Generative Adversarial Networks

Jesse Hines

My project was to create a Generative Adversarial Network, using TensorFlow, to generate realistic images. The original plan was to generate images of multiple different types of animals on demand. However, in order to make the project more feasible, I choose to simplify it down to just one category of images. I first tried to generate images of giraffes, but after difficulties getting the model to converge, I changed to generating human faces as there are much better and larger datasets available for faces.

How GANs Work

Generative Adversarial Networks, or GANs for short, are a fascinating concept that allow neural networks to generate new samples that fit some complex distribution, e.g. to create realistic images, sounds, or text. GANs work by having two different neural networks competing against one another in a zero sum game, the "Generator" and the "Discriminator". The Discriminator model tries to discern between fake, generated samples, and real ones from a dataset. Meanwhile, the Generator model tries to create samples that fool the Discriminator. As the two models compete, the Discriminator gets better at recognizing counterfeits, and the Generator gets better at creating samples that look realistic and fool the Discriminator. The eventual result of the game is a Generator that can create realistic appearing samples.

A good analogy, used in the original paper introducing GANs,[1] is that of a counterfeiter and a detective. The counterfeiter tries to make fake bills that fool the detective. Meanwhile, the detective tries to catch all the counterfeits. Both of them will get better at their jobs with practice. We're rooting for the counterfeiter. The Generator model is like the counterfeiter, and Discriminator is like the detective. To implement this in an actual model, you train both neural networks together. In each training step, the Generator takes a large vector of random noise, and creates a batch of images from it. Then the Discriminator analyzes a mixed batch of the generated images and real images from the dataset, marking each as either "real" or "fake". The Discriminator's loss function is based on whether it correctly marked the images. The Generator's loss is calculated based on how many of its fakes got past the Discriminator unnoticed.

GANs are actually a relatively recent development in machine learning, but they have already made a large impact. GANs were first introduced in 2014 by Ian J. Goodfellow and co-authors.[1] In their paper describing GANs, they successfully trained networks on the MNIST handwriting dataset, the Toronto Face Database, and the CIFAR-10 dataset. Since then, there have been numerous other examples of GANs. Some notable ones include thispersondoesnotexist.com, which generates realistic images of faces, and Google's BigGAN, which can generate convincing photos of everything from dogs to pirate ships. Other GANs can generate 3D models, videos, or text.

Since my model would be working with images, I needed to use deep convolutional neural networks. Image processing neural networks are difficult largely because of the sheer size of the images the are supposed to work with. Using a typical "dense" neural network attached to every pixel in an image, with every "neuron" connected to every other, would be infeasible for anything but the smallest of images.

A solution to this problem is convolutional layers. Convolutional layers consist of a relatively small squares of "neurons" or weights that scan across an image[2]. This allows the same weights to be reused across

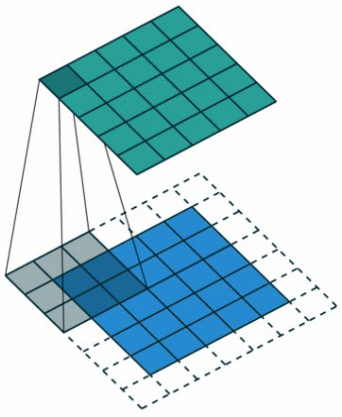

Figure 1: A convolutional layer[2]

the image. Convolutional layers preserve information about spacial locality in the image, which is very important for image recognition. The convolutional layers can be stacked on top of one another, each layer detecting more abstract shapes in the image. For example, the first layer may detect edges and curves, the next layer circles and boxes, and the layers above that detect dogs or bicycles. Convolutional layers have had great success in image recognition algorithms.

Convolutional layers have several parameters. In TensorFlow and Keras, they include stride, filter shape, and number of filters. The filter shape is typically 3x3, 5x5, or 7x7. It is the size of the square that "scans" across the image. In my model, a 5x5 filter gave the best results. Stride is how far the filter moves each iteration. The default is 1, but setting it to higher values will reduce the size of the output layer, which is useful for stacking convolutional layers. You can also have multiple different filters that run in a single convolutional layer.

DCGANs, or Deep Convolutional Generative Adversarial Networks, are GANs that use convolutional layers. In my model, the Discriminator is designed as a normal image recognition network, with convolutional layers stacked on top of one another. The Generator works similarly, but is essentially the reverse of the Discriminator. The Generator uses transposed convolutional layers which, instead of scanning an image to create output, scan over the output to create an image. The transposed convolutional layers can also be stacked.

Hardware

Hardware has been a significant problem in training my models. TensorFlow requires good GPUs to run efficiently. And currently, TensorFlow doesn't support any GPUs other than NVIDIA. My laptop has an integrated Intel GPU which wouldn't have been very powerful in any case, and TensorFlow wouldn't use it at all. Training even a simple tutorial GAN on the MNIST handwriting dataset was taking hours.

However, I found that Google Colab will give you free access to a powerful NVIDIA Tesla K80 GPU. Colab was designed with machine learning (particularly TensorFlow) in mind. Using Colab, I was able to train a model completely on the MNIST handwriting dataset before my machine had even finished one epoch. This greatly improved productivity for the project. Colab does have some limitations. I was limited to 12 hour sessions when using a GPU, and sometimes they would cut me off earlier if they were busy, since its shared hardware. Also, Colab gives users a non-persistent virtual machine, so any state has to be saved to a personal machine or to Google Drive between sessions. To use Colab for training, I had to design my models to be able to interrupt and restart training later. I trained my models for this project almost entirely on Colab, using many hours of GPU accelerated training time.

Giraffes

My original plan was to generate several "categories" of animals on demand, similar to BigGAN. However, that turned out to be a bit out of scope for this project, so I decided to narrow the problem down to a single category. After selecting a category at random from the image databases I was looking into, I started trying to teach the computer to paint a giraffe.

I collected photos from a couple different databases, and combined all of their images of giraffes. Between the COCO (Common Objects in Context)[3], Visual Genome[4], and OpenImages[5] datasets, I collected around 6,000 images of giraffes. All three of the datasets included bounding box information, so I was able to crop the images around the giraffe, and resize all of the images to 128x128 pixels.

I designed my model using the TensorFlow Keras API. The Discriminator consisted of a repeating sequence of a Conv2D layer followed by a Dropout layer, four deep. A small Dense layer lay

on top, connected to the output. The Generator began with a dense layer, attached to the noise vector input, and then a repeating sequence of Conv2DTranspose, BatchNorm, and LeakRelu layers, also four deep. One extra Conv2DTranspose layer reshaped the output to match the expected image size of 3-channel, 128x128 pixels.

After about 50 epochs, my first attempt at giraffes was a complete failure. The images it generated were little more than random noise.

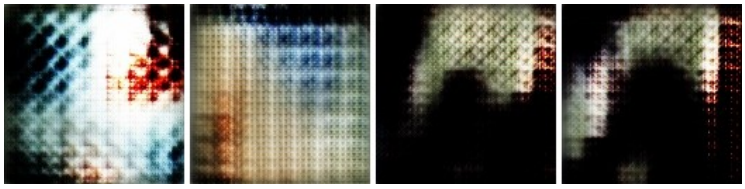

Figure 2: Samples from giraffe model version 1

My next model was somewhat more successful. I fixed a problem where I had the filters distributed in a backwards fashion. Normally you have move filters on convolutional layers "higher up" in the model[6]. Since the higher convolutional layers are smaller, having more filters means that the computation spent on each level of the model should be roughly equivalent. I had intended to set the model up this way. However, I had accidentally set the discriminator up backwards, having more filters in the larger lower layers. This caused ridiculous ram usage in the first model, which slowed things down, and actually crashed the first model once or twice.

In the second attempt I also followed some advise from two articles by Jason Brownlee[7,8]. I added label smoothing, which bends the real or fake labels, 1 and -1 respectively, shown to the Discriminator slightly towards zero. This is supposed to have a regularizing effect. I changed the model to use a Gaussian distribution for the initial random assignment of all the weights. I also changed the hyper-parameters of the Adam optimizers for both models, increasing the learning rate from 1e-4 to 2e-4 and decreasing the beta_1 from the default 0.9 to 0.5.

However, after running the model for a few hundred epochs, the generated images were still very poor. It did improve over the first attempt somewhat. A few images could be interpreted as giraffes if you have adequate imagination. It generated quite a large quantity of unusual, but at least interesting, images.

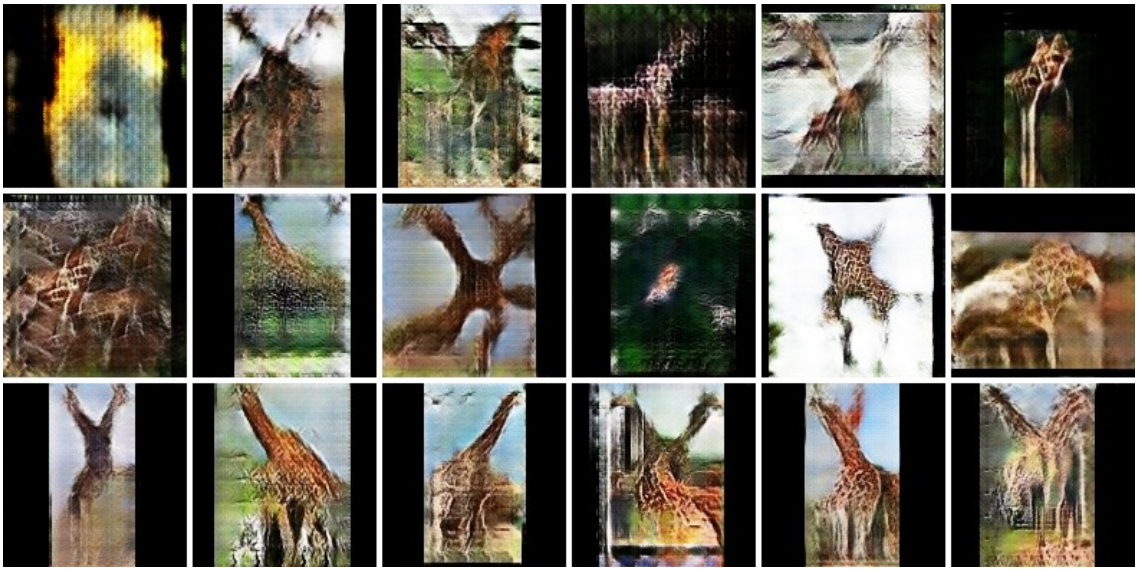

Figure 3: Samples from giraffe model version 2

One of the problems in the model was that the Discriminator was overpowering the Generator. This is a common problem in GANs, called the "vanishing gradient"[9]. The Discriminator has an easier job, and so generally out preforms the Generator. When the Discriminator is correctly marking nearly all of the Generator's images as fakes, the Generator no longer has anything to learn off of since nothing it does succeeded at all. The gradient to apply to the model is nearly zero, and the model stops learning. The Discriminator in my model was correctly flagging 99.9% or more of the Generator's images by the end. At this point, the Generator does not seem to be able to learn any further, and may even digress. The model didn't converge.

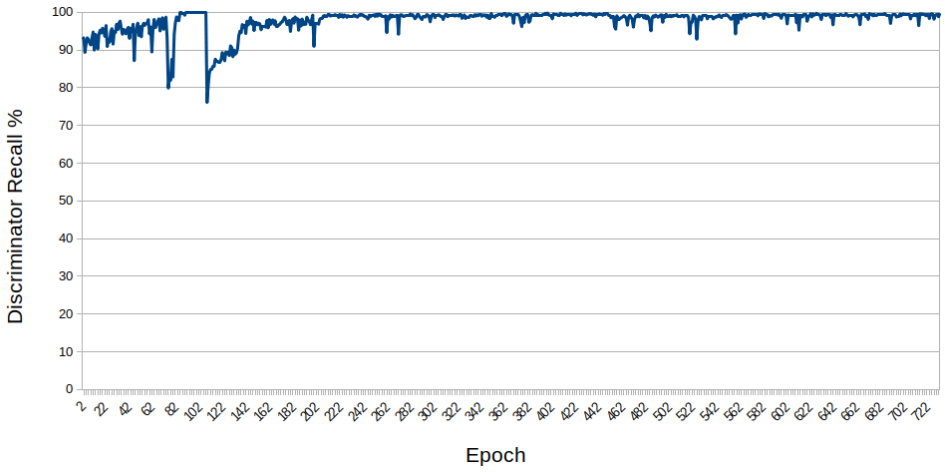

Figure 4: Giraffe model version 2 Discriminator recall over the last 700 epochs

There are multiple ways to try and solve the vanishing gradient problem. One way is to add deliberate noise to the labels shown to the Discriminator, in order to slow it down and regularize it.[7] I also tried a dynamic training schedule which would train the discriminator less when the accuracy got above a threshold. However, none of these strategies helped in my situation, if anything they made things worse.

Faces

The root of the problem was the dataset I was using. A mere 6,000 images is a small amount to train with. Also, a giraffe is a complicated object, and the photos I had were from many different angles and perspectives. In addition, after some more research, I found that GANs seem to have difficulties generating good images over about 64x64 pixels without using advanced techniques. I had been trying to generate 128x128 pixel images.

So, I decided to switch datasets. My next attempt used the CelebA dataset, which is a very large database of human faces. It contains over 200,000 images, vs my 6,000 giraffes. Also, the images are much more uniform in nature, all being faces in about the same pose, pre-cropped and zoomed. I also lowered the resolution to 32x32.

Using the same neural network architecture, just scaled down to the lower resolution, I had much better success on the CelebA dataset. Even after 1 epoch, and just 30 min of training, it was already generating recognizable faces. I ran it for a total of 12 epochs and 8 hours.

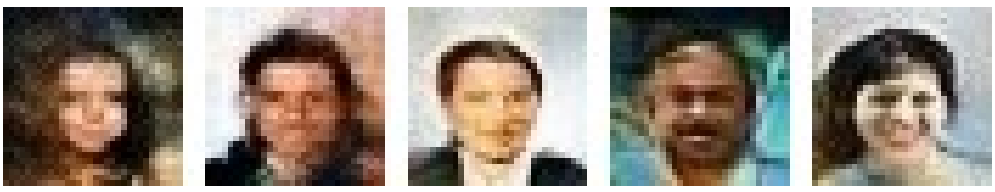

Figure 5: Samples from CelebA 32x32 model

With that success, I decided to try to generate larger images. At only 32x32 pixels, even the real dataset's images are rather difficult to recognize. After increasing the size of the model to take 64x64 pixel images, I ran the new model for about 40 epochs. This version of the model generated recognizable faces as well, but still quite unnatural ones. It also had the same vanishing gradient issue that I had with the giraffes, the Discriminator quickly reached greater than 99% accuracy and the Generator was not improving further.

Figure 6: Samples from CelebA 64x64 model version 1

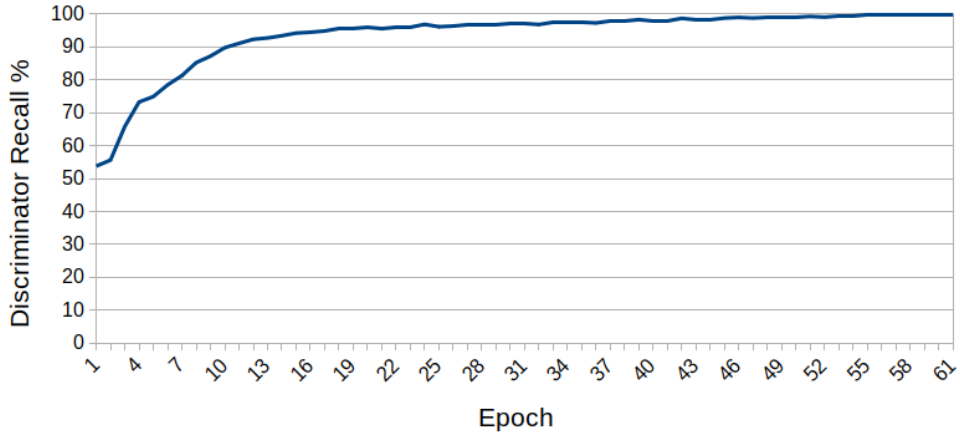

Figure 7: CelebA 64x64 model version 1 Discriminator recall

I decided to make one more attempt. I increased the size of the convolutional filters from 3x3 to 5x5, and chopped the Discriminator's (but not Generator's) learning rate by a factor of 5. I hoped that this would balance the two networks better, slowing the Discriminator's progress down, and allowing the Generator time to keep up. The learning was clearly slower, taking a few more epochs to generate faces comparable to the faces of the previous attempt, but the training seemed more stable. I was able to train the model much longer before hitting the vanishing gradient problem. This model generated the best images by far.

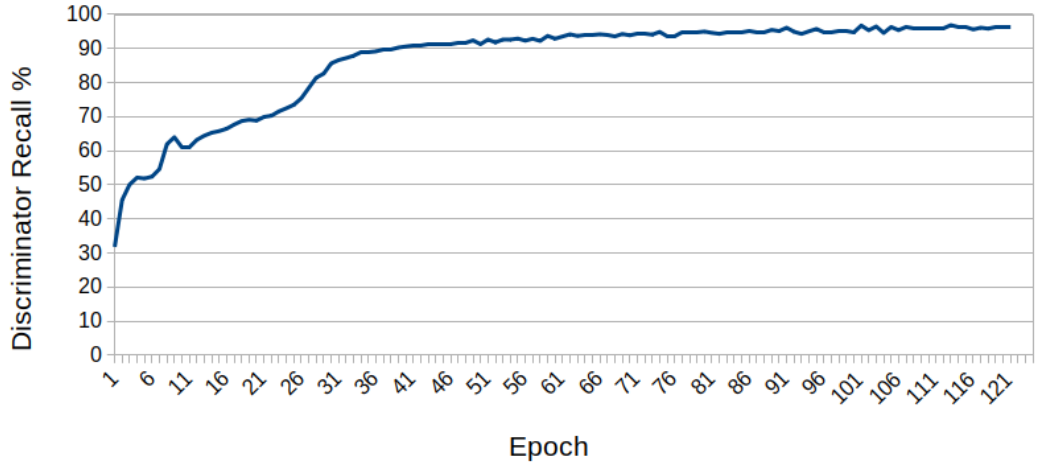

Figure 8: CelebA 64x64 model version 2 Discriminator recall

Figure 9: Samples from CelebA 64x64 model version 2

Potential Improvements

In order to train a GAN to generate larger images there are a couple techniques I found, but did not have time to implement. Regular DCGANs, such as my model, excel at textures, but have trouble capturing overall structure. BigGAN used something called self-attentive learning, which allows the models to focus on individual portions of the image as a group[10]. This enables it to more accurately render the large-scale structures of object. Another technique is progressive growing, in which you first train the GAN on very low resolution images, and then progressively add layers to the model and train it on higher and higher resolution images[11].

GANs are a fascinating idea, and have a lot of powerful potential. Training them successfully is quite difficult, and like most machine learning, very dependent on a good dataset. While I did not achieve my initial goal of giraffes images, I did get an AI to generate decent images of human faces. I still would like to try increasing the resolution of the images further sometime, and perhaps do some more experiments using Generative Adversarial Networks.

1. Goodfellow, Ian J, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. “Generative Adversarial Nets.” Accessed November 24, 2019. https://arxiv.org/pdf/1406.2661.pdf.
2. Saha, Sumit. “A Comprehensive Guide to Convolutional Neural Networks - the ELI5 Way.” Medium. Towards Data Science, December 17, 2018. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.
3. See http://cocodataset.org/ for more information about the COCO dataset.
4. See http://visualgenome.org/ for more information about Visual Genome.
5. See https://storage.googleapis.com/openimages/web/index.html form more information about OpenImages.
6. Ramesh, Shashank. “A Guide to an Efficient Way to Build Neural Network Architectures- Part II: Hyper-Parameter...” Medium. Towards Data Science, May 26, 2018. https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7.
7. Brownlee, Jason. “How to Implement GAN Hacks in Keras to Train Stable Models.” Machine Learning Mastery, July 12, 2019. https://machinelearningmastery.com/how-to-code-generative-adversarial-network-hacks/.
8. Brownlee, Jason. “Tips for Training Stable Generative Adversarial Networks.” Machine Learning Mastery, September 11, 2019. https://machinelearningmastery.com/how-to-train-stable-generative-adversarial-networks/.
9. Hui, Jonathan. “GAN - Why It Is so Hard to Train Generative Adversarial Networks!” Medium. Medium, October 29, 2019. https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b.
10. Chen, Sherwin. “Techniques in Self-Attention Generative Adversarial Networks.” Medium. Towards AI, July 19, 2019. https://medium.com/towards-artificial-intelligence/techniques-in-self-attention-generative-adversarial-networks-22f735b22dfb.
11. Brownlee, Jason. “How to Train a Progressive Growing GAN in Keras for Synthesizing Faces.” Machine Learning Mastery, October 3, 2019. https://machinelearningmastery.com/how-to-train-a-progressive-growing-gan-in-keras-for-synthesizing-faces/.