Generating Synthetic Images with ProGAN

Anonymous author

Abstract

This paper proposes using a Progressive Growing GAN (ProGAN) to generate 64x64 images using the aligned and cropped CELEB-A [3] dataset.

1 Methodology

Generative Adversarial Networks (GANs) are a popular tool used to generate novel synthetic images that look realistic. In a GAN setting, two neural networks and compete against each other. The aim of the generator is to generate data by mapping latent-space points to a high dimensional distribution [6] such that the discriminator, whose job is to distinguish real data from fake, is fooled into thinking the data produced by the generator is real data. Thus, the aim of the generator is to minimise and for the discriminator to maximise as follows:

$$\min_G \max_D \mathbb{E}_{x\sim\mathbb{P}_r}[log(D(x))] + E_{\tilde{x}\sim\mathbb{P}_g}[log(1 - D(\tilde{x}))] \tag{1}$$

where x denotes the data points sampled from the real data distribution $\mathbb{P}_r$, $\tilde{x}$ is $G(z)$ where z is random noise sampled from a noise distribution and $\mathbb{P}_g$ is the model distribution [1].

This paper implements a Progressively Growing GAN(ProGAN) [2] which trains the generator and discriminator on progressively increasing image sizes whilst gradually increasing the number of layers. The key components of this GAN architecture are as follows:

Progressive growing architecture. The training of GANs tend to be unstable for larger images. To address this, the authors of [2] propose a method of starting with low-resolution images e.g. 4x4 and progressively increasing the number of layers of both the generator and discriminator networks simultaneously to allow the training to discover the global structure of the image distribution first before focusing on finer detail [2] (see Table 1 for implementation details). This technique allows the training to be more stable as smaller images have fewer modes and class information and the authors of [2] also claim that this makes the training 2-6 times faster.

Smooth addition of layers. To keep the training stable when new layers are added onto the current architecture of both networks, the authors of [2] propose adding the layers as a residual skip connection block where the output of the newly added layer is controlled by a parameter α which linearly increases from 0 to 1 as training progresses and the output of the previous layers is controlled by 1-α (see Figure 1). The generator combines the two resolutions to generate the final image and the discriminator interpolates between the two resolutions during a resolution transition [2].

Normal distribution weight initialisation. The authors of [2] implement a normal distribution weight initialisation $x \sim N(0, 1)$. These weights w_i are then dynamically scaled at runtime such that:

$$\hat{w} = w_i/c \tag{2}$$

where c is He's initialisation constant.

The authors of [2] argue that dynamically scaling the weights has the advantage of ensuring the learning speed remains the same for all weights when adaptive stochastic gradient descent optimisers such as Adam are used.

Generator	Act.	Output shape
Latent Vector	-	512 x 1 x 1
Conv 4 x 4	LReLU	512 x 4 x 4
Conv 3 x 3	LReLU	512 x 4 x 4
Upsample	-	512 x 8 x 8
Conv 3 x 3	LReLU	512 x 8 x 8
Conv 3 x 3	LReLU	512 x 8 x 8
Upsample	-	512 x 16 x 16
Conv 3 x 3	LReLU	512 x 16 x 16
Conv 3 x 3	LReLU	512 x 16 x 16
Upsample	-	512 x 32 x 32
Conv 3 x 3	LReLU	512 x 32 x 32
Conv 3 x 3	LReLU	512 x 32 x 32
Upsample	-	512 x 64 x 64
Conv 3 x 3	LReLU	256 x 64 x 64
Conv 3 x 3	LReLU	256 x 64 x 64
Conv 1 x 1	linear	3 x 64 x 64

Discriminator	Act.	Output shape
Input image	-	3 x 64 x 64
Conv 1 x 1	LReLU	256 x 64 x 64
Conv 3 x 3	LReLU	256 x 64 x 64
Conv 3 x 3	LReLU	512 x 64 x 64
Downsample	-	512 x 32 x 32
Conv 3 x 3	LReLU	512 x 32 x 32
Conv 3 x 3	LReLU	512 x 32 x 32
Downsample	-	512 x 16 x 16
Conv 3 x 3	LReLU	512 x 16 x 16
Conv 3 x 3	LReLU	512 x 16 x 16
Downsample	-	512 x 8 x 8
Conv 3 x 3	LReLU	512 x 8 x 8
Conv 3 x 3	LReLU	512 x 8 x 8
Downsample	-	512 x 4 x 4
Minibatch stddev	-	513 x 4 x 4
Conv 3 x 3	LReLU	512 x 4 x 4
Conv 4 x 4	LReLU	512 x 1 x 1
Fully-connected	linear	1 x 1 x 1

Table 1: Generator and discriminator ProGAN architecture used to generate 64x64 images. Inspired by [2].

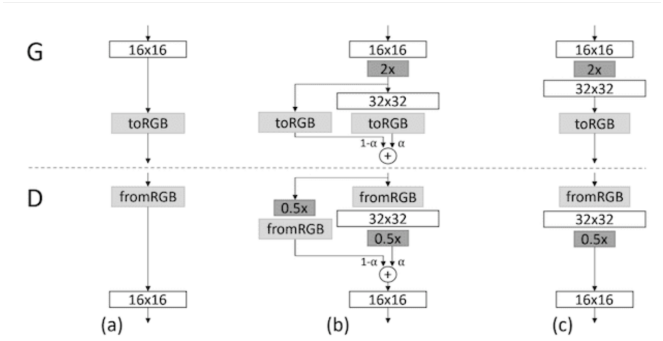

Figure 1: Example of new layer acting as a residual connection when transitioning from 16x16 to 32x32 images. Image from [2].

Pixel-wise feature vector normalisation. To prevent uncontrolled upswings of signal magnitudes in the generator and discriminator networks, feature vectors are normalised to unit length [2] after each convolution layer on a per pixel level:

$$b_{x,y} = \frac{a_{x,y}}{\sqrt{\frac{1}{N}\sum_{j=0}^{N-1}(a_{x,y}^{j})^2 + \epsilon}} \tag{3}$$

where $\epsilon = 10^{-8}$, N is the number of feature maps, $a_{x,y}$ denotes the normalised feature vectors in pixel x and $b_{x,y}$ denotes the normalised feature vectors in pixel y.

Mini-batch standard deviation. Additionally, the authors of [2] suggests a simplified approach to minibatch discrimination originally proposed by [5] to prevent the GAN from capturing only a subset of the variation found in training data, i.e. mode collapse. The proposed method involves calculating the standard deviation individually for each feature across the minibatch and averaging them. This is then concatenated as an additional channel and and was found to yield the best results when added to the output layer.

Improved Wasserstein Loss. The loss function of WGAN-GP, known as the improved Wasserstein loss, was used for ProGAN.

Unlike other GANs, the WGAN discriminator (also known as the critic) returns a range of values instead of a 0 or 1 for fake and real data respectively. It bases its loss on an approximation of Wasserstein distance as opposed to Jensen-Shannon divergence used in other basic GANs to prevent unstable training. The Kantorovich-Rubinstein duality is used to form the following value function[1]:

$$\min_{G} \max_{D \in \boldsymbol{D}} \mathbb{E}_{x \sim \mathbb{P}_r}[D(x)] - E_{\tilde{x} \sim \mathbb{P}_g}[D(\tilde{x})] \tag{4}$$

where $\boldsymbol{D}$ is the set of 1-Lipschitz functions.

Although the authors of WGAN proposed weight clipping to enforce the Lipschitz constraint on the critic, this leads to several issues such as exploding and vanishing gradients without precise hyperparameter tuning of the clipping threshold.

The authors of [1] suggest the use of the following gradient penalty instead of clipping as a solution to the problems and to improve training speed and sample quality:

$$\boldsymbol{GP} = \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}}[(\left\|{}_{\hat{x}} D(\hat{x})\right\|_2 - 1)^2] \tag{5}$$

Thus the loss function of the discriminator in the WGAN-GP architecture can be expressed as:

$$L = E_{\tilde{x} \sim \mathbb{P}_g}[D(\tilde{x})] - \mathbb{E}_{x \sim \mathbb{P}_r}[D(x)] + \lambda \boldsymbol{GP} \tag{6}$$

An additional term with a small weight $\epsilon_{drift} = 0.001$ to keep the discriminator close to 0 is added to the Wasserstein loss to form the loss function for the discriminator in the ProGAN setup:

$$L^{'} = L + \epsilon_{drift} \mathbb{E}_{x \in \mathbb{P}_r}[D(x)^2] \tag{7}$$

2 Hyperparameters

Images were trained starting from 4x4 to 64x64 rescaled images of the aligned and cropped CELEB-A dataset in the following size increments: 4x4, 8x8, 16x16, 32x32 and 64x64. Batch sizes of 512, 256, 128, 64 and 30 were used for the sizes mentioned previously, starting from 4x4. The batch sizes were incrementally decreased to account for the higher memory consumption resulting from processing larger images. It was found that increasing batch sizes improved the quality of images. Approximately 2 million images were used at each size to train both networks. The Adam optimiser was used with $\alpha = 0.001$, $\beta_1 = 0$, $\beta_2 = 0.99$ as mentioned in [2], with a learning rate of $1\text{x}10^{-3}$ as per the implementation of [4]. The generator and discriminator was optimised on a per mini-batch basis [2], unlike the authors of WGAN-GP [1] who suggest a greater number of iterations of the discriminator optimisation step. All other parameters used for the implementation presented in this paper used the same parameters [2] used for generating 1024x1024 images with the CELEBA-HQ dataset.

3 Results

The model is able to identify key features that make up a human face and accurately position them in the majority of cases. Unsurprisingly, some random samples appear distorted, which could potentially improve with more training.

The next results show a non-cherry-picked sample of 64 64x64 images generated by the model (Figure 2), as well as images generated by interpolating between cherry-picked points in the latent space (Figure 3):

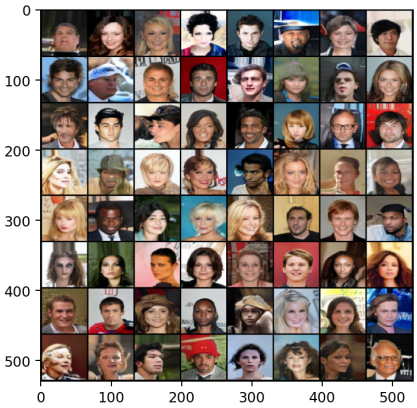

Figure 2: Random batch of 64 images

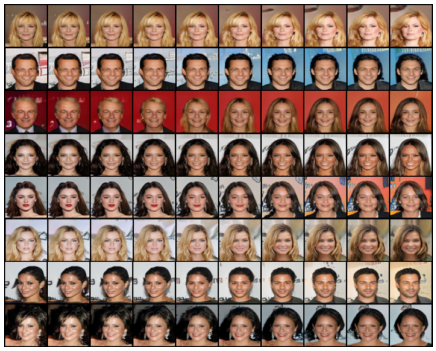

Figure 3: Latent interpolation between 8 pairs of cherry-picked image samples

And here are some cherry-picked samples that show the best outputs the model has generated:

4 Limitations

Although the ProGAN implementation architecture accommodates for training of larger image sizes including 128x128, this paper did not train on such images due to the lack of computational resources and time required (possibly over a week of training for meaningful output). Training for a longer period of time on larger images will generate more realistic samples.

Bonuses

This submission has a total bonus of -4 marks (a penalty), as it is trained on the CELEB-A dataset resized to 64x64 images (+4), but uses a GAN (-8).

References

[1] Ishaan Gulrajani et al. "Improved Training of Wasserstein GANs". In: *CoRR* abs/1704.00028 (2017). arXiv: 1704.00028. URL: http://arxiv.org/abs/1704.00028.

[2] Tero Karras et al. "Progressive Growing of GANs for Improved Quality, Stability, and Variation". In: *CoRR* abs/1710.10196 (2017). arXiv: 1710.10196. URL: http://arxiv.org/abs/1710.10196.

[3] Ziwei Liu et al. "Deep Learning Face Attributes in the Wild". In: *Proceedings of International Conference on Computer Vision (ICCV)*. Dec. 2015.

[4] Aladdin Persson. *Machine-learning-collection/model.py at master · Aladdinpersson/machine-learning-collection*. URL: https://github.com/aladdinpersson/Machine-Learning-Collection/blob/master/ML/Pytorch/GANs/ProGAN/model.py.

[5] Tim Salimans et al. "Improved Techniques for Training GANs". In: *CoRR* abs/1606.03498 (2016). arXiv: 1606.03498. URL: http://arxiv.org/abs/1606.03498.

[6] Rajhans Singh et al. “Non-Parametric Priors For Generative Adversarial Networks”. In: *CoRR* abs/1905.07061 (2019). arXiv: 1905.07061. URL: http://arxiv.org/abs/1905.07061.