Automatic Remastering of Classic Video Games

Zijie Zhu (zz2973)

December 10, 2022

1 Introduction

The 21st century has seen many great video games born. Although some of them are standalone works published by boutique studios, such as Braid, most of the masterpieces are parts of their corresponding franchise series from established game companies, such as Halo, Call Of Duty, and Hitman, etc. As many gamers noted, however, making an excellent sequel often seems even harder than creating an exceptional initiation. Multiple series have introduced games with groundbreaking graphics yet mediocre storylines or play styles, so disappointed players would rather go back to previous installments which lured them into the franchise and brought them immense joy. However, only then would they realize that their previously experienced graphics already seem unacceptable by current standards, and they end up in a dilemma between good graphics and good content.

From time to time, video game companies does release a remastered / remade / rebooted version of their previous works to bring back players into the franchise, such as Halo: The Master Chief Collections (MCC). But such work is not always cost-benefit efficient from the perspective of the game companies, so they make fewer of these than new games or simply charge the same price as a new game. One good example is the uproar among gamers in early 2022 caused by Naughty Dog charging a full price of $70 for a remake of The Last Of Us Part I. Therefore, it would be great if there is a procedure to automatically enhance the graphics of previous installments to current-generation levels. Such a procedure would be highly useful for both the game companies which can cheaply publish remastered versions and the gamers who can create graphics mods on their own. In this project, I will work on prototyping parts of such a procedure using Halo: Combat Evolved and Halo 2, two highly popular and classic first-person shooting games. Both Halo and Halo 2 come with the original version and the remastered versions, so they are suitable for testing in my project.

2 Related Work

The problem at hand concerns video-to-video translation, where we want to translate the original game to a remastered game. Naturally, video-to-video translation is based on image-to-image translation with potential improvements on scene coherence and an emphasis on fast prediction speed so the video can preserve high frames-per-second (FPS). One might think this particular translation is paired, since the original game and the remastered game have highly similar content. However, it is naturally hard to play the two games in the same precise way, and even the non-play-controlled cutscenes may take different camera angles. Therefore, the approaches we visit here focus on unpaired image-to-image translation.

One important dataset for benchmarking studies in this area is GTA[1] →CityScapes[2], where in-game rendered driving data from GTA V are compared with real-world street recordings. Notable

previous works utilizing this dataset include Cycle-Consistent Adversarial Networks (CycleGAN)[3], Contrastive Unpaired Image-to-Image Translation (CUT)[4], and Enhancing Photorealism Enhancement (EPE)[5]. There are also previous important works on unpaired translation and style transfer, such as Style Transfer by Relaxed Optimal Transport and Self-Similarity (STROTSS)[6] and Whitening and Coloring Transforms (WCT)[7], but they are older approaches with higher runtime that would result in low FPS, thus not suitable for our discussion. Since CUT is a sequel of CycleGAN with better performance and lighter architecture (thus faster prediction and higher FPS), we will ignore CycleGAN and focus on CUT and EPE in the Methods section later.

3 Data Curation

3.1 Getting and Cleaning Videos

I get all data from YouTube where gamers publish their campaign walkthrough videos. Such walkthrough videos are usually long (i.e. more than four hours) and have high resolution (i.e. at least 720p) and some have high frame rate (i.e. 60 FPS). Notice that our game usually consists of the single-player campaign and the multiplayer

There are several considerations in the data cleaning part, most of which are achieved with `ffmpeg`:

1. Initially, I aimed for the highest resolution of each video, such as 1080p or 4K. However, I could only get up to 7.5GB of GPU memory size on Google Cloud VM, and high-resolution dataset always quit with CUDA out-of-memory error. Sicne 480p is the highest resolution that the model could train, I had to settle with this resolution for all videos. Naturally, frame rate did not face the same problem and I preferred 60-FPS videos if available.
2. Certain gamers focus more on audience entertainment and lingered at the same location in game for too long, so I also picked walkthrough videos with no commentary when possible, as such videos usually focus more on gameplay.
3. To reduce noise in game scenery data, I also avoid videos with subtitles and gamer webcam. If all available videos have subtitles, I crop out the bottom area of the video containing the subtitles.
4. The opening and closing credits are all trimmed out to focus on gameplay.
5. The campaign gameplay data may still consist of cutscenes, which are argueably different from regular gameplay scenes. However, they are much harder to detect and clean out, and are relatively consistent with in-game data, so I decided to keep them.

3.2 Sampling Videos to Images

Since the videos are generally more than five hours long, we have ample data to train the model, but we need to find a balance between dataset size and training speed. The default parameters for the original CUT model are suited for training its `grumpy cat` dataset with around 300 images in both the original and the transferred domain. To assemble a dataset of similar sizes, I started with sampling videos once every minute. As I will elaborate below, however, I later moved on to sampled videos once every second to build a much larger training dataset. For testing, I sample all frames of the 60 FPS video and later piece the transformed pictures back to a 60 FPS video.

4 Methods

4.1 Constrastive Unpaired Image-to-Image Translation

Constrastive Unpaired Image-to-Image Translation (CUT) is a sequel to the famous Cycle-Consistent Adversarial Networks (CycleGAN), published by the same group of researchers. Recall that the unpaired image-to-image translation problem translate an image x from domain A to an image x' in domain B. The gist is to preserve the content (e.g. shapes) but replace the appearance (e.g. textures). Previous works such as CycleGAN analyzes not only $A \to B$ but also $B \to A$ thus forming a "cycle", while CUT does not rely on it because cycle-consistency may be too restrictive with its bijective assumption and training cycle-consistency takes additional time. Instead, CUT aims to maximize the mutual information between A and B with constrastive learning. Using the terminology of contrastive learning, wanted (sub)images are called positives while those unwanted are called negatives. Notably, CUT argues that internal negatives sampled from the same image x (blue boxes in Figure 1) as the positive (e.g. the rifle scope in the orange box in Figure 1) benefit the training more than sampling external negatives like other images x^*. This design not only results in lighter model architecture and memory usage, but also enables the translation to be trained on single images (we will come back to this in a later section).

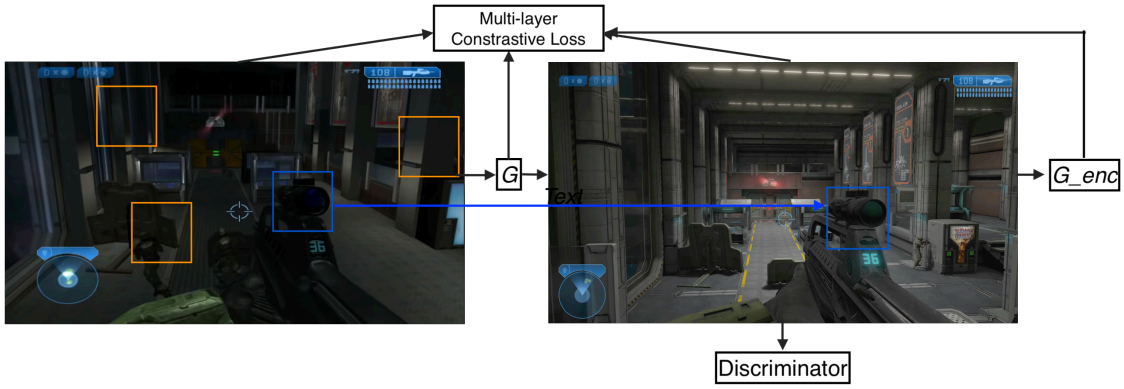


Figure 1. Illustration of Patchwise Contrastive Learning with the game dataset in hand. G refers to the generator function and G_{enc} refers to the encoder part of G.

Mathematically, denote G as the generator function, D as the discriminator function, $X = \{x\}$ as the input domain, and $Y = \{y\}$ as the output domain. Compared with other approaches with multiple loss functions, CUT only has two main loss functions[4]:

1. Adversarial loss: $\mathcal{L}_{GAN}(G, D, X, Y) = \mathbb{E}_{y \sim Y} \log D(y) + \mathbb{E}_{x \sim X} \log (1 - D[G(x)])$. Intuitively, minimizing this loss encourages $D[G(x)] \to D(y) \Rightarrow G(x) \to y$ so the transformed x becomes visually similar to y.
2. Noise-Contrastive Estimation (NCE): $\ell(v, v^+, v^-) = -\log \left[\frac{e^{vv^+}}{e^{vv^+} + \sum_{n=1}^{N} e^{vv_n^-}} \right]$ where v is the query sample, v^+ is the positive, v_n^- is the n-th negative. Intuitively, the term inside the bracket represents the probability of the algorithm correctly identifying the positive from all the negatives.

Therefore, the overall loss function is $\mathcal{L} = \mathcal{L}_{GAN}(G, D, X, Y) + \lambda_X \mathcal{L}_{PatchNCE}(G, X) + \lambda_Y \mathcal{L}_{PatchNCE}(G, Y)$. The CUT model corresponds to $\lambda_X = \lambda_Y = 1$ and the FastCUT model corresponds to $\lambda_X = 10$ and $\lambda_Y = 0$.

4.2 Enhancing Photorealism Enhancement (EPE)

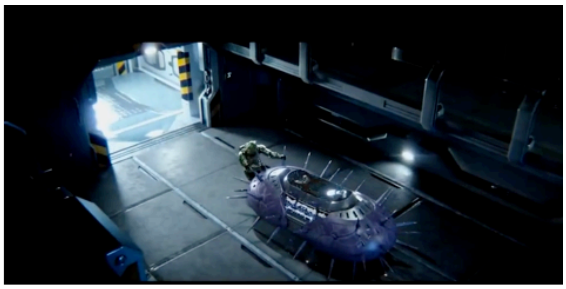
Enhancing Photorealism Enhancement (EPE) is developed by the same group of researchers who created the Play-For-Data tool to assemble the GTA dataset with ground truth semantic labels. Before we dig into EPE, it is worth introducing Play-For-Data which is a tool to hook onto the game and record two parts of the game rendering:

1. When and what types of assets are created in game, e.g. streets, pedestrians, cars, etc. This enables the researchers to create ground truth semantic labels for games like GTA V and result in the GTA dataset.
2. G-buffers created by deferred shading, such as normal maps, depth maps, albedo etc.

It turns out that the key innovation in EPE is extracting features from G-buffers and feed the extracted features into the generator. The ground truth semantic labels also separate the scene into different streams so different types of objects (e.g. streets vs cars vs trees) can be better transformed. Another innovation in EPE is the use of robust semantic labels (e.g. MSeg[8]) and ImageNet[9]-trained VGG-16[10] network in the discriminator, which produces better realism score maps. EPE authors also argue that small crop size (e.g. 7% of the original image size) reduces the hallucination effects and yields better performance. In comparison, CUT authors suggest keeping crop size above 40% of the original image size. As a result, I experimented with different crop size ratios and show the results in the Experiments section below.

On the task of transforming GTA V to be more photorealistic, EPE seems to beat other approaches including CUT by a sizeable margin, and initially I was also interested in adapting it for the remastering project. However, I decided to continue with CUT rather than EPE for the following reasons:

1. G-buffers and ground truth semantic labels are hard to record and collect. Even though the EPE authors created the Play-For-Data capturing tool, it would take me too long to play hundreds of hours of games to create my own dataset. In addition, there are no public dataset containing G-buffers due to legal reasons and it is hard to replicate the EPE success.
2. Robust segmentation does not work well on games focused beyond real world, e.g. Halo games. See Figure 2 below for more details.
3. The VGG network is also trained on real-world data (i.e. ImageNet) and may harm the learning of game scenes. As the CUT authors noted[4], "VGG…can be used in paired image translation tasks…[but] the frozen network weights cannot adapt to the data on hand."

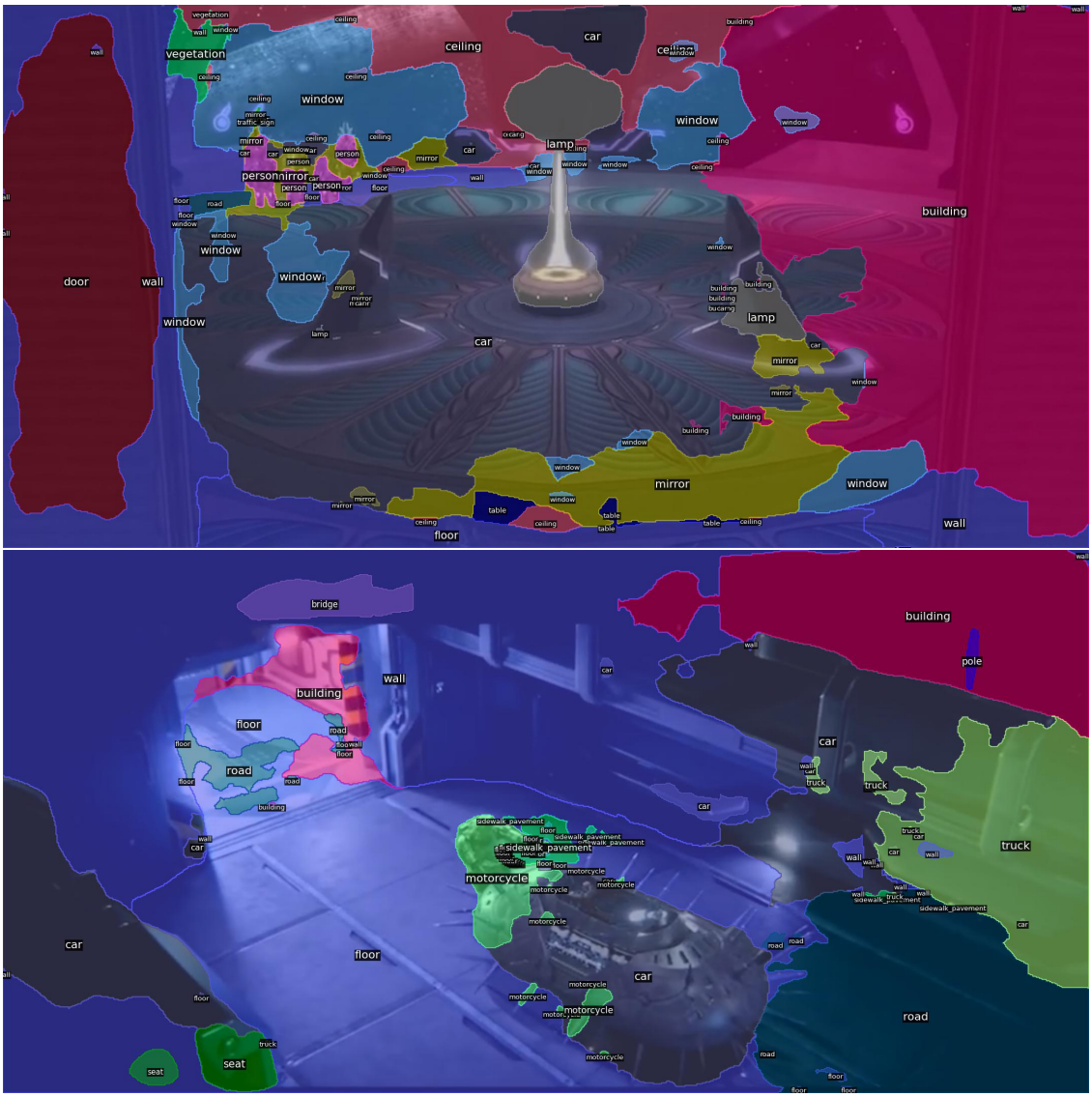


Figure 2. Robust segmentation (i.e. here MSeg) fails on game scenes. The first row shows two scenes from the original and the remastered Halo 2 games respectively. The second row shows robust segmentation fails on the original (top-left) scene and the third row shows semantic labeling fails on the remastered (top-right) scene despite more realistic graphics. Understandably, no semantic labeling models are trained on game data, which itself is very limited (The second and third row are showed as large pictures as otherwise the labels are too small to read).

5 Experiments

5.1 Small Crop Size

In the first iteration of my experiments, I ran CUT with small crop sizes as the EPE authors suggested for reducing hallucination effects. More specifically, since my inputs are 854x480, I selected 128x128 as the crop size which corresponds to 4% of the original image size. However, the results were not impressive; the model seemed only to learn an opaque pink shader across all situations rather than improving specific scenes.

Figure 3. Small crop sizes did not learn graphics difference very well.

5.2 Larger Crop Size

Next, I ran with larger crop size (480x480 for image size of 854x480). Unfortunately, I could not make crop size the same size as the original video as CUDA would go out of memory. This time, the results start to look interesting so I allowed the training to go as deep as 175 epochs, of which the first 100 epochs were regular and the next 75 epochs featured slowly decaying learning rates.

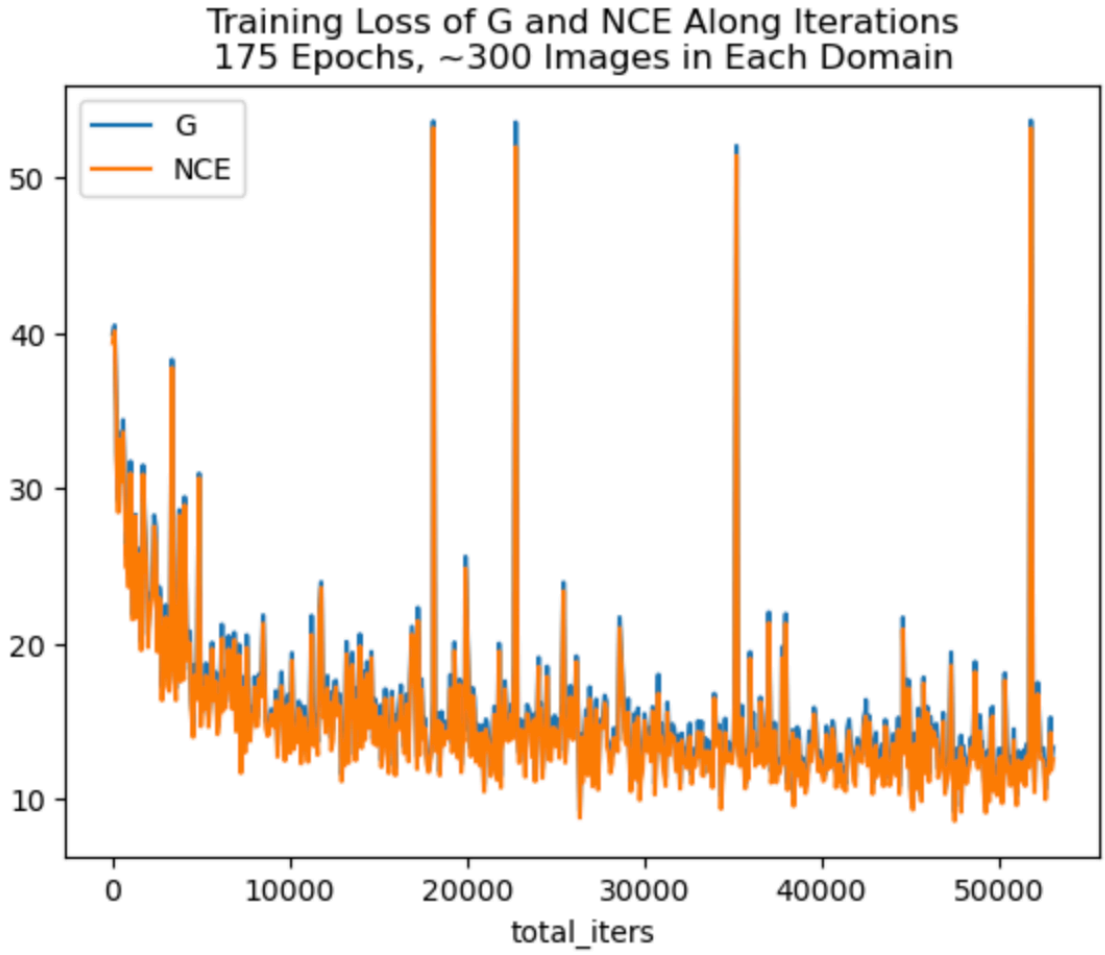


Figure 4. FastCUT training loss (adversarial and patch noise-constrastive estimation) for 300

samples and 175 epochs.

In Figure 5 below, we can see the original game scenes on the left and the automatically remastered ones on the right. The first example added sand and rock textures to the lower-right corner and increase the lighting of the entire scene. Both the Jeep and the silver spaceship in the next example have shinier metal reflection and look more metalic. It's important to point out that the first row scene is from player gameplay and the second row scene is from the cutscene, which means the model is robust enough to improve the graphics of both the realtime rendered scenes as well as the pre-rendered cutscenes. The last example, however, highlights the problem that the model sometimes generates a black hole in the image, mostly over the HUD display elements such as the health percentage (HP) bar or the ammunition bar, which seems most likely a problem of insufficient training samples.

Figure 5. Original game scenes on the left and the automatically remastered ones on the right. First two rows show encouraging results and the last row shows the problem of black holes.

5.3 Increased Sample Size

To fix the problem of black holes, I sampled the videos at an increased frequency of once every five seconds and created a new dataset with around 4000 images in each domain, a size comparable to the finely labeled parts of the CityScapes dataset. To compare apples vs apples, I keep the total number of iterations roughly the same by reducing the number of epochs to 25 as each epoch now takes many more iterations.

Figure 6. Remastering effect comparison. On the left shows the remastered scene with small training samples. In the middle shows the remastered scene with increased training samples. On the right shows the original game scene. Video version of this comparison is available in the presentation. Notice how the left image has the problem of black holes over the HP bar on the top-right of the HUD display while the middle image does not.

6 Conclusion

In this project, I have experimented with automatic remastering of classic video games using deep learning models such as CUT. EPE was another model that I experimented with but in the end was not suitable for the project for multipled reasons listed above. One potential improvement is that the training would greatly benefit from a rough match between the original and the remastered scenes. Even though the task is not strictly paired image-to-image translation, there is significant overlap between the original and the remastered games even after excluding the cutscenes. For example, specific dialogues only trigger after the player performs an assigned action or reaches a certain location, so scenes immediately before and after such anchor points must be highly similar to their counterparts in the remastered game. Then for each anchor point at time t, assign bell-shaped weights w over $[t - \delta, t + \delta]$ when sampling patches and calculating loss.

7 Bibliography

1. Richter, Stephan R., Vibhav Vineet, Stefan Roth and Vladlen Koltun. "Playing for Data: Ground Truth from Computer Games." ArXiv abs/1608.02192 (2016): n. pag.
2. Cordts, Marius, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth and Bernt Schiele. "The Cityscapes Dataset for Semantic Urban Scene Understanding." 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016): 3213-3223.
3. Zhu, Jun-Yan, Taesung Park, Phillip Isola and Alexei A. Efros. "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks." 2017 IEEE International Conference on Computer Vision (ICCV) (2017): 2242-2251.
4. Park, Taesung, Alexei A. Efros, Richard Zhang and Jun-Yan Zhu. "Contrastive Learning for Unpaired Image-to-Image Translation." European Conference on Computer Vision (2020).

5. Richter, Stephan R., Hassan Abu Alhaija and Vladlen Koltun. “Enhancing Photorealism Enhancement.” IEEE transactions on pattern analysis and machine intelligence PP (2021): n. pag.
6. Kolkin, Nicholas I., Jason Salavon and Gregory Shakhnarovich. “Style Transfer by Relaxed Optimal Transport and Self-Similarity.” 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2019): 10043-10052.
7. Li, Yijun, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu and Ming-Hsuan Yang. “Universal Style Transfer via Feature Transforms.” ArXiv abs/1705.08086 (2017): n. pag.
8. Lambert, John, Zhuang Liu, Ozan Sener, James Hays and Vladlen Koltun. “MSeg: A Composite Dataset for Multi-Domain Semantic Segmentation.” 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2020): 2876-2885.
9. Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg and Li Fei-Fei. “ImageNet Large Scale Visual Recognition Challenge.” International Journal of Computer Vision 115 (2014): 211-252.
10. Simonyan, Karen and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition.” CoRR abs/1409.1556 (2014): n. pag.