An investigation of adversarial perturbations toward image generation networks

Eric Cao
University of Illinois at Urbana-Champaign
ejcao2@illinois.edu

Hunter A. Dyer
University of Illinois at Urbana-Champaign
hadyer2@illinois.edu

***Abstract*—In this paper, we performed an investigation into how adversarial noise can be used against image generation and manipulation processes that rely on machine learning for operation. Namely, we investigated the use of AdvGAN[10] to see the how it fared in creating adversarial perturbations for DeepFake networks like FaceSwap, and image translation models like Pix2Pix. Our motivation rests within wanting to provide integrity to images so that the images could not be used in the use of DeepFake generation, or otherwise manipulated with these networks. The goal for the noise is to enable any viewers of the generated DeepFake video to notice temporal and spatial inconsistencies and distortions. We detail our exploration of different noise parameters and generation against these networks, while comparing to random noise as well. We found that the magnitude of needed noise in order to produce noticeable effects in the output was higher than we had predicted, and that further work would need to be done to find the best way to produce distortions that corresponds with low magnitude noise.**

I. Introduction

Many methods for generating adversarial noise have been shown recently [2] [5] [10], and have been applied to many different areas. When noise generation methodologies have targeted classifiers, they aim to misclassify the target images. Other times the noise can be added in order to affect perceived image composition, as was demonstrated in the Houdini attack from Cisse et. al, which successfully attacked human-pose estimation, semantic segmentation, and speech recognition systems. Few works [5] have focused on utilizing adversarial examples for promoting image integrity against DeepFakes, as this may be more of a niche topic and is not useful from a adversary's view.

DeepFakes have become renowned within recent years, particularly due to their effectiveness at creating convincing images and video forgeries that superimpose a target's (person A) face onto arbitrary videos that contain another single person's face (person B). DeepFakes can be utilized for many different purposes, depending on the adversary's intentions and desires. There are many tools that can be utilized to construct DeepFakes, many of which are openly available on GitHub. The easy access to such tools can raise ethical concerns about their use and widespread availability. This concern was our motivating factor to investigate if adversarial noise could be utilized to ensure the integrity of such images.

Our initial goal in this work was to find some adversarial noise that could be applied to an image, such that if it was used within a DeepFake and a face was superimposed onto our protected image, there would be some noticeable distortion within the final photo or video. Like previous adversarial noise, this noise should be imperceptible so as not to disturb the end consumer of the untainted media. Since the goal is to make it apparent to the viewer of the image or video that it had been tampered with, almost any distortions or alterations work. Due to this flexible goal, we selected AdvGAN as the primary focus of our exploration. We selected it since we wanted to see what results it could produce in a non-classifier setting, while it also seemed like there might be a feasible opportunity it would work in a black box setting.

Unfortunately, as we discuss later, we were not able to attain results that met our exact goals set forth in the previous paragraph. We did find that noise could be generated in order to cause such distortions and noticeable defects within an image, however it was at the cost of noticeably altering the original photo. We conducted other investigations on other similar models to those used in DeepFakes in order to understand why we were not finding success in our original pursuit, and we detail our findings later in this paper. We then reflect on how we would continue working to solve our original problem in future work.

We first investigate random and adversarial noise applied to FaceSwap, one of the more popular DeepFake tools that was openly available on GitHub. The results from this guided us in exploring the application of adversarial noise to Pix2Pix, which is an image translation model. We chose Pix2Pix due to the similarity in transforms that the images go through, in which some input image is converted to another in a different output space. We investigated the noise on the Facades Pix2Pix dataset, as well as the Horse2Zebra dataset for CycleGAN. We then conclude by presenting the results that we did find on two different DeepFake models, along with discussion on how our other findings impact our initial problem goals, and how we would continue to solve this problem if time were not a constraint.

The contribution made by this paper is an understanding of how a previously unexplored (in application to DeepFakes) adversarial perturbation generation method, AdvGAN [10], interacts with DeepFake systems. Such an exploration is important to understand how adversarial perturbations might be used in a user-centric manner, and to understand if better alternatives exist, compared to results found in previous works, which are discussed in the next section.

II. Related Work

There have been a few other works that have worked with the same goal in mind as we have presented. They approached the problems in slightly different ways when it comes to generating the adversarial noise. We present a summary of the relevant parts of each work below.

Li et. al[5] In Li et. al., they specifically use adversarial perturbations to target the DNN face detectors used in DeepFakes. The face detectors are targeted specifically since the image synthesis part of DeepFake relies on the face being identified correctly. This is solved through a gradient-based optimization formulation, and they present white, black, and gray box attacks on these detectors.

Ruiz et. a[8] In Ruiz et. al.,they attack general image synthesis networks, like networks that change a target's facial expression or hair color. They utilize gradient approaches in order to generate the adversarial perturbations, fast gradient sign method, iterative fast gradient sign method, and projected gradient descent. They then use their findings to make training for DeepFake GANs more robust by training the discriminator on adversarial examples. From their paper, they claim to be the first to look at using adversarial examples against DeepFakes.

Yang et. al[11] In Yang et. al., they apply the adversarial perturbations to images prior to the training of a DeepFake model. They utilized projected gradient descent, an ensemble model, and random differentiable image transformations. Their results show that the output DeepFake images are of noticeable lesser quality than networks trained on unperturbed images.

Each of these papers have successfully applied imperceptible noise to images to attack different aspects of the DeepFake process. Our approach varies from these in that we were specifically interested in applying AdvGAN as a method for generating the adversarial perturbations. Our attempts also tried to attack both the face detection and synthesis, which we discuss the results of in a later section

III. Methodology and Results

While it is unconventional, we combine our Methodology and Results section as a result of our findings. Since we were challenged to find successful results for our original goal of FaceSwap, we verified our thoughts and intuitions on other networks in a hope to learn some information that might have aided us in getting unstuck. From this, we conducted different analysis and methods on each of our different models and scenarios. As such, we felt it would aid the presentation of our findings by combining the two sections. The sections are presented in the rough order that we had initially investigated them.

In this section we discuss the findings of our investigation into utilizing adversarial noise to affect image synthesis networks. We walk through our investigation into two models, Pix2Pix and CycleGAN, and two different datasets. We also present our preliminary findings of using targeted adversarial noise against two implementations of DeepFake models, and discuss how our findings of applying noise to Pix2Pix and CycleGAN affect our findings on DeepFake models.

A. AdvGAN

Many existing applications of adversarial examples specifically target classification tasks and have shown impressive results in creating examples that look visually similar, but fool classifiers. In adapting this problem to a regression setting, we set out with three goals in mind: the protected images should be indistinguishable from the original image, the results of DeepFakes or some other transformation on the image should generate noticeable artifacts on the generated image, and the process for generating the images should be efficient so that it can be efficiently performed on videos. We chose to use AdvGAN in order to generate the adversarial perturbation. AdvGAN is a slight modification on the original GAN architecture. AdvGAN takes in an input image and generates perturbation from it. To ensure that perturbed images look realistic, AdvGAN utilizes the GAN adversarial loss [7]:

$$\mathcal{L}_{GAN} = \mathbb{E}_x \log \mathcal{D}(x) + \mathbb{E}_x \log\left(1 - \mathcal{D}\left(x + \mathcal{G}(x)\right)\right). \quad (1)$$

And to optimize the generator $\mathcal{G}$ for generating adversarial perturbations to hinder classification, it adds an adversarial loss specialized for classification

$$\mathcal{L}_{adv}^{f} = \mathbb{E}_x \ell_f\left(\left(x + \mathcal{G}(x)\right), t\right), \quad (2)$$

that minimizes the loss function ℓ_f of the target classifier f. The last loss term used is a hinge loss [1],

$$\mathcal{L}_{hinge} = \mathbb{E}_x \max\left(0, \left\|\mathcal{G}(x)\right\|_2 - c\right), \quad (3)$$

that penalizes perturbations with a L_2 norm greater than a hyperparameter c.

In our implementation of AdvGAN, we reuse the GAN adversarial loss because we want the protected images to look realistic and we reuse the hinge loss because we want the protected image to be similar to the original image. Unlike the hinge loss and GAN adversarial loss, we must define our own L_{adv}^{f} since the original one is specialized towards classification. To do this, we defined a loss function to maximize the dissimilarity between the result of running a DeepFake tool on x and the result of running the DeepFake tool on our protected image image $x + \mathcal{G}(x)$. We call the resulting deep-fakes y and $\hat{y}$, respectively. It is as follows:

$$L_{adv} = -\left\|y - \hat{y}\right\|_2. \quad (4)$$

This function is maximized when y and $\hat{y}$ are identical and minimized when every pixel in $\hat{y}$ is as far as possible from the equivalent pixel in y. Our combined loss function is

$$\mathcal{L} = \mathcal{L}_{GAN} + \alpha\mathcal{L}_{adv} + \beta\mathcal{L}_{hinge}. \quad (5)$$

B. FaceSwap

For the evaluation of our methodology we used FaceSwap[9], which is one of the more popular DeepFake tools that is easily accessible and available. We chose it due to its availability, open source nature, and popularity. It was our

intent to find other models to test our methodology on if we found success, however after not finding success in the many repeated attempts that we describe here, we became fixated on attempting to understand why we weren't finding success. With the benefit of reflection and hindsight, we should have investigated on more than the two DeepFake methods that we did do (the other is the AutoEncoder implementation which is discussed later).

As previously mentioned, we utilized AdvGan[10] as the main focus for our investigation. It was utilized due to the ability of being able to generate noise in a white box and black box settings. In our implementation, we evaluate FaceSwap in a black box setting. This was a result of making a prior decision that we would be doing our implementation in PyTorch due to our familiarity with it. This had some unintended consequences down the line that we didn't consider when making our initial decisions. We implemented AdvGAN in PyTorch and then built infrastructure in place in order to allow us to call and utilize FaceSwap in the training.

FaceSwap has two main components in its operation, extraction and converting. The extraction utilizes pretrained models to extract the face bounding box from the larger resolution photos. The face extractions are resized to 256x256. Convert then requires that the user selects one of the different models/methodologies available for creating DeepFakes. For our purposes, we used DFaker[3], which is an implementation that used AutoEncoders, similar to the other DeepFake model we test later.

For testing, we used videos of two YouTubers, Veritasium and Phillip DeFranco. We chose them as they have many shots of just their face and they usually monologue for extended time. We felt that this would provide us with ample, high resolution images to pick and train from. It also provides us with clips that we can extract when they are talking. We had intended to extend our methodology to other datasets or videos once we had found some success, however we unfortunately did not make it that far. Each photo was 1080x1920 resolution.

Due to the high resolution, we focused on starting with a 256x256 placed in the top left of the face's bounding box. Our hope was that the noise produced in this box would attack the face extraction process, or that when the box overlapped with face features such as the eyes or nose, that the noise it produces might cause distortions in the superimposed face

Within our training, we generate the noise through AdvGAN, clipped it to a preset value, .1, (when the images were normalized between 0 and 1). We then added the generated noise to the image and made subprocess calls to FaceSwap. Within these subprocess calls we ran face extraction, which was required for swapping faces, and then we converted the faces using a model we trained. The model we trained could swap Veritasium's face with Phillip DeFranco, or in the reverse direction. We utilized the DeepFake process on the perturbed image, along with the precomputed DeepFake of the unperturbed image in order to calculate the loss for each training batch. It is notable that each batch took between 60-90 seconds each due to the extraction, along with added overhead for making the calls through Python's subprocess module.

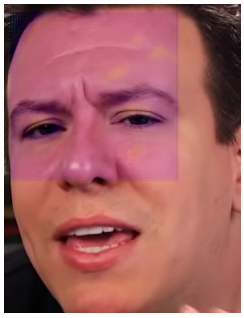

Fig. 1. Noise generated for FaceSwap.

Our main exploration with FaceSwap was through varying our loss function and tuning other hyperparameters. Our exploration was rather short-lived though, as FaceSwap was implemented in Keras and Tensorflow. Due to this we were unable to take advantage of PyTorch's automatic backpropagation. The consequence of not properly backpropagating our loss was that the noise that the model generated would always recede to less than $1e^{-5}$, which is imperceptible and was not enough to move out of the manifold. To rectify this issue, we found a PyTorch implementation of DFaker (that is used in FaceSwap) to test on in order to see how well AdvGAN performs.

We detail the exploration of our loss function more in the next section, as we achieve presentable results there. The results that we obtained from FaceSwap had imperceptible noise but no distortion in the output image, and no amount of tuning seemed to change the results significantly enough. The results we got either had noise that we very noticeable as can be seen in Figure 1, or receded as we described earlier. There seemed to be no middle ground.

The general trend we noticed regarding outputs is that the results from FaceSwap would just superimpose the face, and the noise would still be leftover, as can be seen in Figure 2. In this particular example, there is a discoloration in the eye, however it is unclear if this is a result of actual adversarial noise, or if it is a result of the FaceSwap mispredicting the lighting on this particular part of the face due to the presence and discoloration of the noise. While these may scenarios seem very similar, it is worth making the distinction, as if it were the latter, then the model wasn't actually learning how to make the noise adversarial in nature, and would likely default to just putting discoloration on this part of the face. From that point, it might not have any hope of ever becoming imperceptible.

C. AutoEncoder Implementation

In an attempt to find out if a white box scenario would improve our results, we sought to fix our backpropagation problem that was described in the FaceSwap section by finding a DeepFake tool that was implemented in PyTorch so that we could take advantage of PyTorch's automatic gradient and backpropagation. The data and implementation of AdvGAN remained the same from the FaceSwap section. Our main point

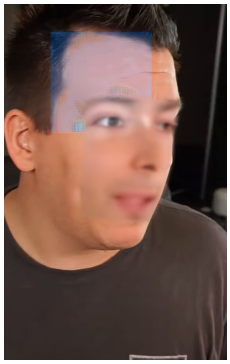

Fig. 2. The results of FaceSwap being applied to a perturbed image.

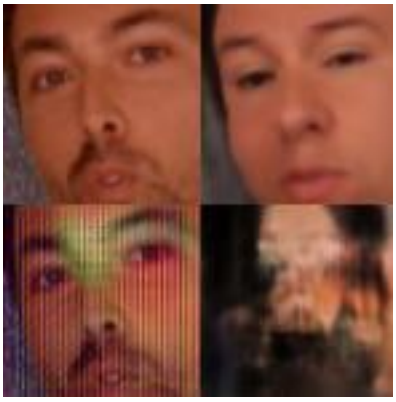

Fig. 3. AutoEncoder - Top Row: Unperturbed input (left) and output of the AutoEncoder DeepFake (right). Bottom Row: Perturbed input (left) and perturbed DeepFake (right).

of exploration in the time that we had to conduct it was of the loss function and tuning other hyperparameters.

We found this model [6] through searching GitHub with the search term 'pytorch deepfake', and we utilized the model that had the most stars and forks, as we felt that popularity would be a good indicator of the model being of reasonable quality and implementation. If it weren't for time constraints, the better alternative would have been to reimplement AdvGAN into TensorFlow and Keras so that it would have been easier to interface with FaceSwap.

This model requires that inputs be face extractions. It then resizes them to 64x64 and uses an AutoEncoder and two Decoders in order to serve as a means to swap faces that it is trained on. Each decoder is responsible for transforming faces in one direction, either face A to face B, or face B to face A. We trained our model on the face extractions of Veritasium and Phillip DeFranco. We pretrained this model for 100,000 iterations and it produces reasonable results. As with FaceSwap, our exploration of how noise affects the model lies primarily in the tuning of the model and the use of our loss function.

We utilized the same loss functions as with FaceSwap, but we got better results due to the loss actually being backpropagated. Most of our tuning was spent tuning the constants $\lambda_{adv} and \lambda_{pert}$ to correctly balance the incentive of the noise to produce distortions in the output and the noise being imperceptible.

Our results were not of the quality that we had hoped, but showed promised. In figure 4, we can see that the strength of the perturbation (in which it is multiplied by the constant at the top of the column) needs to be fairly high in order to see noticeable outputs in the perturbed DeepFake. In this model in particular, the noise seems to affect the parts of the face that is not the cheekbone or mouth.

Figure 3 shows the result of a poorly tuned model. Though the perturbed output is unrecognizable as a human face, the noise is very noticeable, even with a low strength factor. For our purposes, we require the noise to be close to imperceptible, as we wouldn't want to alter the viewing experience for the regular video in circumstance in which it was not run through the DeepFake.

Since the noise in this case needed a large magnitude to make notable distortions in the perturbed output image, we investigated our implementation against two different image generation models in order to ascertain if our issues were a result of the nature of DeepFakes and their implementation, or if it was a result of image generation in general.

D. Pix2Pix - Facades

Our application of AdvGAN and adversarial noise was intended at first only for FaceSwap. Due to results we found, we utilized Pix2Pix [4] as another method to explore how we can effectively control the noise. Particularly we were able to draw insights of how to control the magnitudes, norms, and perceptibility of the noise. We also investigated this model since it was similar to DeepFake models in the fact that its purpose is image translation.

Pix2Pix utilizes a conditional GAN to translate one image into the style of another. We tested AdvGAN on their facades dataset which takes an color mapping and produces a realistic looking image of facades and buildings. Our results on this were a bit more effective than what we were able to achieve with DeepFakes. We utilized the same loss functions and hyper-parameters as with FaceSwap, but the loss parameters $\lambda_{adv} and \lambda_{pert}$ varied slightly in weights, but were tuned in a similar fashion.

We had varied results, and the tuning of the networks was very finicky, as can be seen in Figure 5 and Figure 6. In Figure 5 the noise is very well hidden in the different features and we can see that there is noticeable distortion in the output after the protected images is transformed with Pix2Pix. In particular, as the strength of the perturbation increases, the more the window sills are erased and the images as a whole become more beige. It's worth noting in the case of the well tuned model that all that really changes between the input and perturbed input is that the hue of some of the facades changes, but it is not as noticeable unless you have both images side by side, where as the difference between the outputs is large, with minimal changes in the input. However, from the perspective

Fig. 4. AutoEncoder - A variety of results from our best model.

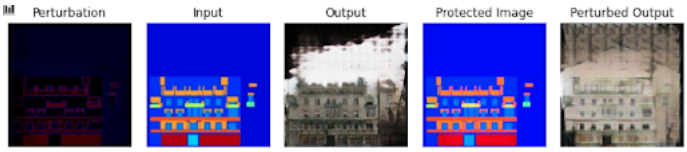

Fig. 5. Pix2Pix - Results from a well tuned model.

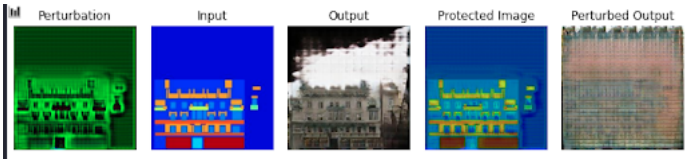

Fig. 6. Pix2Pix - Results from a poorly tuned model.

of a DeepFake image, these results would not suitable due to the color shift of large regions of the image.

We wanted to show the results of a poorly tuned models, which can be seen in Figure 5. The perturbation in this instance is notably different in that it presents a texture in the background, and this texture can be seen in the perturbed input and the perturbed output. The colors of hues and the lines in the background of the top of the perturbation are very apparent when applied to the image. This perturbation is very noticeable, largely due to the green tint and the texturing. Even though the perturbed output is nowhere near the unperturbed output, the large magnitude of the noise makes this result unacceptable within the contexts of DeepFakes.

Figure 7 showcases a variety of other results that we found, but when the noise is linearly scaled by a constant factor. For each number on each top column, the left image of the corresponding number is the input perturbed image, and the right image is the perturbed output. In each case, the perturbation is close to imperceptible, and there are noticeable distortions in the output perturbed image.

E. CycleGAN - Horse2Zebra

In a similar vein to Pix2Pix, we also tested out our methodology on CycleGAN, another image translation model that learns the mapping , trained on the Horse2Zebra data set. We focused on the direction from translating horses to zebras for our analysis. In this data set, images that depict horses will have their skin placed with a pattern that resembles a zebra's skin. We applied our network with the same loss functions as the previous methods, and like the previous sections we only adjusted the loss constants $\lambda_{perturb}$ and λ_{adv}.

Through tuning, we achieve similar results and investigate the effect of different weighting of the noise, as can be seen in Figure 8. In comparison to the results in Figure 7, the results from CycleGAN are more promising. The results of the perturbations are much more obvious, covers large portions of the image, darkens the image, and removes striping from the horse itself. Additionally, the results from CycleGAN are more promising since the distortions in the perturbed output can be seen even when the perturbation of the input image is multiplied by 0.2 of its original value. It is in this case that we actually get close to ideal results where the perturbation is almost imperceptible and there are noticeable distortions in the output image. The zebra pattern is shown in the background of the image rather than on the animal that it should be on. At a strength of 0.6 and 1.0, the perturbation becomes visible in the perturbed image, which is unacceptable for our purposes.

F. Random Noise

In order to ensure that our method had effect due to learned perturbations, and not just due to the presence of noise, we applied random noise in varying magnitudes to images in order to find at what point would the noise present a noticeable distortion in the output. We tested random noise on both CycleGAN and FaceSwap in order to ascertain the effect of noise in both domains, image translation and DeepFakes.

We explored the use of uniformly random noise on each of these processes. We first generated a low-magnitude vector of noise, and then similarly to how we scaled noise in previous

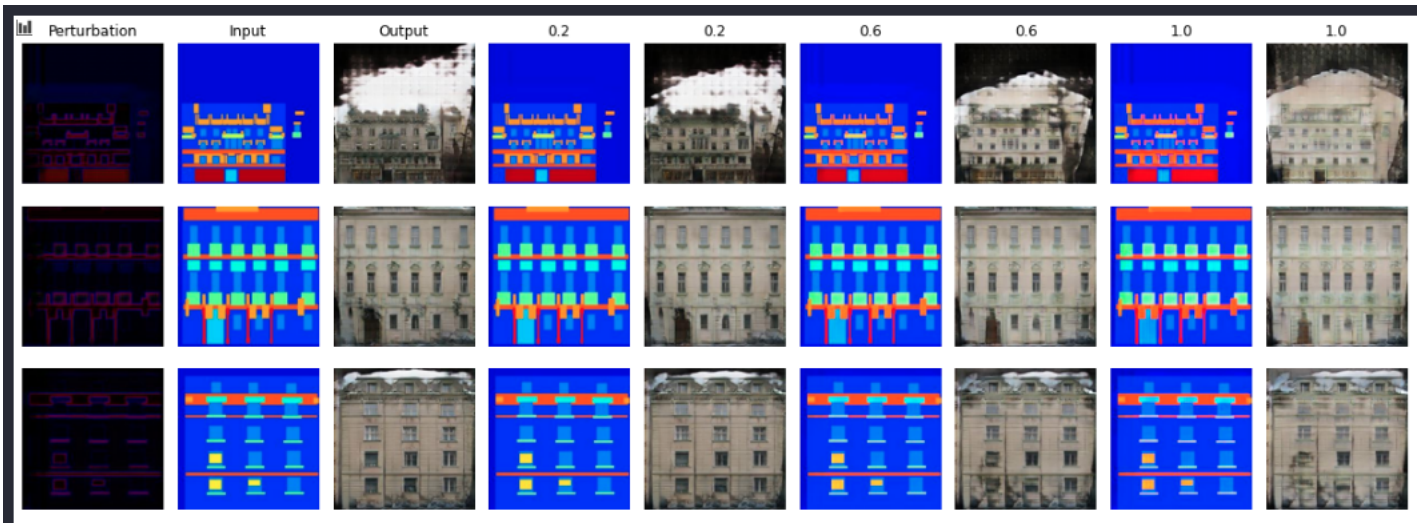

Fig. 7. A variety of results from our best Pix2Pix model.

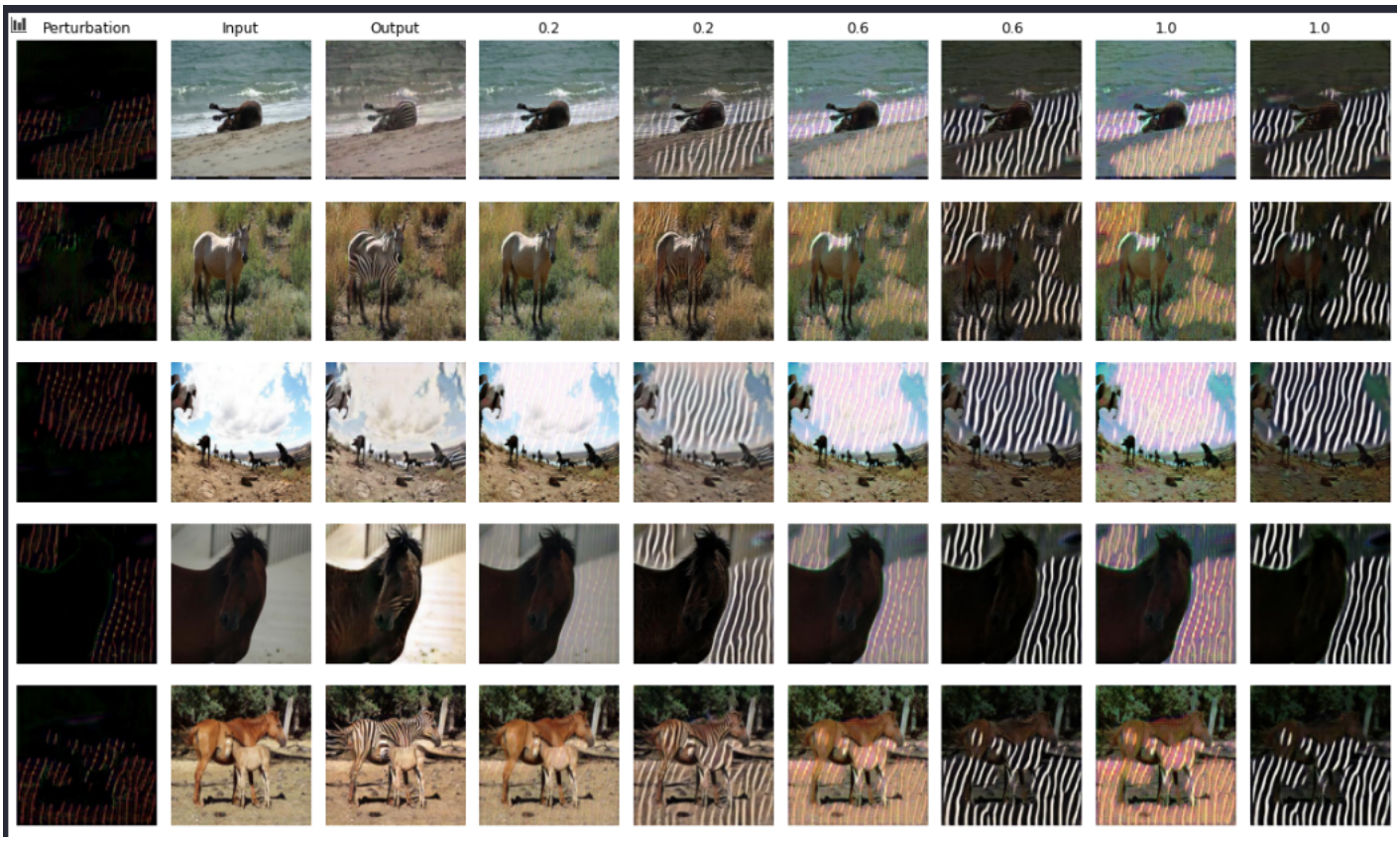

Fig. 8. A variety of results from our best CycleGAN.

sections (by multiplying the vector by some constant), we explored noise by doubling the constant multiplier every time to find the minimum multiplier that produced noticeable noise, at which point anything larger becomes ineffective for the goal of applying imperceptible noise. Additionally, for the horse2zebra data set, we compared it against noise with comparable magnitude to those shown in the rightmost column of Figure 8.

We compare the effect of the adversarial perturbation against that of random noise by looking at how they affect image similarity more concretely by creating histograms for x, y, $\hat{x}$, and $\hat{y}$, using random noise alongside adversarial perturbations, and finding the correlations between the input images and output images. Repeating this over all of the images in our test set, we get the plot in Figure 11. In the plot, each data point represents an image from the test set. As the goal of our model is to generate minimal perturbation while maximizing the effect on the output image, the ideal data point would exist in the bottom left corner of the plot. Each data point is colored, blue data points are the result of adversarial perturbation and red points are the result of uniform random noise. What we find is that adversarial perturbation tends to generate data points that drift towards the top to middle right. These perturbations are significant enough to degrade the input image correlation but have a much larger effect on the output correlation, which is the desired effect. By contrast, uniform random noise tends to be clustered in the top right, meaning that the input images are still highly correlated and tend to have little effect on the image. Thus we conclude that on average, adversarial perturbation have a larger effect on the output image, but is more noticeable than random noise.

We found that the amount of noise that is needed in order to actually affect any image generation process or the DeepFake process is very large. In Figure 9, we show the level of noise at a barely perceptible level, and the results it produces. Additionally, in Figure 10, we show the level of noise that is needed to noticeably show any distortions, which is quite significant. From these results, it is clear to see that the magnitude of random noise needed in order produce noticeable distortions far surpasses what is acceptable in terms of adding imperceptible noise. The magnitudes of noise added were .6 for Horse2Zebra and 512 for FaceSwap.

IV. Discussion

Our results were rather varied in terms of meeting our goals that were set forth at the beginning of the paper. The results that we got from AdvGAN adhered to the goals for Pix2Pix and CycleGAN, but fell a little short when applied to a DeepFake network. We wanted to take the time to discuss the limitations of our original goal and approach to it, as with these limitations, combined with the new knowledge gained, we would reformulate our problem for future investigations and approach it in a different manner. While our results for DeepFakes were close to our goals, we think our selection of noise generation constrained our results, and changing it might yield more promising results.

Fig. 9. Barely perceptible noise within CycleGAN's Horse2Zebra data set.

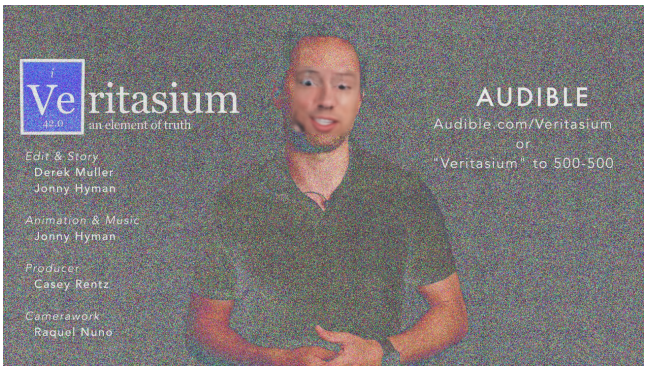

Fig. 10. Noise needed to noticeably distort the DeepFake process.

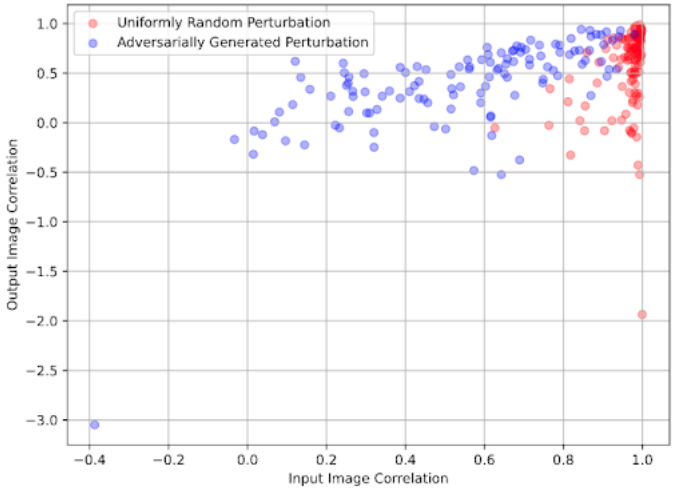

Fig. 11. Correlation between input and output images in the Horse2Zebra data set.

One limitation with our methodology is that perturbations are applied to to the face that is being replaced, and not to the face that is being superimposed. This doesn't provide any guarantees of integrity to the actual target of a DeepFake attack (the person who appears in the doctored footage). However we consider this to be outside the scope of this investigation as we wanted to prioritize the end-user (viewer) of the media, and ideally make the images unusable. Yang et. al. [11] has applied perturbations to the images prior to training the DeepFake model. It would have been computationally infeasible to explore this as well, as training a new DeepFake model on each iteration of training AdvGAN would be a large computational burden, both in terms of machine capabilities, as well as time.

In our project, we utilized FaceSwap and an AutoEncoder implementation from GitHub for producing DeepFakes. This brings about the limitation that since we were using AdvGAN for our noise generation process, the noise can potentially be overfit for the the model that it was being trained on. Given more time, we would have liked to investigated the potential transferability of this noise, as transferable noise within this domain would be very strong in terms of being able to all but guarantee the integrity of an image.

In the end, we feel that constricting ourselves to using AdvGAN hampered our progress. In future work we would hope to work with AdvGAN and investigate ways we would change the architecture and formulation in order to solve our problems, and avoid tunnel vision. We feel that some of our challenges in noise generation could have been solved through changes, as while we made some adaptations, it wasn't enough to fully solve the problem.

V. Conclusion

In this paper we explored the application of adversarial noise from AdvGAN [10] to DeepFake models in order to understand what results could be attained, and see if it would be sufficient enough for utilizing in an integrity-preserving manner. Our results show that the needed magnitude noise for human faces, as is generated through AdvGAN, appears to be notably higher compared to other image generation models and previous work. We established the ability for AdvGAN to successfully generate imperceptible noise that causes distortions in the output image with other image generation models, Pix2Pix and CycleGAN. From there, the results were compared to a baseline of random noise.

References

[1] Nicholas Carlini and David A. Wagner. "Towards Evaluating the Robustness of Neural Networks". In: *CoRR* abs/1608.04644 (2016). arXiv: 1608.04644. URL: http://arxiv.org/abs/1608.04644.

[2] Moustapha Cisse et al. *Houdini: Fooling Deep Structured Prediction Models*. 2017. arXiv: 1707.05373 [stat.ML].

[3] dfaker. *DFaker repository, https://github.com/dfaker/df*.

[4] Phillip Isola et al. "Image-to-Image Translation with Conditional Adversarial Networks". In: *CoRR* abs/1611.07004 (2016). arXiv: 1611.07004. URL: http://arxiv.org/abs/1611.07004.

[5] Yuezun Li et al. *Hiding Faces in Plain Sight: Disrupting AI Face Synthesis with Adversarial Perturbations*. 2019. arXiv: 1906.09288 [cs.CV].

[6] Oldpan. *Faceswap-Deepfake-Pytorch, https://github.com/Oldpan/Faceswap-Deepfake-Pytorch*.

[7] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015).

[8] Nataniel Ruiz, Sarah Adel Bargal, and Stan Sclaroff. *Disrupting Deepfakes: Adversarial Attacks Against Conditional Image Translation Networks and Facial Manipulation Systems*. 2020. arXiv: 2003.01279 [cs.CV].

[9] torzdf, andenixa, and kvrooman. *FaceSwap GitHub Repository, https://github.com/deepfakes/faceswap*.

[10] Chaowei Xiao et al. *Generating Adversarial Examples with Adversarial Networks*. 2019. arXiv: 1801.02610 [cs.CR].

[11] Chaofei Yang et al. *Defending against GAN-based Deepfake Attacks via Transformation-aware Adversarial Faces*. 2020. arXiv: 2006.07421 [cs.CV].

VI. Appendix

A. Details on Image Correlation

Before applying the perturbation, we clip the perturbation element-wise to be between -0.1 and 0.1 for comparison. The images in Figure 9 and Figure 8 already have the clipped perturbation applied and the resulting norms are 34.93 and 33.19 for the random noise and the adversarial perturbation, respectively. For a 256 by 256 image, this difference is negligible. Adding this clipped perturbation, prevents the perturbation from growing too large and generating protected images that are too far from the original image. A notable hyperparameter when creating the histograms is the number of bins. The lower the number of bins, the more colors get grouped together. Thus the less bins used, larger perturbation is necessary to change what bin a pixel is grouped to. For example, Figure 11 only uses 6 bins per color space, this has the effect of overestimating the correlation for the input images. If we increase the number of bins for 32 per channel, then we get the result from Figure 12. We see that while trends observed from before, even as the bin size varies. We can see the overestimation of correlation as the mean correlation shifts downwards.

Additionally, it should be noted that while correlation is often used for image similarity, it is not a perfect measure. As we can see from figures 8 and 9, the adversarial noise is more structured and the result on the output image is similarly structured. This quality is desirable as a defender since it

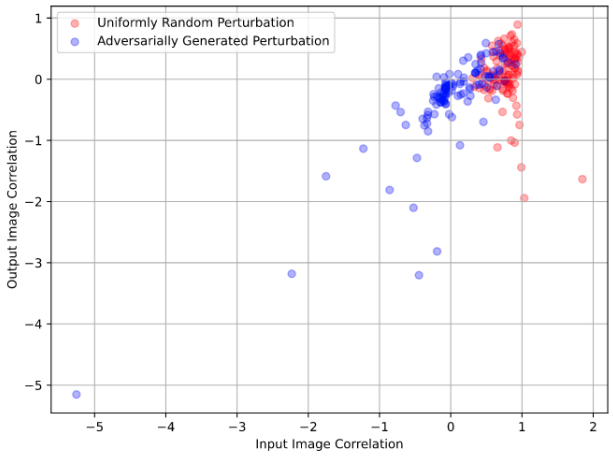

Fig. 12. Horse2Zebra Image correlation using 32 bins.

makes it clear that the image has been tampered rather than being compressed too many times.