Project Report for EECS 6322

Shivani Samir Sheth
Department of Electrical Engineering and Computer Science
York University
Toronto, ON M3J 1P3
shivs29@yorku.ca

Gursahaj Kohli
Department of Electrical Engineering and Computer Science
York University
Toronto, ON M3J 1P3
gursahaj@yorku.ca

Mamta Narang
Department of Electrical Engineering and Computer Science
York University
Toronto, ON M3J 1P3
mnarang@yorku.ca

Presentation Video: https://www.youtube.com/watch?v=q8Bsk0GanjU

GitHub Repository: https://github.com/Shivani29sheth/DNN-Personification

Additional Resources (Optional): https://drive.google.com/file/d/1s5Xtu7x6faKmwLt2Ap3p7yoxeBPyWvzI/view?usp=sharing

33rd Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

Reproducibility Summary

Scope of Reproducibility

Our project focuses on reproducibilty of two papers belonging to similar domains, yet different from each other in terms of performance and produced output. Paper [1] deals with local style transfer of the facial features from source image to destination image. Paper [2] deals with simultaneous transfer of the facial features from source to target image with multiple domains, targeting multiple facial features in one go. In order to reproduce both the papers, we implemented the basic functions of GANs. Since both the papers belong to the same domain, i.e. both of them are dependent on the StyleGAN for facial feature transfer, we tend to reproduce these basic GAN functions in order to get the desired results claimed by the papers.

Methodology

In order to get similar results, we reproduced several functions of GANs; we applied the Truncation method that provided a trade-off between image quality or fidelity and image variety, the mapping network that randomly samples a point from the latent space as input and generates a style vector which is transformed and incorporated into each block of the generator model, adaptive instance normalization, that involve first standardizing the output of feature map to a standard Gaussian, then adding the style vector as a bias term and progressive growing that helps grow both the generator and discriminator progressively. These functions were completely implemented from scratch and were adjusted with other functions from the author's code. In order to perform these functions, we had to study all the basics of GANs that took a lot of time to understand the concepts, but once we did that, we were able to apply these functions to both the papers to get desired results.
The training time for the first paper was based on training of StyleGANs that took nearly 1.5 days and training was done on Google Colab PRO, so the notebook rank continuously for 12 hours and got disrupted so we used to save the model state and delete the previous state to continue the training when the notebook shut itself.
For the second paper, it took nearly 3 days and it also faced the same disruption problems so we saved the models in as checkpoints and used the latest checkpoints as and when it got disconnected. This helped us in achieving the desirable results within our constraint resources.

Results

The results were pretty amazing when we compared to the actual paper. But as we trained on Celeb faces dataset, so the resolution was 512×512 because of which the quality images were not produced. The distortions in the generated faces can be seen in the Github GIF where the images tried to give human faces output but failed at some points. So we can say that for the first paper, 85% accuracy was obtained but some images were very distorted and style transfer performed poorly. For the second paper, we tried to get similar output as claimed by the authors but for low resolution images, the output was not good. Although, for the high resolution images where the face was visible, the output was similar to what was expected. So we can say that we got an accuracy of 70%.

What was easy

For the first paper, understanding the code was easier than running the code. The code was structured in a similar manner as they explained in the paper so figuring the structure and top-down flow of code was relatively easy.
For the second paper, the code was easy to understand and steps were given that directed the training and sampling procedures but the theory behind implementing the code was a bit complex.

What was difficult

For paper [1], the authors code was very difficult to run. The code was commented but still very difficult to understand. So, modifications had to be done in order to run the authors code and hence, the direct implementation of the author's code was not possible. Some parts of the code were broken and needed to be fixed. Also, the code understanding was tough. For the paper [2], the code was easy but resources were a big constraint. We did not have the hardware to even test the output. So we had to purchase Google Colab Pro in order to run the code. Also, dataset collection was a bit tough because finding 512×512 dataset was a bit tough and the implementations were also for 1024×1024 images.

Communication with original authors

Since the papers were written well and author code was available, we did not feel the need to contact the authors.

1 Introduction and Paper Summary

Generative Adversarial Networks (or GANs for short) are one of the most popular Machine Learning algorithms developed in recent times. Generative Adverserial Networks (GANs) are very successfully applied in various computer vision applications, e.g. texture synthesis [3],[4],[5], video generation [6], [7], image-to-image translation [8],[9],[10] and object detection[11]. In the few past years, the quality of images synthesized by GANs has increased rapidly. Compared to the seminal DCGAN framework[12] in 2015.
The current state-of-the-art GANs can synthesize at a much higher resolution and produce significantly more realistic images. Among them, StyleGAN[1][13] makes use of an intermediate *W* latent space that holds the promise of enabling some controlled image modifications. We believe that image modifications are a lot more exciting when it becomes possible to modify a given image rather than a randomly GAN generated one. This leads to the natural question if it is possible to embed a given photograph into the GAN latent space. Generative Adversarial Networks are composed of two models: The first model is called a Generator and it aims to generate new data similar to the expected one. The Generator could be asimilated to a human art forger, which creates fake works of art. The second model is named the Discriminator. This model's goal is to recognize if an input data is 'real' — belongs to the original dataset — or if it is 'fake' — generated by a forger.

1.1 Summary of Papers

While the quality of GAN image synthesis has improved immensely in recent years, the ability to control and condition the output is still limited. Focusing on ***StyleGAN***,[1] introduced a simple and effective method for making local, semantically-aware edits to a target output image. This is accomplished by borrowing elements from a source image, also a GAN output, via a novel manipulation of style vectors. Semantic editing is demonstrated on GANs producing human faces, indoor scenes, cats, and cars and measured the locality.
However in [2], the authors proposed ***StarGAN*** v2, a single framework that tackles both and shows significantly improved results over the baselines. Experiments on CelebA-HQ and a new animal faces dataset (AFHQ) validate the model's superiority in terms of visual quality, diversity, and scalability. To better assess image-to-image translation models, the authors released AFHQ, a high-quality dataset containing animal faces with large inter- and intra-domain differences.

2 Scope of reproducibility

In the first paper[1], the work proposed by the authors maps any one feature of a reference image at a time onto the target (or source) image. We reproduced and modified the mapping of the features from source to target and extend it to transfer multiple features at once from the reference image to the source image.
In [2], the work proposed by the authors map all facial features from source image to the reference image using StarGANs. We reproduced the mapping of multiple facial features across different styles of a domain. For example, the features of a source image will be mapped to multiple styles in a single domain.

- Compare variations of the model: For paper [1], we implemented the mapping of one feature at a time vs the mapping of multiple features at once and compare the results of the two variations. For paper [2], we a ran the baseline model (StarGANs) with additional layers involving a multi-class discriminator, tuning layer, latent code injection layer, style code layer, and a diversity regularization layer and compare the results of the different variations of the model.
- **Truncation trick**:The truncation trick involves using a different distribution for the generator's latent space z during training than during inference or image synthesis. A Gaussian distribution is used during training, and a truncated Gaussian is used during inference. This is referred to as the "truncation trick.
- Mapping Layer: A map layer is a GIS database containing groups of point, line, or area (polygon) features representing a particular class or type of real-world entities such as customers, streets, or postal codes. A layer contains both the visual representation of each feature and a link from the feature to its database attributes
- Noise injection: Injecting noise in the input to a neural network can also be seen as a form of data augmentation. Adding noise means that the network is less able to memorize training samples because they are changing all of the time, resulting in smaller network weights and a more robust network that has lower generalization error.

- Adaptive instance normalization: At the heart of our method is a novel adaptive instance normalization (AdaIN) layer that aligns the mean and variance of the content features with those of the style features. Our method achieves speed comparable to the fastest existing approach, without the restriction to a pre-defined set of styles.
- Progressive growing:We describe a new training methodology for generative adversarial networks. The key idea is to grow both the generator and discriminator progressively: starting from a low resolution, we add new layers that model increasingly fine details as training progresses. This both speeds the training up and greatly stabilizes it, allowing us to produce images of unprecedented quality, e.g., CelebA images.

3 Methodology

We applied the Truncation method that provided a trade-off between image quality or fidelity and image variety, the mapping network that randomly samples a point from the latent space as input and generates a style vector which is transformed and incorporated into each block of the generator model, adaptive instance normalization, that involve first standardizing the output of feature map to a standard Gaussian, then adding the style vector as a bias term and progressive growing that helps grow both the generator and discriminator progressively. These functions were completely implemented from scratch and were adjusted with other functions from the author's code.
The training time for the first paper was based on training of StyleGANs that took nearly 1.5 days and training was done on Google Colab PRO, so the notebook rank continuously for 12 hours and got disrupted so we used to save the model state and delete the previous state to continue the training when the notebook shut itself.
For the second paper, it took nearly 3 days and it also faced the same disruption problems so we saved the models in as checkpoints and used the latest checkpoints as and when it got disconnected. This helped us in achieving the desirable results within our contraint resources.
Hardware we used was the GPU provided by Google Colab PRO.

3.1 Division of Labour

We are working in a team of three people and have planned to divide the work and responsibilities as follows:
Team Member 1 (Shivani): Responsible for understanding, reproducing, and implementing the major details of paper [2].
Team Member 2 (Gursahaj): Responsible for understanding, reproducing, and implementing the major details of paper [1] and extending the results of paper [2].
Team Member 3 (Mamta): Responsible for training of StyleGAN's and unit testing components of the individual papers [1] and [2] where required. Also responsible for major documentational activities.

We also plan to work together on several components of the project such as integration of the two papers and performing integration testing after the code modules have been combined.

3.2 Datasets

For [1], we used the Flickr Faces High Quality dataset that contains 70,000 high-quality PNG images at 1024 x 1024 resolution. For (2), we used CelebA-HQ dataset that contains images of celebrities. Both the datasets are publicly available. Both the datasets are large and are of high dimensions. We will check if the dataset is too heavy, then we thought of subsampling or even decrease the resolution to run the project.

3.3 Hyperparameters

Since the code was complex and the we wanted our process to be successful in least time possible, we decided to use the hypermparameters provided by the authors. This helped us achieve the results claimed by the authors and deliver the correct output within time.
For Paper[2], the hyperparameters values were batch size=8, beta1=0.0, beta2=0.99, ds iter=100000, eval every=50000, f lr=1e-06, hidden dim=512, img size=512, lambda cyc=1.0, lambda ds=1.0, lambda reg=1.0, lambda sty=1.0, latent dim=16, lr=0.0001, num domains=2, num outs per domain=10, num workers=4, print every=10, randcrop prob=0.5, resume iter=0, sample every=5000, save every=10000, seed=777, style dim=64, total iters=100000, val batchsize=32, w hpf=1.0, weight decay=0.0001

3.4 Experimental setup

In order to run the experiments, first we tried to run the code for Paper [1] on Google Colab, and with the limited RAM and GPU, we were able to get the outputs claimed by the paper, but it took time. But training the StyleGAN was not

feasible to be done on Google Colab GPU and RAM.
Also, in order to run the Paper[2] to get the output, the process was exhaustive and got terminated due to low RAM, GPU and virtual disk space. And training on such specifications was next to impossible. We searched for many free resources in order to run the code and train the network but failed to do so. So we had to purchase the Google Colab PRO version.
The Google Colab PRO offered us a Tesla P100-PCIE-16GB GPU, 25.02 GB RAM and 147.15 GB of disk storage that solved our problems. We were able to train the networks, get outputs, store the models, manage sessions by breaking and continuing them as the notebook stopped and hence we able to complete our project.

3.5 Computational requirements

To implement [1], a RAM of 12 GB and a Tesla T4 GPU of 16 GB is sufficient. On the other hand, to implement (2), we need at least 16 GB RAM and a Tesla V100 GPU to execute the process. For (1), the paper has not mentioned anything about the time taken for the processing, however, (2) mentions that on 100K iterations, accurate results were achieved when it was trained for 3 days.

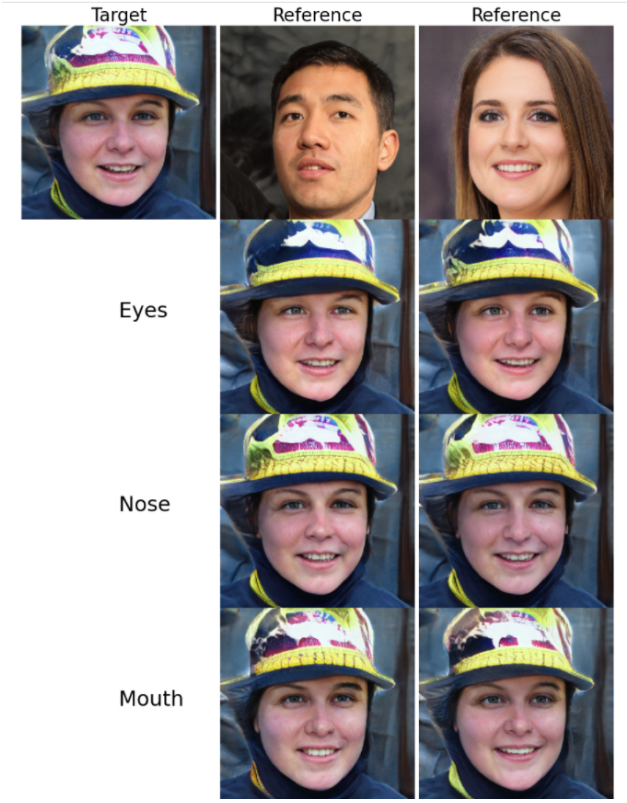
4 Results

We were able to achieve our reproducibility goals mentioned above by building several components of the GANs model from scratch on a lower resolution dataset containing 512 x 512 images. The results that we obtained supported the original claims made the papers but the quality of the reproduced results was lower as compared to the original high quality results due to the difference in the dataset image resolutions. Overall, we were successful in mapping the reference image features to the source (or target) image features on both papers and thus we successfully reproduced the transfer of multiple facial features, which was the core for both the papers.

4.1 Result 1

For paper [1], we took the reference and target images as:

Next, we reproduced the components of the StyleGANs model namely the Truncation method (that provided a trade-off between image quality or fidelity and image variety), the mapping network (that randomly samples a point from the latent space as input and generates a style vector which is transformed and incorporated into each block of the generator model), adaptive instance normalization (that involve first standardizing the output of the feature map to a standard Gaussian, then adding the style vector as a bias term), and progressive growing that helps grow both the generator and discriminator progressively. Thus, we obtain a reproduced StyleGAN which was trained on a low resolution dataset. Next, we implement the k-means clustering based on which we map the facial features of the reference images on to the target image. The results that we thus obtain by reproducing the work in paper [1] are:



In addition to the steps conducted the he original paper, we also trained a StyleGAN from scratch on the lower resolution dataset to better understand the training and working of StyleGANs. While the original paper took a pre-trained model on the high quality image dataset, we trained the reproduced StyleGAN on the celeb dataset consisting of 512 x 512 images where the weights and other outputs of the model were saved at regular intervals for stability. The training time taken for the model was about 1.5 days with Google Colab PRO which significantly reduced the training time due to higher GPU capabilities.

The results thus produced from this reproduced version of StyleGANs was efficient in performing facial feature transfer from a reference to a target image and thus supported the claims made by the authors of paper [1]. However, the quality of the results was not as good as the original results and a few images were distorted. However, given the compute and time limitations, we were still able to achieve very good results as per the dataset and model, which was about 85% of the original results obtained by the authors (approximately 3 in every 20 images were distorted for paper [1]).

4.2 Result 2

Similar to [1], we reproduce several components of StarGAN for paper [2] namely the mapping network, the noise injection, the Truncation method, progressive growing, and adaptive instance normalization (AdaIN) to apply the facial feature mappings between 3 pictures.

However, the work proposed in paper [2] focused only on the high quality datasets such as CelebA-HQ and AFHQ which was unfeasible for us to run due to the compute and time limitations (even on Google Colab PRO). Hence, we switched to the same lower resolution dataset as taken for reproducibility goals in [1], which contains of 512 x 512 size images and trained the model on this dataset. The training time taken was approximately 3 days on the dataset where the intermediate outputs had to be stored in a separate file to main stability.

The image results thus obtained by reproducing the StarGAN components in [2] can be seen below and the video thus obtained from the results can be found on this link: Click to view results for [2].

Furthermore, when compared with the output images in the original work in the paper, the resultant images that we obtained by reproducing the work in [2] had a few distortions in some images due to training on a lower resolution dataset. Approximate as compared to the images in [2], we obtained about 3 distorted images in a set of 10 images, thus giving us an approximate accuracy of 70%. However, with the given time and computational limitations, the model was able to perform very well with respect to the facial feature mappings given the low quality of the dataset. We further extended the facial mapping function on a GIF and a video which is explained in the Stretch Goal below.

5 Stretch Goals

We aimed at achieving the following stretch goals for [1] and [2]:

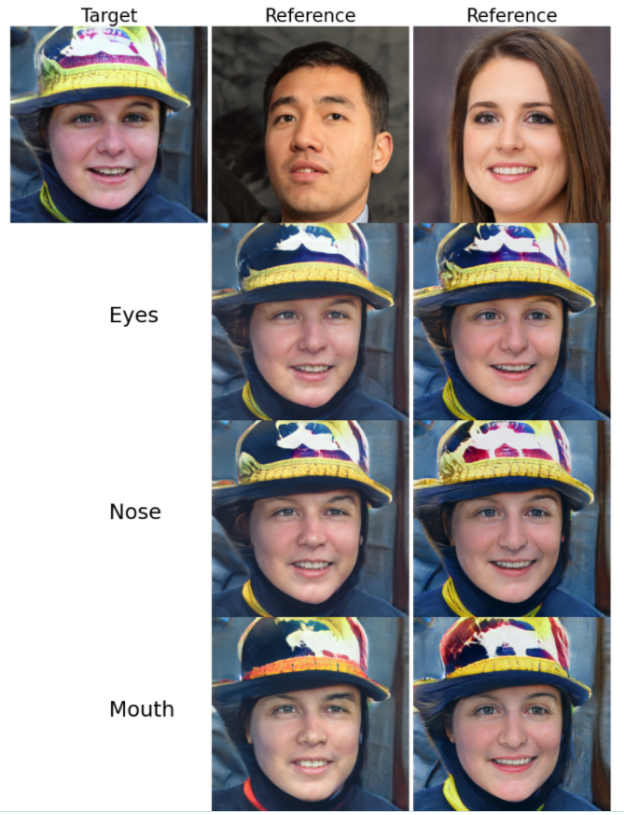
- Reproduce the project of Collins editing of facial features, which only aimed at a specific feature transfer at a time, given a target and reference images, and extend [1] by implementing multiple feature transfer at once. This change would enhance the work implemented by [1] since the extension would be able to transfer more than one features at a time, which the Collins editing method failed to do.
- Reproduce the multiple feature transfer in [2] and extend it by implementing it on a GIF, which can further be converted to a video. While converting the resultant images obtained through multiple feature transfer into a GIF, several things were adjusted and kept in mind such as the synchronization of the GIF, perfect face adjust in the GIF, low time in producing the result, matching the facial expression and ensuring that the quality does not downgrade, image cropping, etc.

We were successful at achieving the stretch goal for [1] wherein we extended the work implemented in [1] through multiple feature transfer. We achieved this by first clustering the features of the target image and the reference image using k-means clustering. Then, for each cluster indicating each feature, we mapped each reference feature on the target feature and assigned it as the starting point for next feature transfer. Thus, for each feature the reference feature was mapped on the target feature, which was then assigned as the original image. For the next feature transfer, we take this image obtained by the previous feature transfer and implement the next reference feature on the target feature. Note that here, the k-means clustering is done only once to cluster all the features after which each reference feature is implemented on the target feature in a loop, where each feature transfer takes the input image as the resultant image from the previous feature transfer.

This can be explained further with respect to the results that we obtained by extending [1]. The results obtained by reproducing the original work in [1], given 2 reference images and a target image can be seen under the 'Result 1' section given above.

As we see in the results, each reference feature such as the eyes, nose, and mouth are implemented one at a time on the target feature. Thus, in the first iteration, for each reference image, only the reference eyes are implemented on the target image. In the next iteration, only the reference nose is implemented on the target image, and in the final iteration, only the reference mouth is implemented on the target image.

However, when we extend this work to a transferring multiple features at a time, we get the result as:



Here, as we see in the first iteration, the reference eyes have been implemented on the target image. In the second iteration, both the reference eyes and nose have been implemented on the target image, and in the final iteration, all three reference features, namely the eyes, nose, and mouth have been implemented on the target image. Thus, the resultant image in the final iteration has all 3 reference features implemented on the target image.

We were also successful in achieving the stretch goal for [2] wherein we simulated the facial expressions taken from a source video (a clip of Tony Stark from Iron Man) and mapped the features from the source video to our target image. Thus, we were able to achieve a resultant video where our target image simulated the facial expressions of Tony Stark (source video) on top of which we added the background audio to give a complete experience of the clip from the movie.

We achieved this stretch goal for [2] by first converting the source video to a GIF format. Next, each frame from the GIF was taken as a single image on top of which we applied the starGAN feature mapping from each image of the source video to the target image.

This can be shown by:

Thus, we obtained a set of source images and target images where each target image had features mapped form the source image. These sets of target images were then extracted and compiled in a batch of 10 images per batch which was then converted to a GIF. Further, we converted this GIF to a video to obtain a target video where the target image followed the same facial expressions as the source video. While converting the target images to a GIF and video, several factors were taken into consideration such as adjusting the speed of the frames, the resultant image size as compared to the source image size, etc. where processing such as cropping was implemented to make the video look better. Finally, a the background audio taken from the source video was applied in a clip where both the source and target videos are simultaneously played to observe the facial expressions of the target video. This video can be found on the link: Click to view stretch goal for [2].

6 Discussion

The results that we obtained by reproducing the code in both papers was consistent with the claims made in papers [1] and [2]. We were able to obtain similar results for paper [1], and additionally, were also successful in extending the implementation of the reproduced code. In addition to the implementations given in [1], we also trained the styleGAN model with a lower resolution dataset consisting of 512×512 images, whereas the paper took a pretraind model of high resolution images. Thus, we were not only able to understand the working of styleGANs better, but we also learnt on how to train the model with a different dataset.

For paper [2], the given implementations were for 1024×1024 images. However, due to computational and time limitations, we ran the experiments on a lower resolution dataset consisting of 512×512 images. Thus, we were able to obtain similar results as compared to the results given by paper [2], but the quality of the resultant images was not as good as the original results due to the resolution difference in the dataset. We were also successful in extending the produced results to a GIF and a video consisting of a source and a target image.

6.1 What was easy

For paper [1], the explanation of the work proposed in the paper was easy to follow. The content was well structured and each component was well explained according to the task it performed in the model. For example, k-means clustering, which was an integral part of the model for feature clustering, was very well explained in the paper. Hence, even though the code implementation had a few inconsistencies, we were able to program the elements according to their function based on our understand. Similarly, other components for styleGANs as used in the paper was also explained thoroughly which helped us understand the model structure and reproduce the components of GANs, such as the truncation method, the mapping layer, adaptive instance normalization (AdaIN), and progressive growing.

For paper [2], although the code was a little complex, it was very well structured and the explanation was done in such a manner that it was easy for us to understand the workflow of the code. Hence, it was easy for us to identify the components of the starGAN v2 model and the entire flow of the model. Additionally, we also researched regarding the theoretical concepts of multi-domain tranfer which further aided in understanding of the model as a whole. Thus, the code structure along with individual research supplemented our code reproduction task. Further, since the original code was implemented on a dataset consisting of 1024×1024 images, through understanding each component along with the flow of the model, we were able to implement the model for another dataset consisting of 512 x 512 size images (due to computational and time constraints).

6.2 What was difficult

For paper [1], the code was not structured properly and had some errors while running them. While training for 512 $\times$ 512 images, we faced errors on training the forward passes of the network due to a lower resolution dataset and hence had to modify the forward pass in order to run the code. Additionally, training StyleGAN was difficult since the notebook was becoming unstable when we tried training it, even on a lower dimension human faces dataset. Hence, initially we had to keep a record of saving the notebook again and again in shorter intervals and the generated states took a lot of time due to which the training was happening very poorly. This was solved when we purchased Google Colab PRO as the GPU and High RAM provided faster and better executions with better run-time space to save the models efficiently without letting the notebooks to shut down unexpectedly.

For paper [2], although the code was structured very efficiently, we faced difficulties with the computational resource required to execute the paper without any extension. Initially, it was unfeasible to execute the basic functionalities of the code to generate the required output to check if the authors code is running as claimed on our local systems (and on Google Colab), let alone training. Hence we had to purchase the PRO version of Google Colab, which helped us a lot in running the output. Although the output was efficiently generated in some time, the training was still very extensive. It took 3 days and in order to generate the required outputs, we had to save the states and begin the training again after the notebook was connected, which was very difficult as the size of generated models was very high and that was difficult to save on drive due to limited drive storage.

Thus, finding the problem, searching for alternate resources, collecting lower dimension dataset, debugging the code, limiting the training time, managing the sessions of the notebooks, managing the size of data, and managing the model states were some of the major challenges we faced in order to reproduce the work in the project. Moreover, generating the video from StarGAN was also difficult since dividing the videos to frame and using 10 images at a time for facial generation was quite complex and time consuming.

6.3 Communication with original authors

We did not find any such major inconsistencies in both the papers due to which we did not find the need to contact the authors. We referred to Wikipedia and YouTube videos to understand the components of the paper that we did not understand, such as the latent code injection, style code, and regularizations (tuning and diversity regularization) used in the paper. Additionally, we also programmed the components of the code as per our understanding that we found were difficult to deal with in the original code. These modifications were also consistent with the results obtained in the paper, and the results that we obtained and thus, we did not feel the need to contact the authors regarding the same. We plan on sending the full report to the authors for their feedback and comments.

References

[1] Collins, E., Bala, R., Price, B., & Susstrunk, S. (2020). Editing in style: Uncovering the local semantics of gans. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 5771-5780).

[2] Choi, Y., Uh, Y., Yoo, J., & Ha, J. W. (2020). Stargan v2: Diverse image synthesis for multiple domains. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 8188-8197).

[3] .Slossberg, R., Shamai, G., & Kimmel, R. (2018). High quality facial surface and texture synthesis via generative adversarial networks. In Proceedings of the European Conference on Computer Vision (ECCV) Workshops (pp. 0-0).

[4] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba.Generating videos with scene dynamics. In Advances in Neural Information Processing Systems 29. 2016.

[5] Vondrick, C., Pirsiavash, H., & Torralba, A. (2016). Generating videos with scene dynamics. arXiv preprint arXiv:1609.02612.

[6] Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1125-1134).

[7] 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).

[8] Alharbi, Y., Smith, N., & Wonka, P. (2019). Latent filter scaling for multimodal unsupervised image-to-image translation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 1458-1466).

[9] Li, J., Liang, X., Wei, Y., Xu, T., Feng, J., & Yan, S. (2017). Perceptual generative adversarial networks for small object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1222-1230).

[10] Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.

[11] Karras, T., Laine, S., & Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 4401-4410).

[12] Wenqi Xian, Patsorn Sangkloy, Varun Agrawal, Amit Raj, Jingwan Lu, Chen Fang, Fisher Yu, and James Hays. Texturegan: Controlling deep image synthesis with texture patches. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2018.

[13] Xian, W., Sangkloy, P., Agrawal, V., Raj, A., Lu, J., Fang, C & Hays, J. (2018). Texturegan: Controlling deep image synthesis with texture patches. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 8456-8465).