CSCI 1430 Final Project Report: theRealPetGenerator

theRealPetGenerator: Jacob Vietorisz, Sean Yamamoto
TA name: Eric Tang, Brown University

May 8, 2023

Abstract

We implement a deep convolutional generative adversarial network (DC-GAN) to produce fake images of cats. Unlike traditional GAN architectures, which use pooling layers, DC-GANs use transpose convolutions to up-sample in the generator and strided convolutions to down-sample in the discriminator. We demonstrate that even with small and diverse training data sets ($<$ 15k images) we produce convincing images of cats. The success of our network relies on reproducing a small number of distinctive feline features, such as ears, eyes, and noses.

1 Introduction

We creating a deep convolutional generative adversarial network (DC-GAN) to fabricate images of cats. Generative models are a subset of machine learning models that learn to reproduce content that mimics the data used for training. In computer vision this involves using digital images of real-world content to generate convincing images whose content has no real-world counterpart – they are purely made up by the model. Generative adversarial netowrks (GANs) are a type of generative model comprising two closely-related networks: a generator network, which learns to produce content, and a discriminator network, which learns to distinguish between real and fabricated data. While a well-trained generator is the goal and shining star of a GAN, the discriminator is a critical tool for training. The premise of a GAN involves pitting the two networks against each other – hence the name 'adversarial' – where the discriminator learns to distinguish between real and fake content, and the generator is optimized to create content that fools the discriminator. GANs were first introduced in 2014 and have since become a popular approach to generative AI. DC-GANs use convolutional neural networks (CNNs) as its generator and discriminator, which feature architectural improvements for stable training compared with earlier published approaches. Here we detail our implementation of a DC-GAN that learns to generate images of cats *de novo* using the 'dogs vs cats' dataset freely available on Kaggle. Our work comprises the code for the network architectures as well as code for the training and result visualization procedures.

2 Related Work

Our implementation is based on a pioneering paper by Radford et al. that provides detailed CNN architectural suggestions for stable training of a DC-GAN [1]. Our project was programmed in Python 3.9 using the following external packages: Pytorch, Matplotlib, Numpy, and Yaml.

3 Methods

Our task was to create and train two distinct CNNs that together were capable of stable training on a data set of images. In order to do this, we needed to design the network architectures for the generator and the discriminator, design a training pipeline that optimizes both networks in concert, and select an appropriate data set of real images.

Our network architectures were inspired by the guidelines of [1]. Diagrams of our networks are shown in Fig-

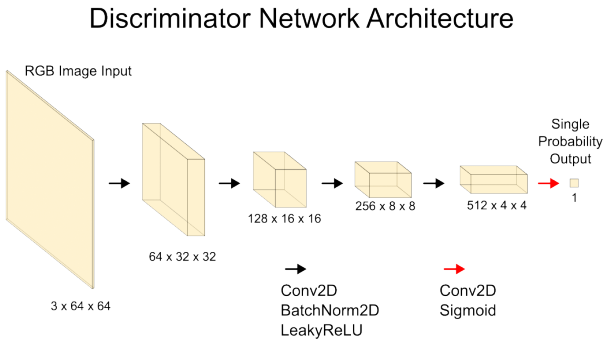

Figure 1: Our final discriminator network architecture

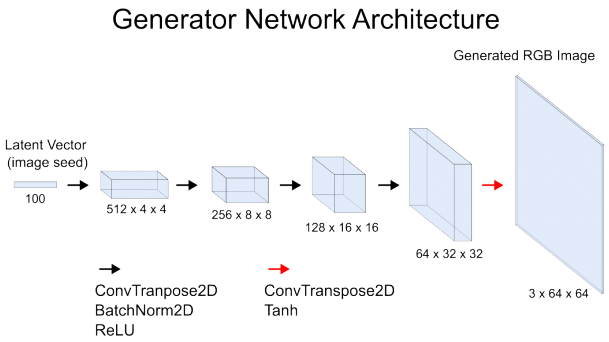

Figure 2: Our final generator network architecture

ures 1-2. Figure 1 shows a schematic of our discriminator network, which is a binary classifier that takes in a 64x64 pixel RGB image as input as outputs a single value on the interval [0,1]. This output is a probability that represents a prediction of whether an image is real or fake. If the output is > 0.5, the prediction is that the image is real; if the output is ≤ 0.5, the prediction is that the image is fake. The loss of the generator is calculated with a cross entropy loss using the ground truth labels of 1 for real images and 0 for fake images. The discriminator architecture reduces the size of the image input while increasing the number of channels at each layer. Each kernel in the convolution layers operates depth-wise on all of the channels of the layer's operand. Instead of using max pooling layers to down-sample, our discriminator uses strided convolutions, which is a distinctive characteristic of a DC-GAN. The activation function between each layer is a 'Leaky ReLU' function, which is like a typical ReLU except that it assigns non-zero, negative activations for negative values according to:

$$\text{ReLU}(x) = \begin{cases} 0, & x \geq 0 \\ -\alpha x, & x < 0 \end{cases} \quad (1)$$

where α is a hyperparameter. We left this parameter as the Pytorch default. All convolutions were performed with the same stride of 2 and the same kernel size of 4x4. The final activation funtion uses the default Pytorch sigmoid function so that the output is between 0 and 1.

Figure 2 shows a schematic of our generator network, which takes a single dimensional tensor of length 100, called a latent vector, as input and outputs a 64x64 pixel, 3 channel RGB image. The values of the latent vector are random noise on the interval [0, 1]. Instead of max unpooling layers for up-sampling, our generator uses transpose convolutions to up-sample the vector to the shape of the real training images read by the discriminator. This is another typical feature of a DC-GAN. As per the recommendations of Radford et al., we used ReLU activation functions in each internal layer, and hyperbolic tangent activation function in the final layer. The idea with this final activation function is to jump-start the discriminator, which will quickly learn that images with negative values are fake. This is desirable because accurate discriminator predictions allow the generator to properly optimize its weights. The generator therefore also learns not to output negative values. In terms of layer dimensions, the architecture of the generator is approximately the inverse of the discriminator, compressing the number of channels while increasing the size at each layer.

Our training pipeline simultaneously trains the discriminator and the generator. Both of these networks together have over 7 million tunable parameters (weights), so in order to train efficiently, we used 8 GPUs from the Brown University Compute Cluster (Grid). Using multiple GPUs allows us to process data in parallel and average the results over small batches when tuning the weights. We were able to access the Grid using SSH and Fuse to remotely edit and run scripts. Figure 3 illustrates the training cycle that is repeated for every batch in each epoch. Using randomly generated latent vectors, we generate 32 fake images using the generator. These images are given the ground-truth label 0, which marks them as fake for calculating the dis-

criminator loss. 32 real images with the ground-truth label 1 are also pulled to feed to the discriminator. For simplicity and for logging training statistics, we calculate the discriminator predictions and losses on the real and fake images separately. We then update the discriminator weights according to the calculated binary cross entropy loss using an Adam optimizer with a learning rate of 0.0002 and $\beta = 0.5$, according to the suggestions of Radford et al. We can then use discriminator's predictions to calculate the binary cross entropy loss of the generator based on whether the generated images fooled the discriminator. To do this we flip the labels of the fake images to calculate the binary cross entropy loss and update the weights of the generator using an Adam optimizer with the same hyperparameters as for the discriminator. This cycle is repeated 12.5k/32 times per epoch.

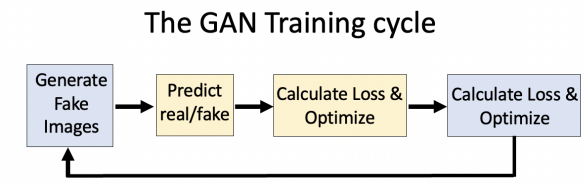

Figure 3: Schematic of the GAN training cycle. The colors of the boxes corresponds to which network the is involved in each step, either the generator (blue) or the discriminator (yellow).

We selected the 'Dogs vs Cats' data set in order to train our model to produce images of household pets. Our choice was motivated by our ambition to fabricate cute images whose complexity far surpasses that of benchmark data sets for generative models, such as MNIST. The disadvantage of our data selection was that the entire data set comprises only 37k images of both cats and dogs with an enormous diversity of color, breed, pose, and even number of animals in each photo. In order to improve our performance, we decided after attempting training to simplify our data by using only the cat images. We were left with only 12.5k images on which to train, well below the recommended data set size for training a generative model. We overcame this limitation by augmenting our data: resizing the images, taking random internal crops, and randomly flipping images horizontally. These measures did not increase the total number of training images, but rather were introduced randomness so that the discriminator needed to make predictions on never-before-seen images each training cycle.

Our implementation also used custom methods for making and loading models, generating random latent vectors, making constant latent vectors and storing the images they generate at various points during training, generating and visualizing new images from the generator, visualizing real images from the data set, pulling parameters from a configuration YAML file, and plotting losses, discriminator accuracy, and the discriminator predictions.

4 Results

We were able to get our model to perform reasonably well after tuning hyperparameters like the batch size and after introducing more uniformity in the real image data by excluding dog photos. Figure 4 shows samples from the real image data set after augmentation. The resizing causes slight distortions in the cat's proportions (for example, the cat in the top row, second column); however, it is clear that 64x64 pixels is a sufficiently high resolution to preserve details in the images. Despite our attempts to make the data set more uniform, there was still a significant amount of variation in the background, brightness, pose, and color of the cats.

Figure 5 shows the images generated from the same latent vector at different points over 600 training epochs. It is clear that our generator initially generated random noise before incrementally learning to produce feline features such as the eyes, face, nose, and ears.

Figure 6 shows 20 fake images produced by our generator after 1200 epochs of training. These images represent approximately the best 10% of generated images, taken from approximately 200 randomly generated images. Distinctive features like the eyes, ears, nose, and fur patterns are clearly visible. Generated images include both individual cats as well as pairs.

4.1 Technical Discussion

Our implementation of a DC-GAN demonstrates that this approach to generative modeling is capable of producing convincing results from minimal training data. The small size of our training set does however strongly affect the

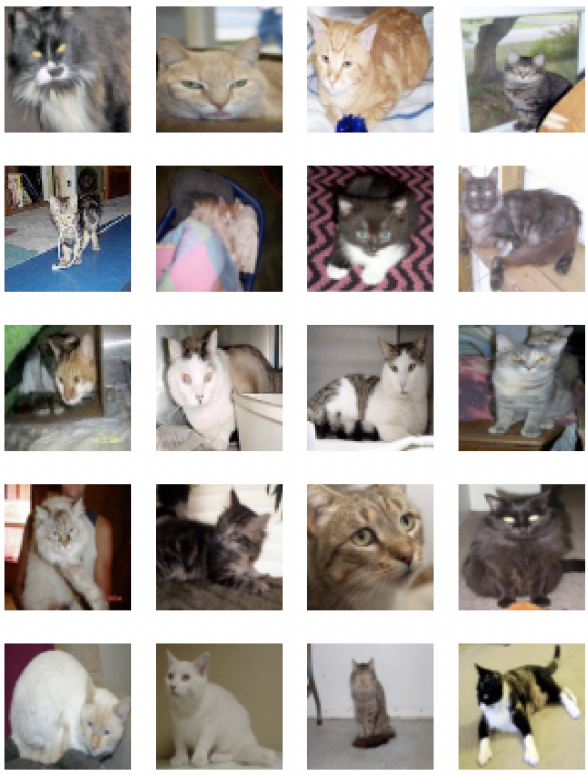

Figure 4: Sample photos from the real image data set after augmentation.

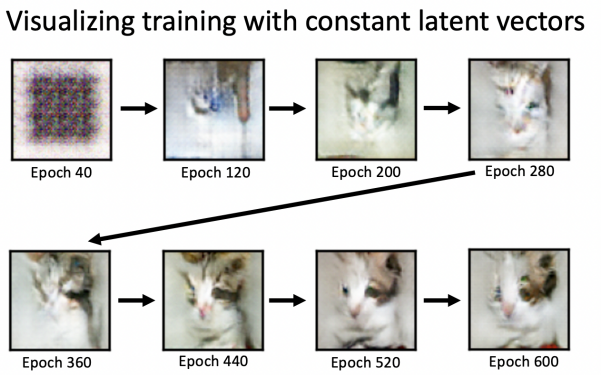

Figure 5: Generated photos from the same latent vector at various points during training.

training dynamics of our model. The theoretical equilibrium of a GAN is reached when the generator consistently produces images that are indistinguishable from the real training set, and the discriminator makes 50% confidence classifications on each image. The losses of both our generator and discriminator, along with the discriminator accuracy and average prediction, are plotted in Figure 7. It is clear from the plot of the losses that the discriminator becomes performance much faster than the generator, whose losses consistently increase even as the generator learns to produce better and better images. The discriminator's classification accuracy is extremely high, which suggests that the discriminator has overfit the training set. Even with augmentation to introduce randomness in the real images, these random crops still preserve features in the images at the original (pre-crop) scale. Since the convolution kernels in the discriminator operate across the entire image, the same kernels will generate a high response on an image regardless of where a particular feature appears.

We attempted to mitigate the discriminator's over-fitting by updating the discriminator's weights every other training cycle. This approach produced good results after 1200 epochs of training, but it is possible that further tuning of the model's hyperparamters would lead to more efficient training of the generator. One such approach involves 'smoothing' the ground-truth labels so that real images are labeled for the discriminator as $1 - \epsilon_1$, and the fake labels are ϵ_2 for small hyperparameters ϵ_1 and ϵ_2.

4.2 Socially-responsible Computing Discussion via Proposal Swap

We will address the three concerns that our critics submitted in response to our project proposal. First, we certainly share the concern that our discriminator over-fits the training set instead of producing a robust decision boundary especially given our model's relatively large size and our data set's relatively small size. In order to mitigate this, we introduced data augmentation to artificially expand our data set. In the future, we could also collect a larger data set or introduce other techniques to mitigate over-fitting such as dropout layers. Next, there are definitely stakeholders other than those that we listed since the goal of this project is to produce fake images that look as realistic as possible. The critics bring up a good point - this model

Figure 6: Generated (fake) photos from random latent vectors after 1200 epochs of training. The images shown here represent approximately the most convincing 10% of generated images, selected from 200 randomly generated images.

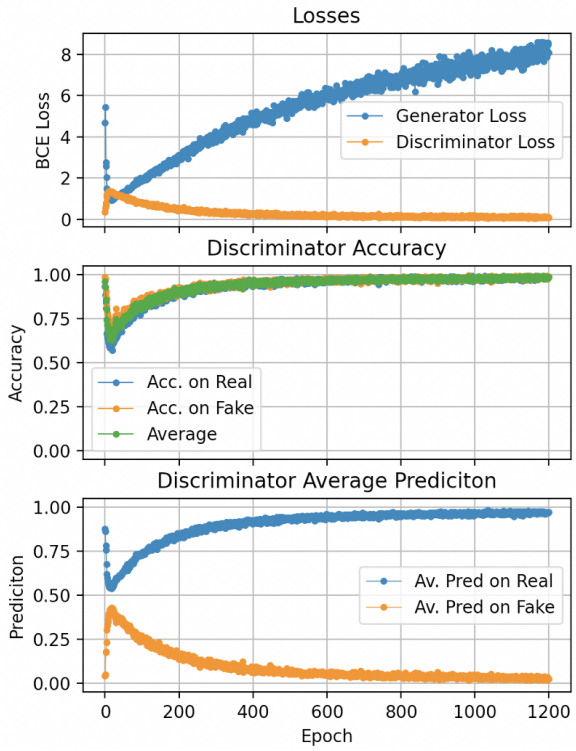

Figure 7: Plots showing statistics of the network losses for the discriminator and generator, the generator accuracy, and the generator's average predictions. Training statistics taken from the same training that produced the generated images shown in Figure 6

could be used to create fake imagery and disinformation. After considering that, we are all stakeholders in this model and similar models because we are all susceptible to being affected by disinformation. Finally, our critics suggest that our model should include a watermark so that people know that our generated images are fake. This is an interesting suggestion because it could communicate to viewers that our images are generated. The watermark would of course need to be added after training so that the discriminator did not learn to classify images based on the presence or absence of the mark. This is the approach of other generative models like Dall·E 2. However, someone could easily remove a watermark from an image with photoediting software, so, while we share the concern that our model and other generative models can be used for harm, we don't think that this is the most effective way to mitigate the risk. A better, though by no means ideal, solution could involve binding each generated image to a secure archive of the generating and editing history.

5 Conclusion

We successfully developed and trained a DC-GAN that is capable of fabricating images that are clearly recognizable to human viewers as cats. While our model was trained specifically to produce images of cats, the DC-GAN architecture is applicable to any image type so long as a real training set contains enough images with sufficient uniformity. We do not have specific metrics for evaluating the

feasibility of training on a particular data set, but we are interested in the future to train our model on other kinds of images and at higher resolutions. Before extending the scope of our model's applications, an appropriate next step would be to further optimize the training regimen to save compute energy and time in future applications.

Appendix

Team contributions

Person 1 This person developed the majority of the code base, including the runner code, generator, discriminator, YAML file, logging processes and files, data collection, and data visualization tools. This person also implemented multi-GPU training and evaluation on the Brown Compute Cluster and helped Person 2 set up their code base, virtual environment, and SSH tools so they could develop and run the code using an efficient workflow. This person also collected and cleaned the dataset and did preliminary model training to make sure everything was set up correctly and the model was learning. This person provided ongoing help to person 2 related to clarifying code and SSH operation and discussing ideas for improving performance.

Person 2 This person engaged in extensive tuning and trial and error to optimize the model's performance. This included tuning hyperparameters, introducing data augmentation, adjusting the model architectures, and implementing these changes in the training regimen. This person improved the performance of our generator profoundly from generating unrecognizable images to the distinctly feline images shown in Figure 6. This person developed the data loading process and tools to visualize the performance of the generator. They also produced each of the figures in this report and our poster board including the figures portraying the generator and discriminator architectures. Additionally, this person designed and built the entire poster board and wrote the majority of this project report.

Bibliography

[1] Radford, Alec and Metz, Luke and Chintala, Soumith. "Unsupervised representation learning with deep convolutional generative adversarial networks". ArXiv preprint arXiv:1511.06434. 2015.