Neural Networks Final Project

John Farrell, Lawton Manning, Liyao Zhang

December 5, 2020

1 Background

Generative Adversarial Networks (GANs) have been popular lately in recent Deep Learning publications. GANs frequent projects such as face aging, video prediction, super resolution, and much more. This class has covered basic feed-forward neural networks and the basic ideas associated with deep learning, so we wanted to use this project as a chance to learn something new and more complicated, so we decided to try to create a simple GAN model.

Lawton Manning has already presented a solid introduction of GANs during our class, but we will cover some very basic GANs concepts before proceeding. It can be noted that our GAN implementation is a *conditional GAN* which allows us to additionally input the class we are trying to generate.

1.1 Basic GAN Structure

At the most basic level, GANs are comprised of two sub-models: a generator and discriminator. The training process flips between each sub-model learning after each step (i.e. the generator learns one step, the discriminator the next, and back to the generator). Overall, the discriminator's role is to discern what is real and fake, while the generator is attempting to "fool" the discriminator. We are training a generator to learn how to create something close enough to the inputs, but without ever seeing the inputs.

GANs come in many flavors with added benefits. We decided to implement a conditional generative adversarial network (cGAN). Typically, a GAN trained on a multiclass dataset has no limitation to generate for a specific class. cGANs improve the original GAN with an additional input layer that specifies the category to be generated. This allows us to control what the generator draws.

1.1.1 Discriminator

As mentioned previously, the discriminator is the model which attempts a binary classification of real or generated inputs. As a cGAN, when we feed data into the discriminator we will have a list of inputs (X) with associated input categories (labels), but also an additional input describing the 'real' or 'fake'-ness of the

inputs. For instance, this additional input would be all ones if the images are taken from the dataset, but all zeros if the generator creates them.

1.1.2 Generator

The generator is the model trying to fool the discriminator. We feed an input of random numbers known as the "latent space" into this model to seed learning, and over time the generator learns how to interpret these random values. In GANs, the discriminator can be understood as a "convolutional network" while the generator is a "deconvolutional network." While we are already familiar with convolutional networks (especially useful in image classification and identifying translation-independent features), deconvolutional networks are a slightly newer idea. It can be best understood as working backward – constructing the original input by removing filter distortions. [20] Since we are implementing a cGAN, we add one more input for the category specification.

1.2 Google's "Quick, Draw!" Dataset

The Quick Draw Dataset is a collection of 50 million drawings across 345 categories, contributed by players of the game Quick, Draw!. The drawings were captured as timestamped vectors, tagged with metadata including what the player was asked to draw and in which country the player was located. You can browse the recognized drawings on quickdraw.withgoogle.com/data. [Goo]

2 Related Work

The first GAN related paper introduced was [Goo+14] which presented the concept of a GAN as opposed to similar concepts in the field like predictability minimization, where two networks would try to optimize an objective function but have different goals. A GAN is more aptly described as a minimax or zero-sum game: pitting two networks against each other where high loss on one is low loss on the other and vice versa. These networks are thus forced to compete and learn from each other's actions. Instead of being applied to something like a game, GANs are often applied to media such as images, videos or sound.

The authors of [Goo+14] pointed out that GANs can be difficult and slow to train, which is why many have moved from traditional GANs to their extensions. This provided new structure and optimizations to common GAN problems. One such extension is a Conditional GAN (cGAN) proposed by [MO14]. This extension solved the unpredictability of a traditional GAN by altering the structure of the generator and discriminator to take in class labels or other data modalities as additional inputs. For example, a cGAN created to generate MNIST digits may use the class labels of those digits to generate more digits of the same type. The discriminator would be given a class label and an image and be asked "Is this a real 2?". The generator would be given a random noise vector (like in a traditional GAN) and a class label to be told "draw a 2".

A large data set used for recognizing and generating hand-drawn sketches is the Google Quick Draw data set [Goo]. There have been other approaches using generative networks to generate hand-drawn drawings from this data set. One such application is the Sketch-RNN by [HE18]. This model uses a RNN to do a variety of tasks, among which are: reconstruct input drawings using randomization, unconditionally draw images of a given class, and predict future strokes in an incomplete drawing. By taking into account the order and timing of individual strokes, the Sketch-RNN can gain more information and actually draw as a human would draw. For our work, the GAN has no RNN component and treats the input drawings as one static image instead of a series of strokes. While not as flexible as Sketch-RNN, this type of GAN allows for training on a relatively larger data set since the static images can be made small (28 x 28 pixels).

Since the discriminator and generator of a GAN model on static images are basically convolutional and deconvolutional networks, the work by [Kab20] shows a potential design and weights for a discriminator model. This work achieved high accuracy of recognizing drawing classes of the Quick Draw data set. Although transfer learning is common, it would be difficult to formulate a discriminator using this model since a fully trained discriminator would cause a newly minted generator to not start out on equal footing in training.

3 Methodology

3.1 Downloading the Dataset

The original github repository[Goo] includes "the raw moderated dataset" and "preprocessed dataset". The preprocessed dataset are split into categories and different formats from the raw dataset and we chose to use the numpy bitmaps in the preprocessed dataset. Though we lose the information on the order of strokes, it is easier for us to load and explore the dataset in python. We then downloaded the .npy files with command line instructions:

```
conda install -c conda-forge gsutil
gsutil ls gs://quickdraw_dataset
gsutil -m cp gs://quickdraw_dataset/full/numpy_bitmap/*.npy .
```

3.2 Keras Custom Sequence Class

Google's Quick, Draw! '.npy' dataset is fairly large (38 GB). In order to work with the entire dataset and train our cGAN, we developed a custom keras Sequence class. This allows the data to be loaded and batched without loading all 28x28 grayscale images into memory at once. Without this custom class, we would have to work with a much smaller subset of data, but we know that Neural Networks are improved with more data so we decided to try our best to incorporate this feature. Once time started running out, however, we decided to

shrink the dataset to allow the quickest training. Otherwise we wouldn't have anything to talk about in our Results section.

3.3 Model Structures and Layer Flowcharts

The structure of the discriminator and generator models were co opted from a helpful online cGAN tutorial [Bro20] with some minor additions based on personal preferences. We made very minor changes to connect with our dataset's size, but for the most part stayed true to the model composition. Figure 1 shows the full flowchart for the discriminator submodel. Figure 2 shows the generator's flowchart. The next couple pages contain both flowcharts.

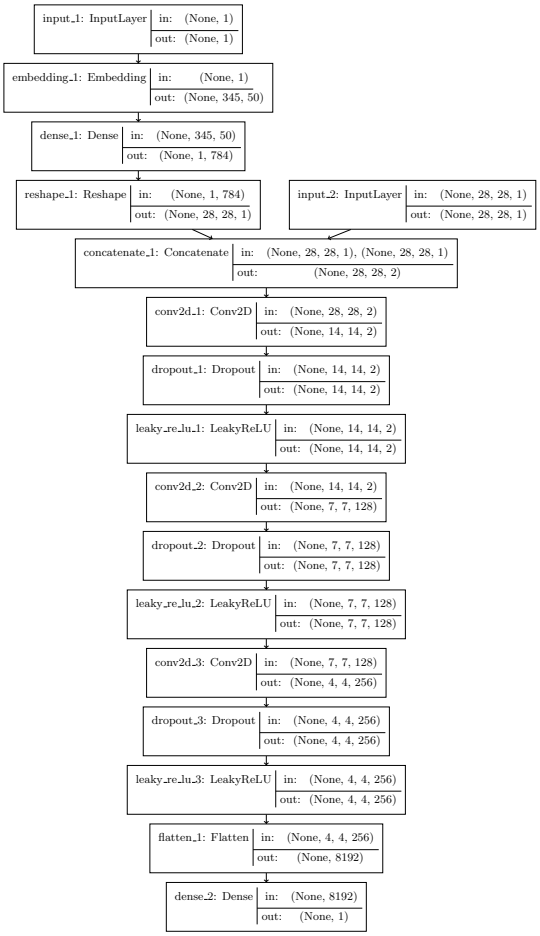


Figure 1: Discriminator Submodel: Layer Flowchart

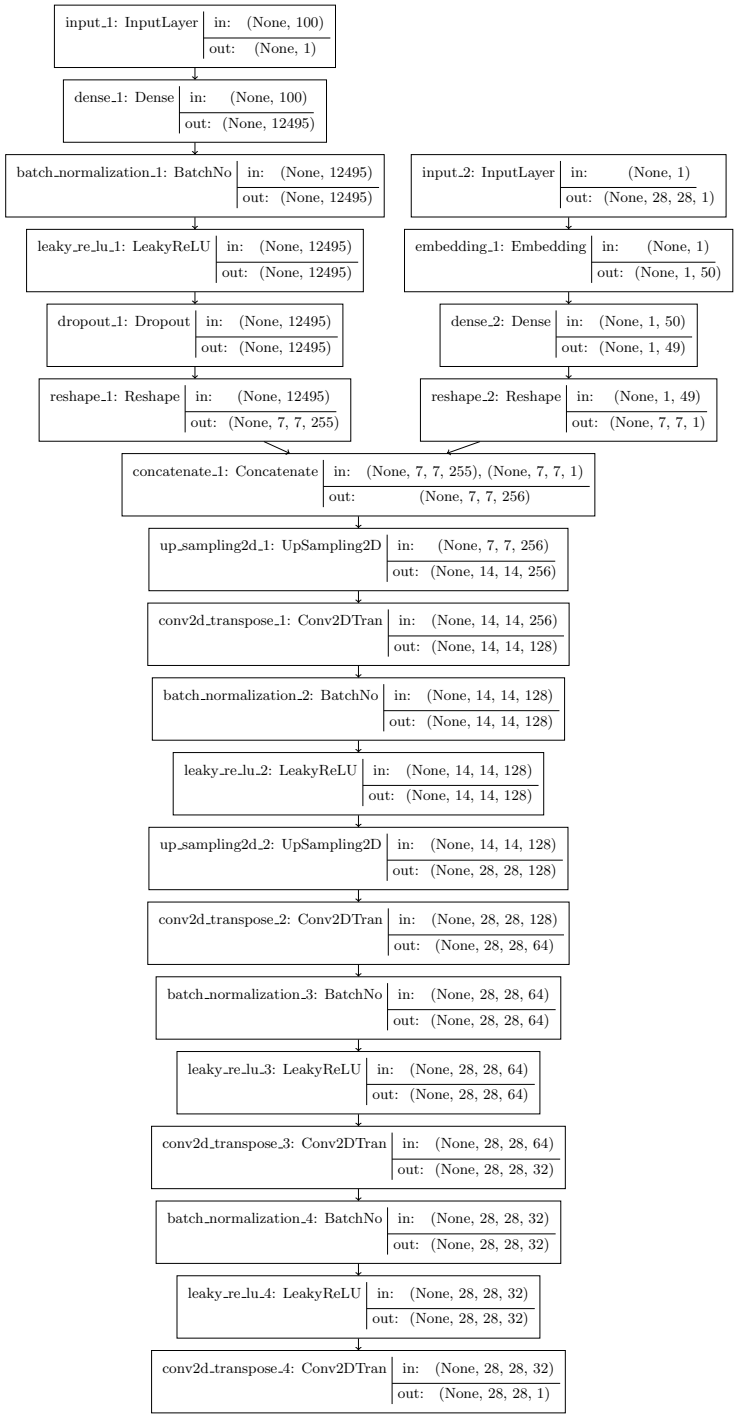


Figure 2: Generator Submodel: Layer Flowchart

4 Results

As our first venture into the deeper, more complicated end of neural networks, this was an incredible learning experience. Our first attempt at composing a cGAN did not result the way we hoped. Figure 3 shows a series of 16 categories being generated by the trained generator model we saved at the end of our first stage of training.

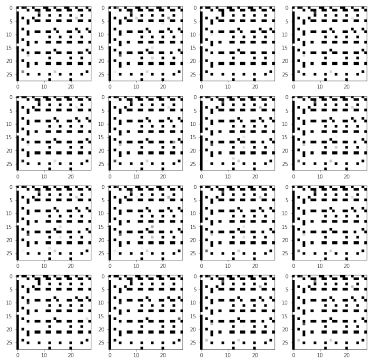


Figure 3: First Attempt at cGAN Generation

The generator produces only small variations among its outputs, despite being in distinctly different categories. In addition to the generator's failures, our discriminator was producing a loss of about 1e-14, and identifying real/fake images very well. This is problematic because a well trained network should accurately identify real/fake 50% of the time, but our generator was making it too easy. Unsure about how to probe our failed network, and with limited training time, we decided to test on a proven network [Eri] to evaluate our expectations of the data. With a little tweaking of inputs, we returned promising data. Figure 4 shows the generator's learned outputs for a limited input size of 16 categories with 1000 images each.

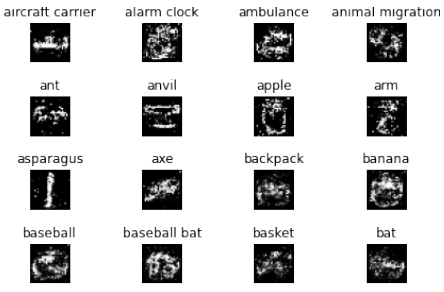


Figure 4: Generator Learning at 19500 Steps

5 Analysis and Discussion

We have uploaded a GIF of the generator's learning process on our GitHub. Every 500 steps a snapshot is saved of the current generator's learned parameters, and we are given insight in how different categories are learned. Figures fig. 5a and fig. 5b show snapshots of the generator at 500 and 1900 steps, respectively.

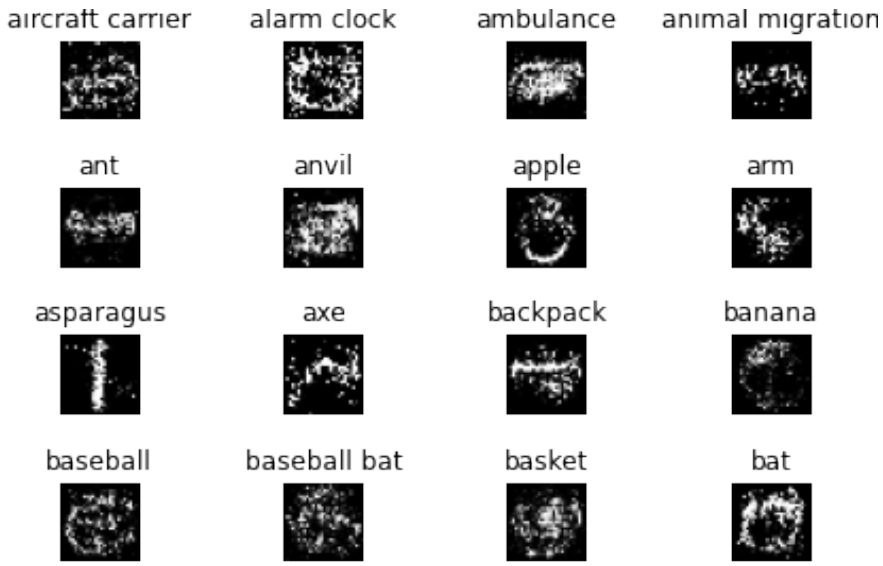
By observing and comparing learned outputs at different times, we learn about the symbolism behind the objects. For instance, the model learned the shape of 'apple' and 'asparagus' user-sketches within the first 500 steps. These are extremely simple shapes: the apple is circular with some kind of ornamentation on the top, and the asparagus is a straight, stick object center-screen. But for more nuanced symbols, such as animal migration, the model struggles to find consistent meaning.

This difficulty could also arise from the orientation of the object the illustrator presents. Looking at outputs for 'arm' and 'banana' we see a multitude of different orientations with no defining features. This can be interpreted as inconsistencies between individual drawings that haven't been extracted. This indicates that further training is necessary or the latent space must be increased to compensate for all the latent features.

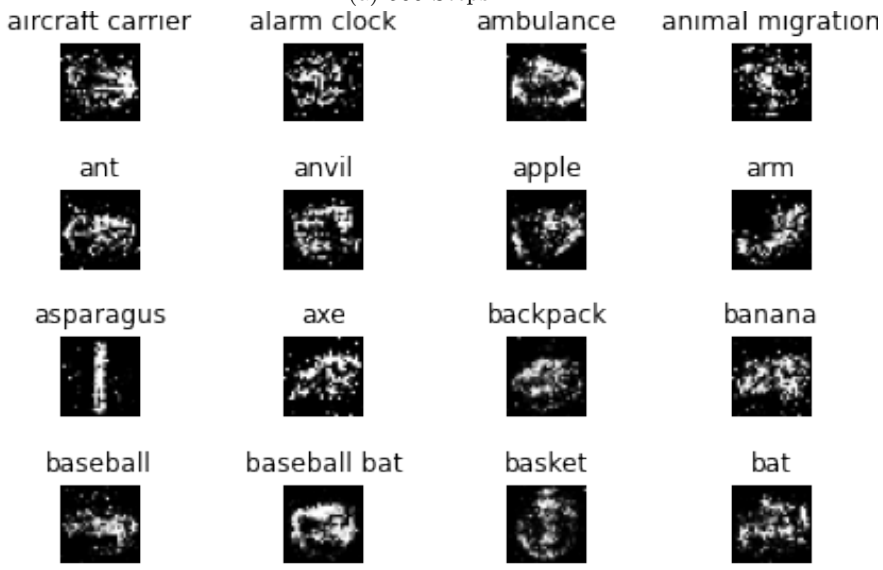
Another problem with the data set itself is that when the AI on the Quick Draw website correctly identifies an image, or the game times out before the user finishes drawing, the images can be incomplete. While we don't expect this systematic effect to cause one class to be less accurate than another, incompleteness is yet another latent feature that must be learned by our cGAN.

5.1 Next Steps

With more time we would have been able to train at length our first model. For every category in the Google dataset there were more than 100,000 images, with more than 300 categories available. The common complaint is that not enough data exists, but this time there was just too much to train our cGAN within a reasonable amount of time. If we continue this project, we would need to rigorously test our cGAN and then train it on Wake Forest University's VMs with dedicated GPUs.



(a) 500 Steps



(b) 19000 Steps

Figure 5: Generator Learning Snapshots

References

[Goo+14] Ian J. Goodfellow et al. "Generative Adversarial Nets". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 2672–2680.

[MO14] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets*. 2014. arXiv: 1411.1784 [cs.LG].

[HE18] David Ha and Douglas Eck. "A Neural Representation of Sketch Drawings". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=Hy6GHpkCW.

[Bro20] Jason Brownlee. *How to Develop a Conditional GAN (cGAN) From Scratch*. Sept. 2020. URL: https://machinelearningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/.

[20] *Generative adversarial network*. Nov. 2020. URL: https://en.wikipedia.org/wiki/Generative_adversarial_network.

[Kab20] A. T. Kabakus. "A Novel Sketch Recognition Model based on Convolutional Neural Networks". In: *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*. 2020, pp. 1–6. DOI: 10.1109/HORA49412.2020.9152911.

[Eri] Eriklindernoren. *eriklindernoren/Keras-GAN*. URL: https://github.com/eriklindernoren/Keras-GAN/tree/636ec0533df1d1ba2bfe4ec9ae7aa66bd7ee2177.

[Goo] Googlecreativelab. *googlecreativelab/quickdraw-dataset*. URL: https://github.com/googlecreativelab/quickdraw-dataset.