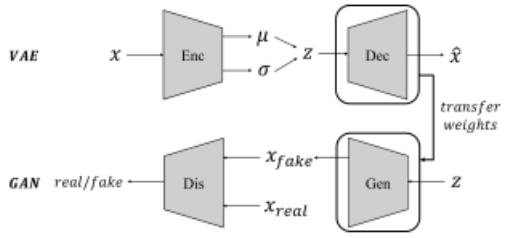
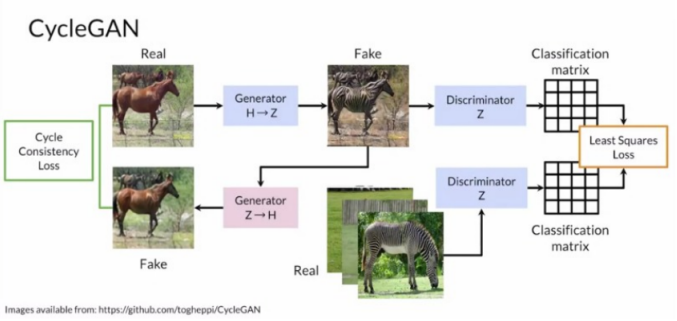


Fig. 1. VAE and GAN architecture [2]

Most VAE architectures consist of a Encoder and a Decoder. The role of the Encoder is to encode the provided images into a certain latent space. Then the job of the decoder is to produce the original image from the latent space. The VAE itself has two separate loss functions: one for reconstruction e.g. MSE and another for KL-divergence, which measures the difference between the latent distribution and normal distribution. The KL-divergence term is computed between the means and standard deviation in the encoding layer (shown in Figure 1).

A GAN architecture (shown in the bottom of Figure 1) is made up of a discriminator and a generator. The role of the discriminator is to differentiate between real and generated images. The role of the generator is to produce fake images to try and trick the discriminator.



The figure above shows the CycleGAN we plan to use for the experiments. It consists of two GANs one which generates images from class A to B and another which generates images from class B to A. The original model keeps training until both discriminators are no longer able to tell the real images apart from the generated ones.

Our goal is to use the generators that CycleGANs implement as the decoder of the AE. Then, we will train the AE on the data set, which will improve the initial random set of weights of the decoder. Finally, we will transfer the decoder back to the CycleGAN's and begin training/generating images as shown in Figure 1. By doing this, we have essentially initialized the weights of the generator in a GAN with the decoder of a AE. We hope this will result in better quality images from the generator in a smaller amount of epochs.

Note, that we did not use a VAE implementation as shown in the original paper since initial implementations led to a vanishing KL term where the data distribution of the latent space is essentially the normal distribution [2]. We attempted to fix this issue; however, it led to a large time sink as the KL term was vanishing and we were left with only the reconstruction loss. Thus, we opted to design a separate model instead. The original VAE attempt is attached with this project.

The AE implementation that is shown below. As you can see, the model consists of about 6.1 million parameters and the encoding layer is made up of 2D CNNs using a GELU activation function. GELU is similar to RELU except that it weights inputs by their value rather than gates inputs by their sign [6]. This leads to better performance in computer vision tasks and helps avoid the vanishing gradients problem. In the decoder model, we unpack the latent space, and perform 2D CNN transpose operations to unpack the encoded latent space, we apply Tanh to obtain our decoded image, and feed this into an underlying Resnet generator that the CycleGAN uses. Using MSE as our reconstruction loss function, we seed the weights of the resnet generator within our AE for eventual transfer into the CycleGAN model. We repeat this training process for both classes in our case: Apples and Oranges.

We train the CycleGAN using no AE initialization and train it with AE-based initialization for both the generators within the CycleGAN. We take both of these CycleGAN's and derive some quantitative and qualitative metrics discussed further below.

```
Layer (type:depth-idx)                                  Output Shape              Param #
Autoencoder                                             --                        --
├─Encoder: 1-1                                          [1, 256]                  --
│    └─Sequential: 2-1                                  [1, 256]                  --
│    │    └─Conv2d: 3-1                                 [1, 64, 16, 16]           1,792
│    │    └─GELU: 3-2                                   [1, 64, 16, 16]           --
│    │    └─Conv2d: 3-3                                 [1, 64, 16, 16]           36,928
│    │    └─GELU: 3-4                                   [1, 64, 16, 16]           --
│    │    └─Conv2d: 3-5                                 [1, 128, 8, 8]            73,856
│    │    └─GELU: 3-6                                   [1, 128, 8, 8]            --
│    │    └─Conv2d: 3-7                                 [1, 128, 8, 8]            147,584
│    │    └─GELU: 3-8                                   [1, 128, 8, 8]            --
│    │    └─Conv2d: 3-9                                 [1, 128, 4, 4]            147,584
│    │    └─GELU: 3-10                                  [1, 128, 4, 4]            --
│    │    └─Flatten: 3-11                               [1, 2048]                 --
│    │    └─Linear: 3-12                                [1, 256]                  524,544
├─Decoder: 1-2                                          [1, 3, 32, 32]            --
│    └─Sequential: 2-2                                  [1, 2048]                 --
│    │    └─Linear: 3-13                                [1, 2048]                 526,336
│    │    └─GELU: 3-14                                  [1, 2048]                 --
│    └─Sequential: 2-3                                  [1, 3, 32, 32]            --
│    │    └─ConvTranspose2d: 3-15                       [1, 128, 8, 8]            147,584
│    │    └─GELU: 3-16                                  [1, 128, 8, 8]            --
│    │    └─Conv2d: 3-17                                [1, 128, 8, 8]            147,584
│    │    └─GELU: 3-18                                  [1, 128, 8, 8]            --
│    │    └─ConvTranspose2d: 3-19                       [1, 64, 16, 16]           73,792
│    │    └─GELU: 3-20                                  [1, 64, 16, 16]           --
│    │    └─Conv2d: 3-21                                [1, 64, 16, 16]           36,928
│    │    └─GELU: 3-22                                  [1, 64, 16, 16]           --
│    │    └─ConvTranspose2d: 3-23                       [1, 3, 32, 32]            1,731
│    │    └─Tanh: 3-24                                  [1, 3, 32, 32]            --
│    └─ResnetGenerator: 2-4                             [1, 3, 32, 32]            --
│    │    └─Sequential: 3-25                            [1, 3, 32, 32]            4,297,219
==========================================================================================
Total params: 6,163,462
```

IV. Dataset and Implementation

We were able to use two different data sets to test our VAE and CycleGAN architectures. We used the MNIST data set as our base case to test to see if the VAE containing the CycleGAN's generator structure as the decoder, would produce acceptable images. Once we were able to verify this, we began testing with the apples and oranges dataset. However, as we mentioned earlier this did not work out as the

KL term appeared to vanish. Thus, we designed a new AE model pictured above and began testing with the apples and oranges dataset. The data set itself was broken into training and testing datasets. There were 995 pictures of apples and 1019 pictures of oranges in the training dataset. There were 266 pictures of apples and 248 pictures of oranges in the testing dataset. Finally, we rescaled the original dataset from 256x256 to 32x32 due to performance constraints in training.

As far as hyper parameter optimization is concerned, we stuck with a small latent space of size 16 which worked well for image reconstruction examples - anything lower and the image would become grainy and anything higher and the image would essentially stay the same. It is important to mention that the CycleGAN uses varying underlying implementations for the generator model primarily consisting of resnet blocks. They had two such variants with 6 blocks and 9 blocks. The 6 block variant had 7.8 million parameters and the resnet 9 block variant had 11.4 million parameters. We opted to design a separate underlying generator model that only used 3 blocks due to computational resources, which gave us 4.3 million parameters to work with. Additionally, we varied batch size - with this type of model we found that the smaller the batch size the better results we obtained so we stuck with a default value of 1 in the batch size for both the AE and the CycleGAN training. Finally, we selected a learning rate of 1e-6 for the AE architecture since anything larger resulted in mode collapse for the AE.

We used Google Colab Pro; thus, we had access to a GPU. Training time for both the AE and CycleGAN was substantial regardless of whether the CycleGAN was AE based or not. The final AE took about 10 seconds to train per epoch for a total of 500 epochs. This resulted in about 3 hours of total training time for both combined Apple and Orange datasets. Finally, the CycleGAN took about 50 seconds to train per epoch for a total of 75 epochs or 1 hour for both the non-AE and AE based examples.

A. Evaluation Metrics

It can be a little difficult to evaluate how well a CycleGAN is performing, since the loss usually fluctuates, making it hard to determine when the model has converges. We were able to find a few methods to determine how well our CycleGAN was performing. The first method is visually inspected the images produced by the generator after a certain number of epochs. This is the method most commonly used since the quality of the image is subjective to many things, and a human eye can better determine the overall quality and accuracy of the images generated than a score. However, this can be time consuming and humans are also subjective; therefore, we decided to use two other metrics.

The first metric we used was FID (Frechet Inception Distance), which is the distance between the curves of two images. Additionally, the score accounts for the position and the order of the curves [4]. The score is calculated by first training a classification model on the images provided. Then the activation of an intermediate layer is used to compute the metric. We used an InceptionV3 as our classification model with average pooling and input shape of (299, 299, 3). For the implementation we used, we first scaled the input images and trained the InceptionV3 model. Then we took the outputted activations and computed the mean and covariance. We used the mean and covariance to compute the FID score. The best FID score has a value of 0 and the higher the score the more different the images are. The second metric we used was SSIM (Structural Similarity), which determines the structural differences between two images. It is able to do this by taking into account three aspects of an image, Luminance, Contrast, and Structure [1]. For our implementation we were able to call the structural similarity function provided by the skimage.metrics library. The score is calculated based on two images, and we were able to calculate the score for all images and produce line and box plots. SSIM is a score bounded between 0 and 1, with 1 being the best possible score that can be achieved.

The first experiment we conducted was to see if we can obtain at least similar results to the original CycleGAN without any modifications. Therefore, we ran a experiment on the apples and oranges dataset and trained the AE for 25 epochs and the CycleGAN for 15 epochs. We were relieved to see we obtained very similar results as the original CycleGAN. The figures below show the SSIM scores obtained from the testing images of apples and oranges. The figures show how were closely matched the performance of the original when the AE was only trained for a few number of epochs. We also include the score between the reconstructed images and the fake (generated images). The reconstructed images are passed through the model to try to generate the original image from the generated images. We were able to see that our reconstructed images matched the original images very well showing the model was cycle-consistent.

V. Experiments and Results Analysis

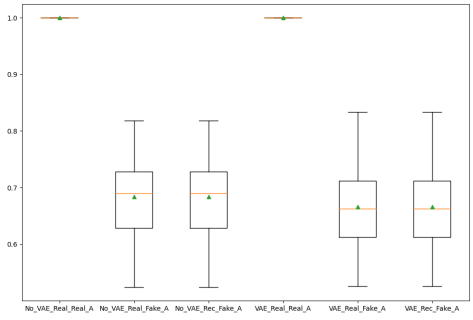


Fig. 2. The box plot of the SSIM scores obtained from training the AE for 25 epochs and the CycleGAN for 15 epochs on the Apples data set.

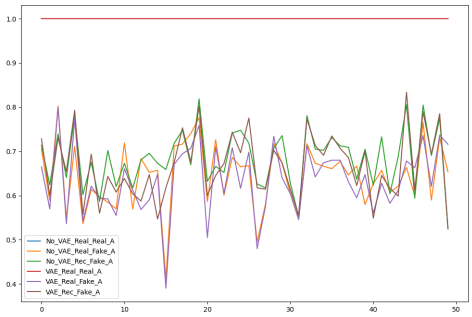


Fig. 3. The line plot of the SSIM scores obtained from training the AE for 25 epochs and the CycleGAN for 15 epochs on the Apples data set.

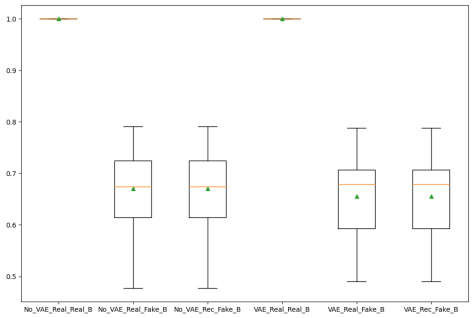


Fig. 4. The box plot of the SSIM scores obtained from training the AE for 25 epochs and the CycleGAN for 15 epochs on the Oranges data set.

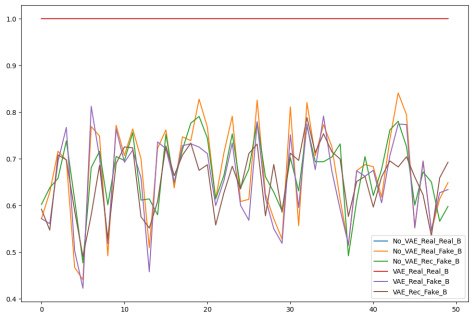


Fig. 5. The line plot of the SSIM scores obtained from training the AE for 25 epochs and the CycleGAN for 15 epochs on the Oranges data set.

For the second experiment we decided to increased the number of epochs the AE trained for to 500 epochs, and we trained the CycleGAN for 75 epochs. The figures below show the new SSIM scores obtained between all of the images. We were a bit unhappy to see that we had only slightly improved the scores only for the Apples data set as shown in Figure 6. To our surprise the oranges performed slightly worse than the original model.

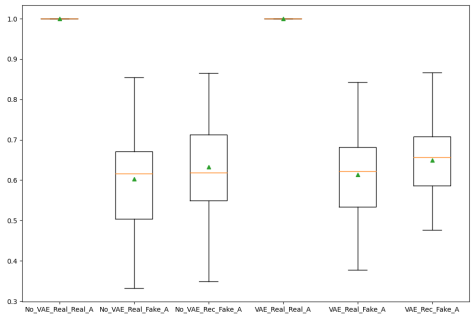


Fig. 6. The box plot of the SSIM scores obtained from training the AE for 500 epochs and the CycleGAN for 75 epochs on the Apples data set.

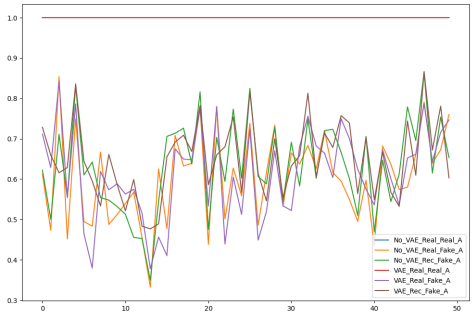


Fig. 7. The line plot of the SSIM scores obtained from training the AE for 500 epochs and the CycleGAN for 75 epochs on the Apples data set.

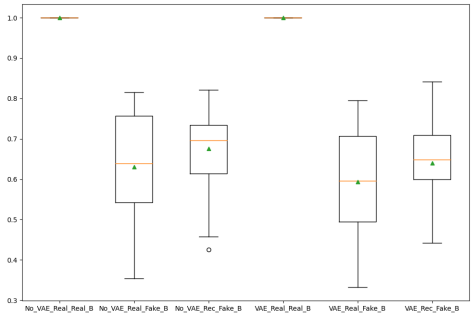


Fig. 8. The box plot of the SSIM scores obtained from training the AE for 500 epochs and the CycleGAN for 75 epochs on the Oranges data set.

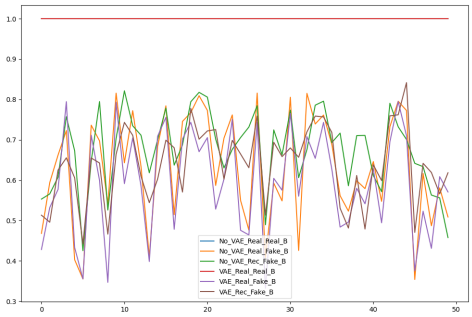


Fig. 9. The line plot of the SSIM scores obtained from training the AE for 500 epochs and the CycleGAN for 75 epochs on the Oranges data set.

For both of our experiments we also recorded the FID scores of all the images. The FID score is calculated between two sets of images and were were able to calculate the score for the Real and Fake images as well as the Reconstructed and Fake images.

25 Epochs	FID
No VAE Real and Fake Apples	288.733
No VAE Rec and Fake Apples	248.116
VAE Real and Fake Apples	275.323
VAE Rec and Fake Apples	234.695
No VAE Real and Fake Oranges	253.719
No VAE Rec and Fake Oranges	257.986
VAE Real and Fake Oranges	259.028
VAE Rec and Fake Oranges	247.626

Fig. 10. The FID scores when the AE was run for 25 epochs and the CycleGAN was run for 15 epochs.

500 Epochs	FID
No VAE Real and Fake Apples	272.411
No VAE Rec and Fake Apples	223.042
VAE Real and Fake Apples	262.127
VAE Rec and Fake Apples	231.655
No VAE Real and Fake Oranges	280.329
No VAE Rec and Fake Oranges	255.039
VAE Real and Fake Oranges	289.917
VAE Rec and Fake Oranges	256.814

Fig. 11. The FID scores when the AE was run for 500 epochs and the CycleGAN was run for 75 epochs.

The final experiment we conducted was to see if we are able to obtain better results in a smaller number of epochs. To determine this we trained the two different CycleGANs, one with a AE and one without, and calculated the SSIM scores of the images after every fifth epoch for a total of 75 epochs. We were again disappointed to see that for some reason we obtained the same score for every epoch. After examining the images they appeared to be the exact same. Figure 12, show how the score for both models did not change with the number of epochs. Since this experiment took some time to perform and obtain the scores, we were not able to change the hyper parameters and try to find the issue. However, the figure shows also shows that adding the AE obtained slightly better scores, only that the scores did not improve per epoch. We were satisfied with that result.

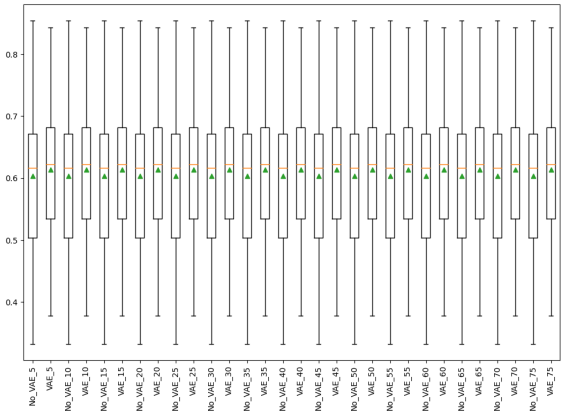


Fig. 12. The SSIM scores from the real and fake images every fifth epoch for both models.

Throughout our experiment we also looked at the images our model was generating. The figures below show the images generated from one of images from our apples data set. Figure 13 is the original image which was passed through each model we tested. Figures 14, 15, 16, and 17 show the difference between the generated images produced from the original CycleGAN (left) and CycleGAN initialized with the AE (right). The figures show the reconstructed and fake images produced by the CycleGAN when they were trained for a different number of epochs. We concluded from the images that our AE model with 25 epochs produced the best image, the shape of the new oranges did not get deformed and the color was also correct. We felt the original model had a few more red pixels in the generated image. We did not like the images generated from the models run for more epochs, since the colors began to change and the images were a little deformed.

Fig. 13. The original real picture of the apples.

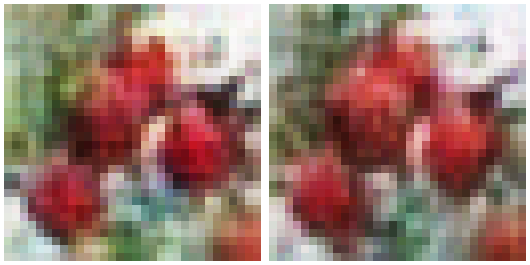


Fig. 14. The reconstructed image of the apples when the CycleGAN was trained for 15 epochs and the AE was trained for 25 epochs.

Fig. 15. The reconstructed image of the apples when the CycleGAN was trained for 75 epochs and the AE was trained for 500 epochs.

Fig. 16. The fake (generated) image of the apples as oranges when the CycleGAN was trained for 15 epochs and the AE was trained for 25 epochs.

Fig. 17. The fake (generated) image of the apples as oranges when the CycleGAN was trained for 75 epochs and the AE was trained for 500 epochs.

VI. Conclusion

With this project, we were able to learn about the effects of using an AE with a CycleGAN. We were able to see the how the images produces by the CycleGAN slightly improved when initialized with the weights from the AE. We were able compare the images using the FID and SSIM score. The experiments show how there is a slight increase in scores when using the AE. We also saw how the structure and the color of the images were slightly better in our opinion. We were disappointed to know the scores did not improve with the range of epochs we tested. We believe this might be due to the range and increment we used, since the 500 epoch AE images produced different scores. The slight increase in the results has given us some motivation to continue to work on the project in the future. We believe with more time, we will be able to find the right models and configurations of hyper parameters to obtain far better results. We also had to reduce the size of our images from 256x256 to 32x32 pixels to train the models faster. In the future we can use better quality full sized images. Most of the time was spent trying to obtain good results on the AE, so we can use it for the CycleGAN. We experimented with few different VAE and AE models and chose the one that generated the best images. In the future, we hope to circle back to the VAE model and devise a way to deal with the vanishing KL term. Since our data set contained a variety of apples/oranges taken in multiple locations with a different number of apples/oranges in each image, the initial models we tested would only produce a large circular red and orange blobs. After multiple experiments with different models and hyper parameters, we were finally able get slightly blurry images of the original images with an AE, and we initialized the CycleGAN with those weights to get better FID and SSIM scores than the original model.

References

[1] Datta, Pranjal. “All about Structural Similarity Index (SSIM): Theory + Code in Pytorch.” Medium, SRM MIC, 4 Mar. 2021, https://medium.com/srm-mic/all-about-structural-similarity-index-ssim-theory-code-in-pytorch-6551b455541e.

[2] Ham, H., Jun, T. J., Kim, D. (2020, February 6). Unbalanced gans: Pre-training the generator of generative adversarial network using variational autoencoder. arXiv.org. Retrieved March 31, 2022, from https://arxiv.org/abs/2002.02112

[3] Larsen, A. B. L., Sønderby, S. K., Larochelle, H., and Winther, O. Autoencoding beyond pixels using a learned similarity metric. arXiv preprint arXiv:1512.09300, 2015. Retrieved March 31, 2022, from https://arxiv.org/pdf/1512.09300.pdf

[4] Thakur, Ayush. “How to Evaluate Gans Using Frechet Inception Distance (FID).” Wamp;B, 15 Jan. 2021, https://wandb.ai/ayush-thakur/gan-evaluation/reports/How-to-Evaluate-GANs-using-Frechet-Inception-Distance-FID—Vmlldzo0MTAxOTI.
[5] Zhu, J.-Y., Park, T., Isola, P., Efros, A. A. (2020, August 24). Unpaired image-to-image translation using cycle-consistent adversarial networks. arXiv.org. Retrieved March 31, 2022, from https://arxiv.org/abs/1703.10593
[6] Hendrycks, D., Gimpel, K. (n.d.). Gaussian Error Linear Units (GELUs). arxiv. Retrieved May 17, 2022, from https://arxiv.org/pdf/1606.08415.pdf

VII. Appendices

A. Code Design

The general structure of the code is provided in ’ae-cyclegan.ipynb’. An Encoder, Decoder, and Autoencoder classes are added - each are designed and written in Pytorch. They all include forward functions to show how the layers are linked together and the constructors initialize those layers. The model itself is entirely sequential. Note, that the Decoder layer implements the CycleGAN’s generator at the end and uses that as the final output. The Autoencoder class has a few additional functions that compute the MSE reconstruction loss, configures optimizers: Adam and learning rate scheduler, along with training step and validation step functions. Finally, PyTorching Lightning trainer is used to train the autoencoder and is monitored via tensorboard. Once AE models have been trained, the code saves the generator weights from the decoder within the autoencoder for usage in the cycle gan. If the user provides the option ’–vae ”store_true”’ to the cycle-gan train.py function, the augmented CycleGAN will utilize ’models/vae-gen(A—B)-model.pt’ to initialize the generators within the CycleGAN. Finally, these are trained and tested, and a subsequent visualization is shown to illustrate the difference between using no vae and with vae as shown in latter figures. We had a few scripts to help process the data and generate charts showing the scores. The resize.py program created resized images to be trained on the models. We were able to use the PIL and Numpy libraries to resize the images. The scores.py program creates the plots showing the two different scores we used, FID and SSIM. The scores file contains the website which was used to write the code to calculate the FID score and the SSIM score was calculated using a library. The function from the website was modified to use our images. To calculate the FID score there is a function that takes two sets of images and outputs the score. It calls two helper functions one to scale the image and the second to calculate the FID given the InceptionV3 model and scaled images. Finally the program has two function which uses the scores functions to compute the scores for the CycleGAN models. The first function produces line and box charts for a CycleGAN with and without a VAE. The second function produces a box chart showing the differences in the SSIM score for CycleGAN models trained for every fifth epoch.

B. Workload Distribution

The workload was evenly distributed between the both of us. Ankush trained the VAE and CycleGAN models on the data set. He was able to modify the hyper parameters as he trained the different models and saved the images produced by the model. Ravi helped create some functions to help with training such as the resize images script and some functions to visualize the images generated by the models. Ravi also wrote the code to calculate the FID and SSIM scores, as well as the code to generate the charts of the scores from the images. Both of the team members equally contributed to writing the report.