Generating Unique, Diverse and Higher Resolution Images using a ProGAN Model

Hashim Hussain

Abstract

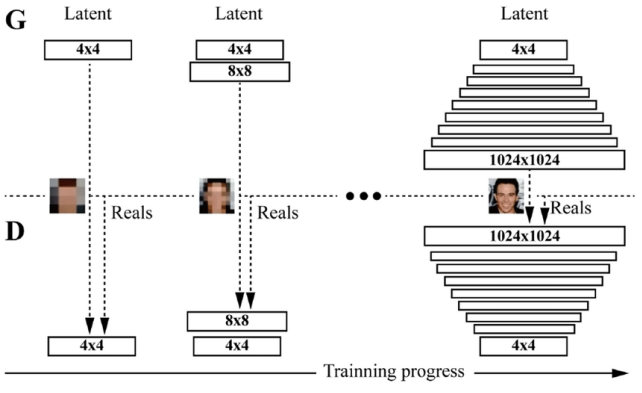
This paper proposes using a general adversarial model (GAN), in the form of a ProGAN, to generate unique and diverse 32x32 and 128x128 pixel images of cars.

1 Methodology

A GAN uses unsupervised learning to learn the underlying patterns of input data, allowing the creation of unique outputs which closely resemble the original input. This is performed using an adversarial system, where a generator creates fake items and claims them to be part of the input and a discriminator evaluates the falsity of each claim.

$$\min_G \max_D \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[1 - \log D(G(z))] \tag{1}$$

In this instance, the two networks improve one another, through a min-max game (1), to produce new images that can pass being a part of the input data. Two multi-layer perceptrons, for both a generator (G) and a discriminator (D), are created and trained together, as outlined in [1]. D attempts to maximise this objective function and identify all the fake generated images, whereas G attempts to minimise the function and fool D. Initially random noise is generated, allowing the discriminator to create ground truth labels for fake and real images.



[4]

Within this paper, a GAN was implemented, in the form of a progressive growing of GAN (ProGAN) model, to produce 32x32 pixel images when trained with CIFAR-10 and 128x128 pixel images when trained with STL-10.

The ProGAN model extends the GAN architecture [7] by slowly growing the generator and discriminator through incrementing the resolution of the generated images, as seen in the image above.

An issue with simple GANs is that the discriminator and generator must be synchronized well with one another. G can collapse if it is trained too much without changing D; similarly, D may reach its optimal state too fast, causing G to be trained improperly. Hence, ProGAN was chosen as it avoids this issue by slowly increasing the size of the model, allowing G and

D to work efficiently and in a stable manner with one another when training. Consequently, ProGAN produces higher quality images compared to simple GANs.

When compared to other models, it is clear that the ProGAN model is a sophisticated one, as it has a low FID score of 15.52 [3], meaning that it does well in producing 'real' images, which are diverse due to good mode coverage. This is since ProGAN increases the variation of the generated images by using mini-batch standard deviation [7] to try to capture more of the data rather than a subset. Additionally, ProGAN is strong when it comes to training as it produces outputs quickly. It is able to learn the generic features of images first, followed by the fine details as the image resolution increases.

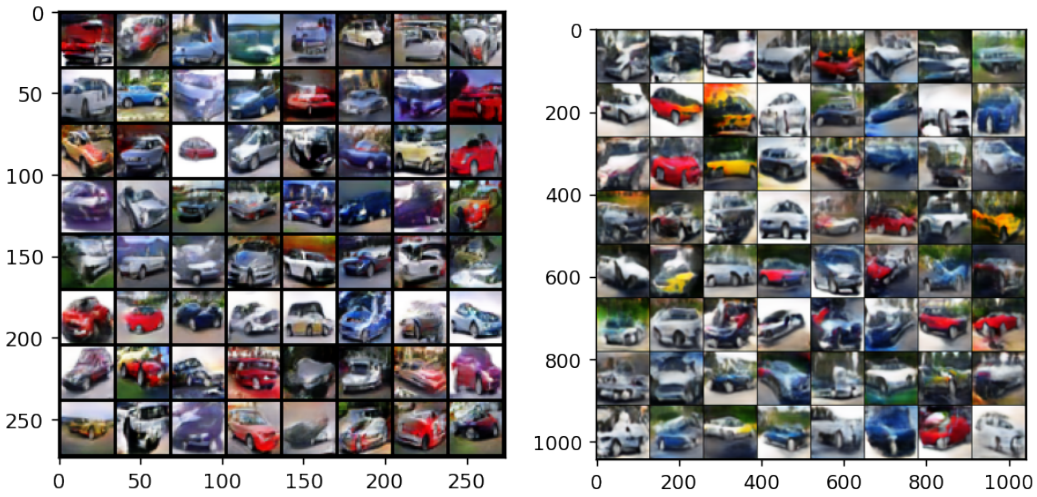
The implementation of this model was adapted from [8, 5], which follows closely to the original paper [7], after a simple GAN was initially implemented [2]. The subset of cars was chosen within these datasets, to allow the model to focus training on one image.

Discriminator	Act.	Output shape	Params
Input image	–	3 × 1024 × 1024	–
Conv 1 × 1	LReLU	16 × 1024 × 1024	64
Conv 3 × 3	LReLU	16 × 1024 × 1024	2.3k
Conv 3 × 3	LReLU	32 × 1024 × 1024	4.6k
Downsample	–	32 × 512 × 512	–
Conv 3 × 3	LReLU	32 × 512 × 512	9.2k
Conv 3 × 3	LReLU	64 × 512 × 512	18k
Downsample	–	64 × 256 × 256	–
Conv 3 × 3	LReLU	64 × 256 × 256	37k
Conv 3 × 3	LReLU	128 × 256 × 256	74k
Downsample	–	128 × 128 × 128	–
Conv 3 × 3	LReLU	128 × 128 × 128	148k
Conv 3 × 3	LReLU	256 × 128 × 128	295k
Downsample	–	256 × 64 × 64	–
Conv 3 × 3	LReLU	256 × 64 × 64	590k
Conv 3 × 3	LReLU	512 × 64 × 64	1.2M
Downsample	–	512 × 32 × 32	–
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Downsample	–	512 × 16 × 16	–
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Downsample	–	512 × 8 × 8	–
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Downsample	–	512 × 4 × 4	–
Minibatch stddev	–	513 × 4 × 4	–
Conv 3 × 3	LReLU	512 × 4 × 4	2.4M
Conv 4 × 4	LReLU	512 × 1 × 1	4.2M
Fully-connected	linear	1 × 1 × 1	513
Total trainable parameters			**23.1M**

[4]

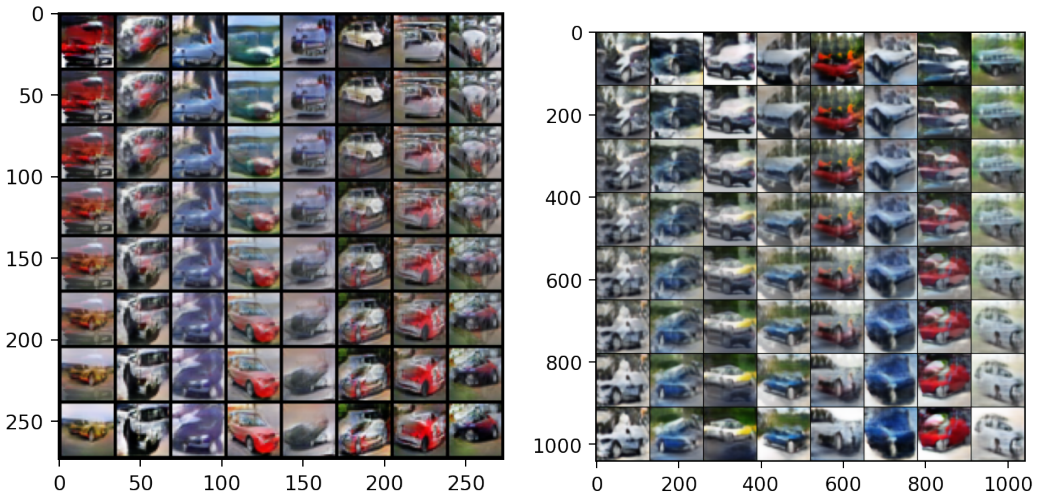
To implement the GAN, the hyperparameters were initially set, including a batch size of 32, 200 epochs for each iteration and a learning rate of 1e-3. The CIFAR-10 and STL-10 datasets were loaded, ensuring that the cars subset was selected and that the images could be increased at each iteration.

The ProGAN model architecture, as seen above, was then implemented, which fades in blocks of layers depending on the image size, and passes on weights from one image resolution to the next to ensure rigorous training. The model also projects feature vectors to and from RGB, for G and D respectively, and normalises the weights. Before training and outputting images, a gradient penalty was defined and applied, which prevents the critic's output gradients from becoming too large or small.



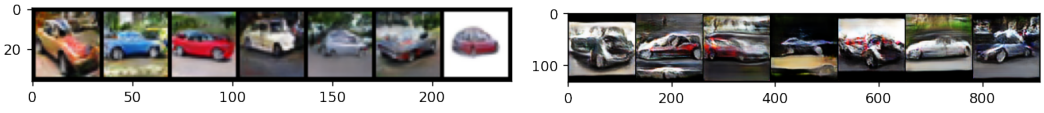
(a) Random CIFAR-10 samples (32x32). (b) Random STL-10 samples (128x128).

Figure 1: Generated random outputs of cars.



(a) CIFAR-10 interpolation. (b) STL-10 interpolation.

Figure 2: Interpolation of car outputs.



(a) Cherry-picked CIFAR-10 outputs. (b) Cherry-picked STL-10 outputs.

Figure 3: Cherry-picked images.

2 Results

As seen in Fig 1, the generated images appear to be diverse and unique in nature and resemble a car in almost every cases. The images appear to be rather clear in both samples. The images have moderately realistic shapes and textures, however these could be better. The lower resolution CIFAR-10 generated images appear to be slightly better than the STL-10 images.

Fig 2 shows images generated by interpolating between points in the latent space. The cars appear to be close to their nearest neighbours, as they are interpolated smoothly for both cases, yet appear to be diverse across the different columns.

For the cherry-picked examples in Fig 3, the cars look more realistic, and a many of them could be 'real'.

3 Limitations

In general, the results appear to be of a high quality and diverse, however they could be even more realistic. In terms of physical limitations, this model could have been trained for longer or run on a system with more RAM and a better GPU. In terms of the limitations with the model, this could have been extended to Style-GAN2 [6], which is a state-of-the-art adversarial approach.

$$\text{AdaIN}(x, y) = \sigma(y) \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \mu(y) \tag{2}$$

Style-GAN2 uses adaptive instance normalization (AdaIN) (2), which matches the mean and variance of content within an image with the style of particular images. This could be useful in also focusing on the type of input image of the sparse dataset, such as an input illustrations or older images, to generate more realistic images. Although the images appeared to be relatively diverse and unique, Style-GAN2 could would put emphasis in this area.

However, Style-GAN2 takes a very long time to train, but could have been run if there was more time. Hence, ProGAN may still have been the best choice, in terms of balancing mode coverage, expressivity and time [3].

Additionally, a non-adversarial model could have been chosen or explored, such as an energy-based model, however these have fewer parameters and some take much longer to run.

Another limitation was that STL-10 contained images of size 96x96 pixels, however the generated images were of size 128x128 pixels, due to the design of the model (as it doubles the image size from a power of 2, such as 4x4 to 8x8 to ... to 128x128).

Finally, the datasets used may have been too diverse to begin with, as upon inspection, there were a few images where it was difficult to recognise any cars.

Bonuses

This submission has a total bonus of -2 marks (a penalty), as it uses an adversarial training method, however it is trained on CIFAR-10 and STL-10 at 96x96 pixels.

References

[1] I. Goodfellow et al. *Generative Adversarial Nets*. `https://arxiv.org/pdf/1406.2661.pdf`. [Online; accessed 02-Feb-2023]. 2014.

[2] A. Atapour-Abarghouei. *Simple GAN Example*. `https://github.com/atapour/dl-pytorch/tree/main/Simple_GAN_Example`. [Online; accessed 02-Feb-2023]. 2022.

[3] S. Bond-Taylor et al. *Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models*. https://arxiv.org/abs/2103.04922. [Online; accessed 02-Feb-2023]. 2021.

[4] Google Developers. *Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy*. https://www.researchgate.net/figure/Progressive-growing-step-for-PROGAN-during-the-training-process-Training-starts-with-4_fig3_349189619. [Online; accessed 02-Feb-2023]. 2021.

[5] T.A. Elilah. *ProGAN Implementation from Scratch PyTorch*. https://www.kaggle.com/code/tauilabdelilah/progan-implementation-from-scratch-pytorch/notebook. [Online; accessed 02-Feb-2023]. 2014.

[6] T. Karras, S. Laine, and T. Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. https://arxiv.org/pdf/1812.04948.pdf. [Online; accessed 02-Feb-2023]. 2019.

[7] T. Karras et al. *Progressive Growing of GANS for Improved Quality, Stability, and Variation*. https://arxiv.org/pdf/1710.10196.pdf. [Online; accessed 02-Feb-2023]. 2018.

[8] A. Persson. *ProGAN*. https://github.com/aladdinpersson/Machine-Learning-Collection/tree/master/ML/Pytorch/GANs/ProGAN. [Online; accessed 02-Feb-2023]. 2021.