Progress Report for Dr. Ray: Modelling Phishing-Website Characteristics using Generative Adversarial Networks

Vignesh Pagadala and Hossein Shirazi

July 28, 2020

1 Deep-Convolutional Generative Adversarial Network (DCGAN)

My initial idea was to use a DCGAN to generate the forged website images, as DCGANs modify GANs such that additional convolutional layers are added in the neural network. This would allow the GAN to capture characteristics such as inter-correlations within a dataset, and use that for generating the model. This meant that DCGANs were great for modelling image and video data. I used PyTorch and implemented it, initially running it in my own laptop. After a lot of recommendations from Hossein, I migrated the project to get it to run on the lab machines and their GPUs, which sped up training significantly.

The generated images captured certain features of the training data, however, the framework could only deal with very small images (64x64 pixels), and although it seemed like further training would make it better, the images were too small to effectively capture the featured we needed. Some examples of these images have been provided below (Adobe website).

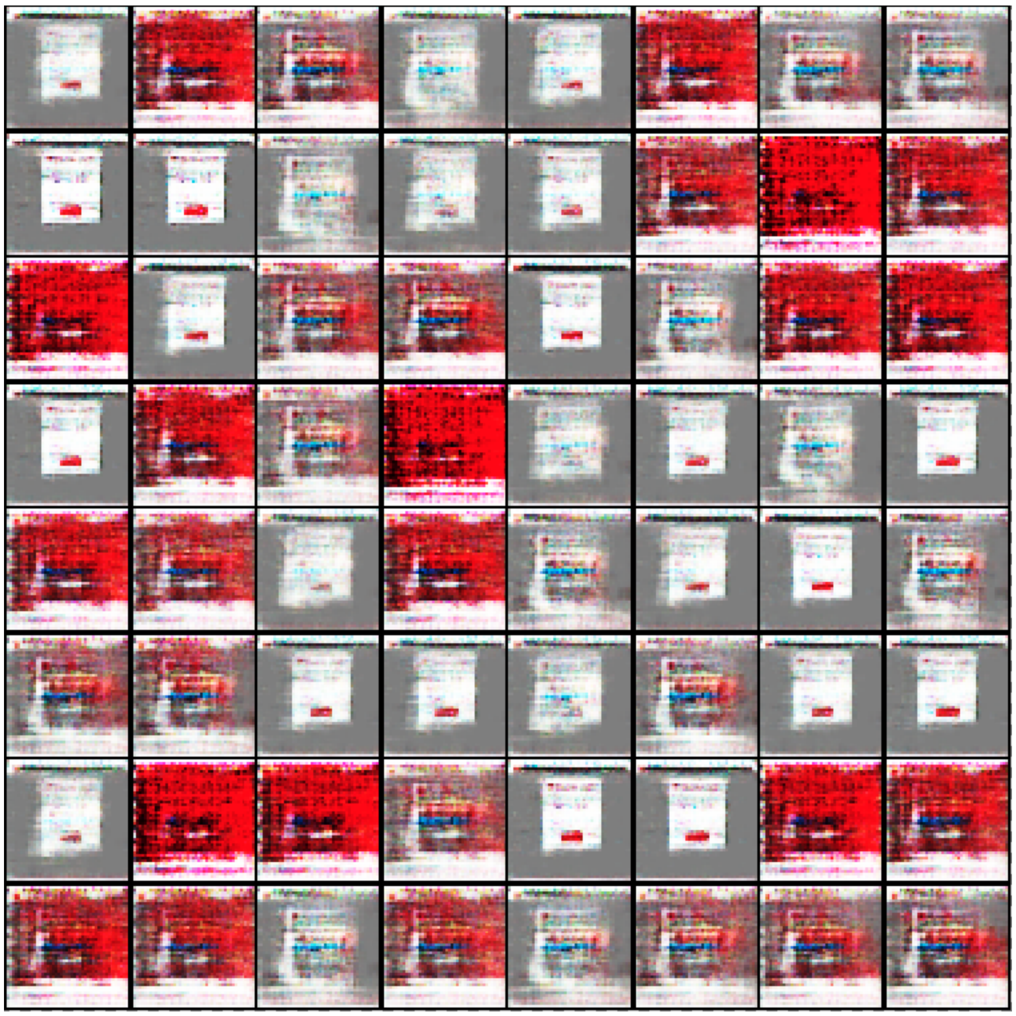


Figure 1: 64x64 DC-GAN Images

I implemented the necessary changes in the framework to accommodate larger images (128x128) images, and ran the training again. However, this did not lead to good results. The training was highly unstable, characterized by constant mode-collapse, and poor values of the loss functions. I came to understand that larger images are not suitable in the DCGAN framework which I was using. I tried various hyperparameter settings, and changes to the network itself, but this did not yield better results. I learned from various forums that implementing DCGANs for larger images was indeed very challenging, and involved a lot of training instability.

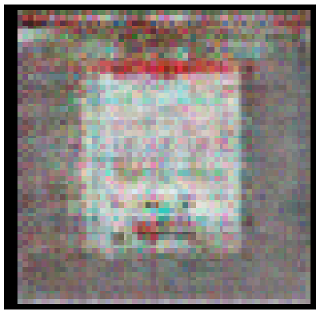
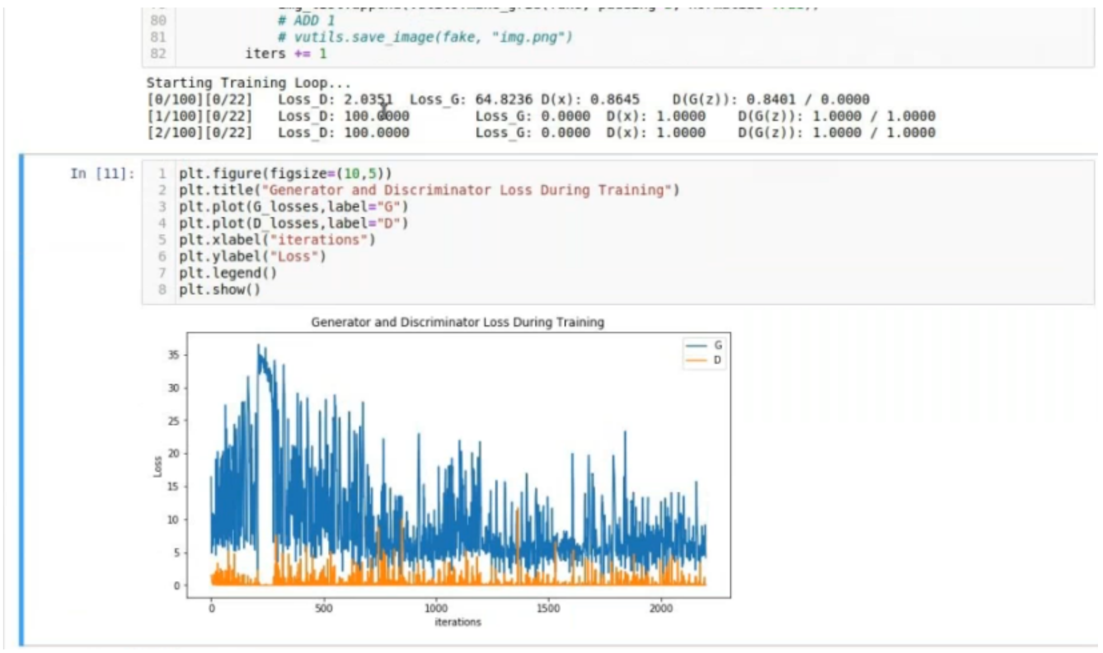


Figure 2: 64x64 DC-GAN Images

Figure 3: 64x64 DC-GAN Images



```
            # ADD 1
            # vutils.save_image(fake, "img.png")
        iters += 1
```

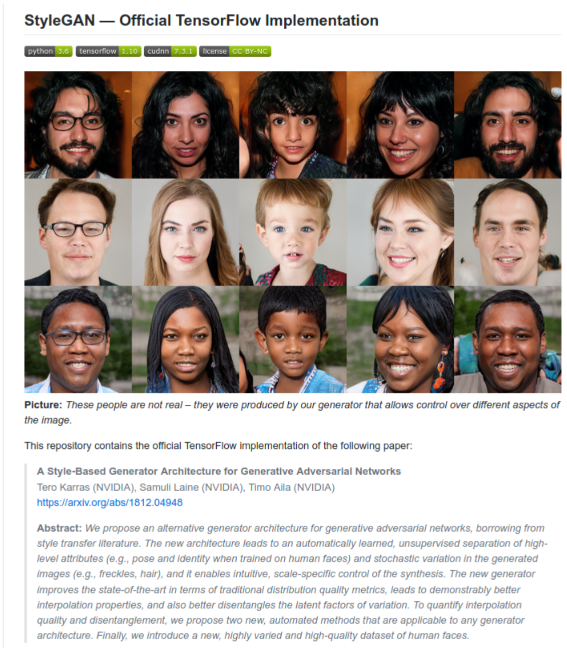
```
Starting Training Loop...
[0/100][0/22]   Loss_D: 2.0351  Loss_G: 64.8236 D(x): 0.8645    D(G(z)): 0.8401 / 0.0000
[1/100][0/22]   Loss_D: 100.0000        Loss_G: 0.0000  D(x): 1.0000    D(G(z)): 1.0000 / 1.0000
[2/100][0/22]   Loss_D: 100.0000        Loss_G: 0.0000  D(x): 1.0000    D(G(z)): 1.0000 / 1.0000
```

```
In [11]:
plt.figure(figsize=(10,5))
plt.title("Generator and Discriminator Loss During Training")
plt.plot(G_losses,label="G")
plt.plot(D_losses,label="D")
plt.xlabel("iterations")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

Figure 4: Mode collapse and instability were common problems faced.

2 Nvidia StyleGAN

I did further literature survey to find a more suitable framework for this use-case. I came across a novel GAN implementation made open-source by Nvidia. It was a TensorFlow implementation called 'StyleGAN' which seemed to show very good results. It dealt really well with high definition images. Their code was open-source, so I immediately forked the repository and set about modifying the code to accommodate my custom dataset. Link to the StyleGAN repository: https://github.com/NVlabs/stylegan.



StyleGAN — Official TensorFlow Implementation

python 3.6 | tensorflow 1.10 | cudnn 7.3.1 | license CC BY-NC

Picture: *These people are not real – they were produced by our generator that allows control over different aspects of the image.*

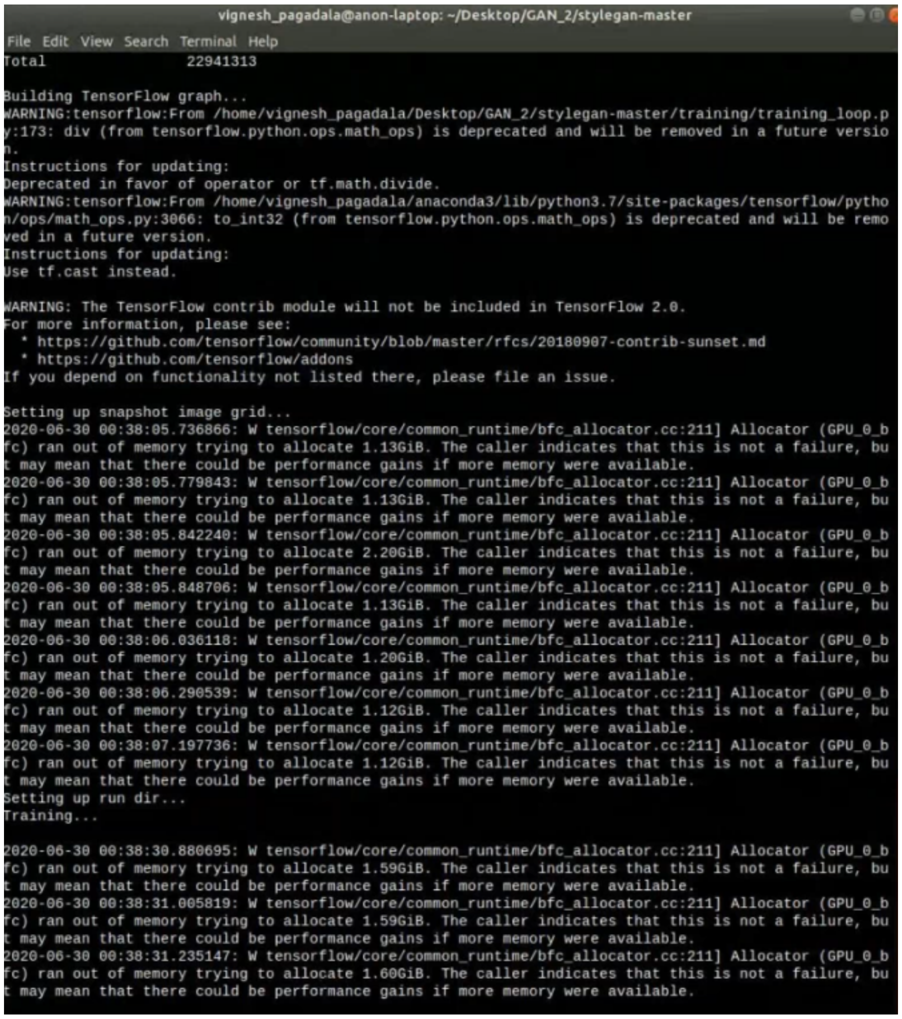
This repository contains the official TensorFlow implementation of the following paper:

> **A Style-Based Generator Architecture for Generative Adversarial Networks**
> Tero Karras (NVIDIA), Samuli Laine (NVIDIA), Timo Aila (NVIDIA)
> https://arxiv.org/abs/1812.04948
>
> **Abstract:** *We propose an alternative generator architecture for generative adversarial networks, borrowing from style transfer literature. The new architecture leads to an automatically learned, unsupervised separation of high-level attributes (e.g., pose and identity when trained on human faces) and stochastic variation in the generated images (e.g., freckles, hair), and it enables intuitive, scale-specific control of the synthesis. The new generator improves the state-of-the-art in terms of traditional distribution quality metrics, leads to demonstrably better interpolation properties, and also better disentangles the latent factors of variation. To quantify interpolation quality and disentanglement, we propose two new, automated methods that are applicable to any generator architecture. Finally, we introduce a new, highly varied and high-quality dataset of human faces.*

Figure 5: StyleGAN GitHub

I was able to successfully set up the experiment, and was able to run the code with my custom websites dataset. However, the implementation was very demanding in-terms of GPU power required. Even though I ran it in a laptop with an Nvidia GTX 1050ti GPU, I still kept getting warnings regarding memory overflows.

Based upon Hossein's recommendations, I tried to set it up using the lab-machines, because it was getting apparent that the training would take too much time when run with only one or even two GPUs. On the Nvidia StyleGAN GitHub, it is

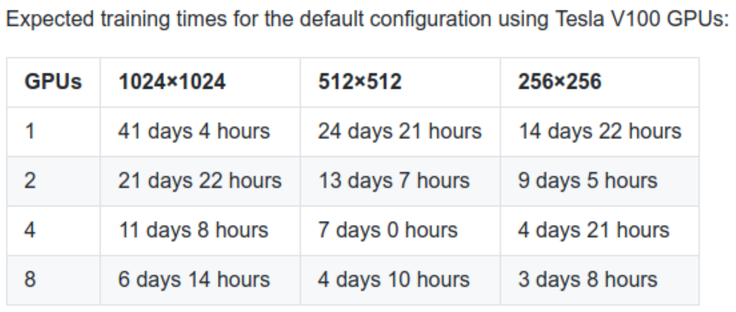


vignesh_pagadala@anon-laptop: ~/Desktop/GAN_2/stylegan-master
File Edit View Search Terminal Help

```
Total                   22941313

Building TensorFlow graph...
WARNING:tensorflow:From /home/vignesh_pagadala/Desktop/GAN_2/stylegan-master/training/training_loop.p
y:173: div (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future versio
n.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
WARNING:tensorflow:From /home/vignesh_pagadala/anaconda3/lib/python3.7/site-packages/tensorflow/pytho
n/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be remo
ved in a future version.
Instructions for updating:
Use tf.cast instead.

WARNING: The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-sunset.md
  * https://github.com/tensorflow/addons
If you depend on functionality not listed there, please file an issue.

Setting up snapshot image grid...
2020-06-30 00:38:05.736866: W tensorflow/core/common_runtime/bfc_allocator.cc:211] Allocator (GPU_0_b
fc) ran out of memory trying to allocate 1.13GiB. The caller indicates that this is not a failure, bu
t may mean that there could be performance gains if more memory were available.
2020-06-30 00:38:05.779843: W tensorflow/core/common_runtime/bfc_allocator.cc:211] Allocator (GPU_0_b
fc) ran out of memory trying to allocate 1.13GiB. The caller indicates that this is not a failure, bu
t may mean that there could be performance gains if more memory were available.
2020-06-30 00:38:05.842240: W tensorflow/core/common_runtime/bfc_allocator.cc:211] Allocator (GPU_0_b
fc) ran out of memory trying to allocate 2.20GiB. The caller indicates that this is not a failure, bu
t may mean that there could be performance gains if more memory were available.
2020-06-30 00:38:05.848706: W tensorflow/core/common_runtime/bfc_allocator.cc:211] Allocator (GPU_0_b
fc) ran out of memory trying to allocate 1.13GiB. The caller indicates that this is not a failure, bu
t may mean that there could be performance gains if more memory were available.
2020-06-30 00:38:06.036118: W tensorflow/core/common_runtime/bfc_allocator.cc:211] Allocator (GPU_0_b
fc) ran out of memory trying to allocate 1.20GiB. The caller indicates that this is not a failure, bu
t may mean that there could be performance gains if more memory were available.
2020-06-30 00:38:06.290539: W tensorflow/core/common_runtime/bfc_allocator.cc:211] Allocator (GPU_0_b
fc) ran out of memory trying to allocate 1.12GiB. The caller indicates that this is not a failure, bu
t may mean that there could be performance gains if more memory were available.
2020-06-30 00:38:07.197736: W tensorflow/core/common_runtime/bfc_allocator.cc:211] Allocator (GPU_0_b
fc) ran out of memory trying to allocate 1.12GiB. The caller indicates that this is not a failure, bu
t may mean that there could be performance gains if more memory were available.
Setting up run dir...
Training...

2020-06-30 00:38:30.880695: W tensorflow/core/common_runtime/bfc_allocator.cc:211] Allocator (GPU_0_b
fc) ran out of memory trying to allocate 1.59GiB. The caller indicates that this is not a failure, bu
t may mean that there could be performance gains if more memory were available.
2020-06-30 00:38:31.005819: W tensorflow/core/common_runtime/bfc_allocator.cc:211] Allocator (GPU_0_b
fc) ran out of memory trying to allocate 1.59GiB. The caller indicates that this is not a failure, bu
t may mean that there could be performance gains if more memory were available.
2020-06-30 00:38:31.235147: W tensorflow/core/common_runtime/bfc_allocator.cc:211] Allocator (GPU_0_b
fc) ran out of memory trying to allocate 1.60GiB. The caller indicates that this is not a failure, bu
t may mean that there could be performance gains if more memory were available.
```

Figure 6: StyleGAN is very GPU-intensive

mentioned that they ran the training for 48 days, when using one TeslaV100 GPU.



Expected training times for the default configuration using Tesla V100 GPUs:

GPUs	1024×1024	512×512	256×256
1	41 days 4 hours	24 days 21 hours	14 days 22 hours
2	21 days 22 hours	13 days 7 hours	9 days 5 hours
4	11 days 8 hours	7 days 0 hours	4 days 21 hours
8	6 days 14 hours	4 days 10 hours	3 days 8 hours

Figure 7: StyleGAN is very GPU-intensive

The conclusion here was that, this process had to be distributed using a GPU cluster, inevitably. I set about trying to parallelize the training over some of the racecar and fish machines (which have good GPUs). But this process had turned out to be more difficult than I had previously thought. The main problems I kept running into was due to the incorrect versioning of CUDA, TensorFlow and graphics drivers. The lack of superuser privileges with lab machines made it more difficult in terms of flexibility.

While dealing with these issues, I simultaneously began experimenting with a machine learning framework known as 'RunwayML' which had a some good reviews. RunwayML has a StyleGAN implementation built into it. I found it very easy to setup, and was able to obtain very good results from their API. After a day's worth of training, my model started to really resemble the original dataset.

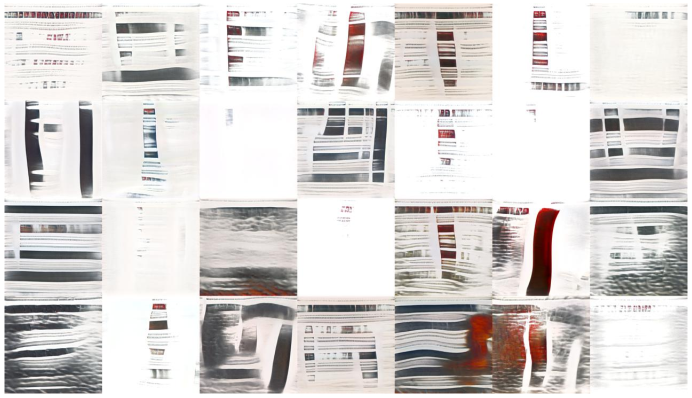


Figure 8: After 2200 steps

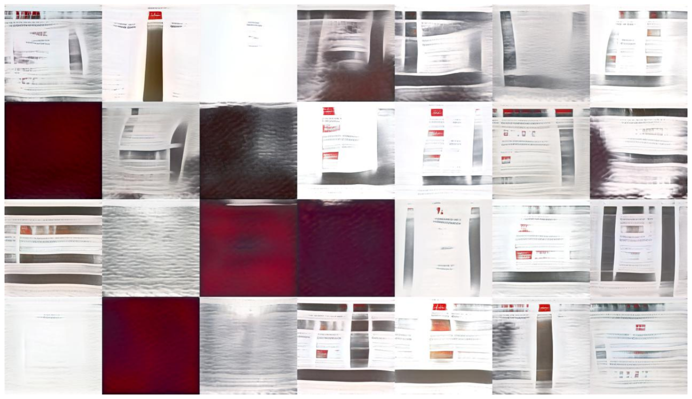


Figure 9: After 3000 steps

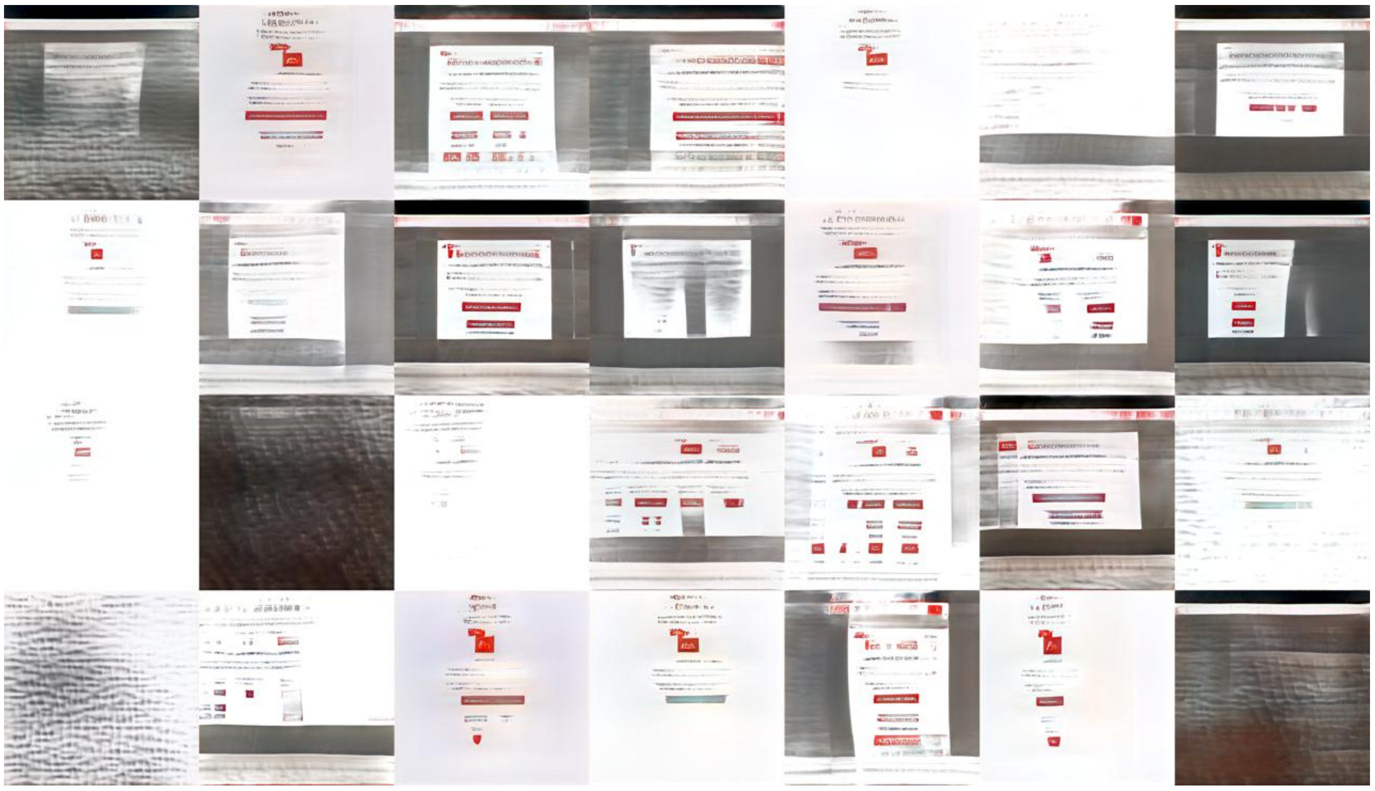


Figure 10: After 5000 steps

There is one downside, however. They charge us to rent their server-side GPUs. To complete the training I have done so far, cost me around 50 dollars. As Hossein suggested to me last week, there appears to be three ways we could move forward with the project:

- Implementation using Google Cloud GPUs with free credits.
- Implementation using CSU's HPC cluster with ACNS.
- Using RunwayML

Options 1 and 2 would not involve any extra cost, but might take a longer time to set up due to the complexities with the implementation, as stated above. I have been working on that simultaneously over the past week. If I can resolve the versioning problems within this week, that would be great progress.

Option 3 would not take any further implementation time, but would involve some cost to rent their GPUs.