Intro to GANs with Potatoes

Aaron Stopher

Ritchie School of Engineering and Computer Science, University of Denver

COMP 4449: Capstone

Prof. Claudio Delrieux

10/19/2022

Abstract

The purpose of this project is to introduce the concepts behind basic image generation using a simple implementation of Deep Convolutional Generative Adversarial Networks. This project will detail a basic example, training DCGANs to reproduce images of a single class. Exploring the concepts related to how GANs work in general.

Introduction to DCGANs for image generation with Potatoes

1 Introduction

A Generative Adversarial Network (GAN) is at its core a coupling of neural networks which is used to generate predictions within the same distribution as the input data from random noise. To do this we will use an Encoder-Decoder network configuration.

Understanding GANs provides a good basis for understanding more modern generative methods like Diffusion Models. There are many uses for this including synthesizing data to supplement missing data within an underrepresented class for classification**.**

1.1 Data

The Potato Image Dataset (TPID): The included dataset consists of red (600), red washed (604), white (600) , and sweet potatoes (600). It contains 2,404 color images in total. Image parameters are 64 x 64 x 3 (RGB). This set has been manually derived from the following dataset: https://www.kaggle.com/datasets/moltean/fruits?select=fruits-360_dataset

1.2 Data Preparation

Resize to 64 x 64, center crop, then normalize. Scaling each channel of the input image is defined as: **channel_value = (channel_value - mean) / std** in other words the range for our input data would be **((0-0.5)/0.5) = -1** to **((1-0.5)/0.5) = 1**

2 Model Architecture

Input images are fed into a Discriminator network (CNN), these images will be classified as real images. Simultaneously output images from the Generator network will be generated from static noise then fed into **D** these will be classified as fake images. For each epoch of training we will update **G** growing closer to convergence for the distribution of our input data.

[2] *"Theoretically, the model could learn to memorize training examples, but this is experimentally unlikely as we train with a small learning rate and minibatch SGD. We are aware of no prior empirical evidence demonstrating memorization with SGD and a small learning rate."*

[2] All model weights should be randomly initialized from a normal distribution with **mean=0, stdev=0.02**. The **weights_init** function provided by PyTorch takes an initialized model as input and reinitializes all convolutional, convolutional-transpose, and batch normalization layers to meet this criteria.

2.1 Generator

The Generator starts with a noise vector z. We then reshape the input vector into a three-dimensional hidden layer with a small base (width x height) and large depth (channels). Using transposed convolutions, the

input is progressively reshaped such that its base grows while its depth decreases until we reach the shape of the image we are seeking to synthesize, 64 × 64 × 3 in this case.

2.2 Transpose Convolutions

Transposed Convolutions are used to upsample the input feature map to a desired output feature map. [1] This type of convolution is also referred to as a fractionally strided convolution, this is because the stride over the output is equivalent to the fractional stride over the input. For example, a stride of 2 over the output is 1/2 stride over the input. Thus, we have a nice relationship with this method for upscaling our noise to the same dimensions as our input image. Note that this is sometimes wrongly referred to as a deconvolution, however this operation is an upscaling method not a negation of downsampling convolutions.

2.3 Discriminator

Between each layer we will place a LeakyReLu function (0.2) for expressing our neurons, utilizing dropout of 0.2; proper dropout essentially simulates training large numbers of architectures simultaneously. Importantly, dropout can drastically reduce the chance of overfitting during training. Finally, we use Sigmoid to output for our loss function. Note the lack of pooling seen in a typical CNN.

[2] *"Core to our approach is adopting and modifying three recently demonstrated changes to CNN architectures. The first is the all convolutional net (Springenberg et al., 2014) which replaces deterministic spatial pooling functions (such as maxpooling) with strided convolutions, allowing the network to learn its own spatial downsampling. We use this approach in our generator, allowing it to learn its own spatial upsampling, and discriminator."*

3 Details of GAN training

Given the function: **Ex~pdata(x)[log D(x)] + Ez~pz(z)[log(1 − D(G(z)))]**. Where the Generator tries to minimize the function and the Discriminator tries to maximize it.

[2] *"In practice, **log(D(x)) + log(1 - D(G(z)))** may not provide sufficient gradient for **G** to learn well. Early in learning, when **G** is poor, **D** can reject samples with high confidence because they are clearly different from the training data. In this case, **log(1 − D(G(z)))** saturates. Rather than training **G** to minimize **log(1 − D(G(z)))** we can train **G** to maximize **log D(G(z))**. This objective function results in the same fixed point of the dynamics of **G** and **D** but provides much stronger gradients early in learning."*

[2] *"Optimizing **D** to completion in the inner loop of training is computationally prohibitive, and on finite datasets would result in overfitting.*
*Instead, we alternate between **k** steps of optimizing **D** and one step of optimizing **G**.*
*This results in **D** being maintained near its optimal solution, so long as **G** changes slowly enough."*

3.1 Loss

Binary Cross Entropy is an intuitive selection for our loss since we need to determine if the image is real(1) or fake(0). It compares each of the predicted probabilities to actual class output, this can be either 0 or 1.

It then calculates the score which penalizes the probabilities based on the distance from the expected value. This will inform our activation function choice (Sigmoid) for the output layer in our Discriminator network as our range should be between 0 and 1.

3.2 Label Smoothing

[3] *"Label smoothing, a technique from the 1980s recently independently re-discovered by Szegedy et. al, replaces the* ***0*** *and* ***1*** *targets for a classifier with smoothed values, like* ***.9*** *or* ***.1****, and was recently shown to reduce the vulnerability of neural networks to adversarial examples. Replacing positive classification targets with α and negative targets with* ***β****, the optimal discriminator becomes* ***D(x) = α pdata(x) + β pmodel (x) / pdata(x) + pmodel(x)****. The presence of* ***pmodel*** *in the numerator is problematic because, in areas where* ***pdata*** *is approximately zero and pmodel is large, erroneous samples from pmodel have no incentive to move nearer to the data. We therefore smooth only the positive labels to* ***α****, leaving negative labels set to* ***0****."*

3.3 Limitations of GANs

Model parameters oscillate and never truly converge. The Generator can collapse producing limited varieties of samples. Highly sensitive to the hyperparameter selections. The Discriminator can become extremely successful such that the Generator gradient disappears. Checkerboarding can result from transpose convolutions.

4 Model Evaluation

Qualitative criteria for a successful generator, i.e. it should generate images with:

- good quality. High fidelity and "realistic"
- diversity / variety. A good representation of the training images different categories

Quantitative methods for determining closeness in distribution:

- Inception Score (IS)
- Fréchet Inception Distance (FID)

4.1 Qualitative

Image quality: The final generated images do begin to resemble our training images with some anomalies around the edges of our generated potatoes. Coloring towards the center of the real potatoes appears to be reflected in the generated samples relatively well, however some checkerboarding patterns from the transpose convolutions have been produced.

Diversity: Generated sample variety appears to represent our training set very well.

4.2 Quantitative

For the sake of time and simplicity I choose to not implement either IS or FID scores for quantitative evaluation. Part of the reason for this includes the simplicity of the dataset which was purposefully picked to match the explanatory goal of the project. I encourage the curious to explore quantitative evaluation methods of your choosing with included pre-trained generative models.

References

[1] Goodfellow, Ian J. and Pouget-Abadie, Jean and Mirza, Mehdi and Xu, Bing and Warde-Farley, David and Ozair, Sherjil and Courville, Aaron and Bengio, Yoshua. (2014). “Generative adversarial networks”. *arXiv:1406.2661*. https://arxiv.org/abs/1406.2661

[2] A. Radford, L. Metz, and S. Chintala. (2015). “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks,” *arXiv:1511.06434v2*. https://arxiv.org/abs/1511.06434

[3] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen. (2016). “Improved Techniques for Training GANs” *arXiv:1606.03498v1*. https://arxiv.org/abs/1606.03498