Image Generation using Wasserstein GANs with Gradient Policy and Auxiliary Classification

Anonymous author

Abstract

This paper proposes the use of Generative Adversarial Networks with Wasserstein Loss, Gradient Penalty and Auxiliary Classification to generate images on CIFAR-10 and STL-10 at a resolution of 32x32.

1 Methodology

This project uses a Generative Adversarial Network (GAN) [2] to accomplish the synthesis of 32x32 resolution CIFAR-10 and STL-10 images. GANs comprise of two networks; a generator and a discriminator. The generator takes a batch of random noise as input and uses it to synthesise a batch of images. The discriminator takes a batch of images as input and tries to determine whether the images belong to the original dataset or have been created by the generator. These networks then play a min-max game whereby the generator attempts to convince the discriminator that the images it produces are real (from the dataset) and the discriminator tries to improve its ability to determine between the real and fake images. These networks are trained iteratively using the back-propagation algorithm.

$$\min_G \max_D V(D, G) \tag{1}$$

where

$$V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}\ (\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{2}$$

The GAN used for this assignment improves upon the basic GAN setup through the use of 4 key components outlined below.

1.1 Wasserstein Loss

Wasserstien GANs [1] improve upon the basic GAN model by replacing the discriminator with a critic network which evaluates the 'realness' and 'fakeness' of an image (as opposed to classifying them binarily). This is done using the 1-Wasserstein Distance (also known as the Kantorovich–Rubinstein metric) and results in the generator and critic being trained according to the following two loss functions:

$$\mathcal{L}_{\text{C}} = \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[D(G(\boldsymbol{z}))] - \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}\ (\boldsymbol{x})}[D(\boldsymbol{x})] \tag{3}$$

$$\mathcal{L}_{\text{G}} = -\ \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[D(G(\boldsymbol{z}))] \tag{4}$$

1.2 Gradient Penalty

When initially proposed, WGANs used weight clipping in order to enforce the 1-Lipschitz constraint. The authors of [3] propose using a gradient penalty to enforce this constraint,

in an effort to improve both training speed and sample quality. This gradient penalty can be defined as followed:

$$\mathcal{GP} = E_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[\left(\left\|\nabla_{G(\boldsymbol{z})}D(G(\boldsymbol{z}))\right\|_2 - 1\right)^2\right] \tag{5}$$

This gradient penalty is multiplied by a constant value λ (usually set to 10) and added to the critic loss such that it can now be defined as:

$$\mathcal{L}_{\mathrm{C}} = E_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[D(G(\boldsymbol{z}))] - \mathbb{E}_{\boldsymbol{x}\sim p_{\mathrm{data}}\,(\boldsymbol{x})}[D(\boldsymbol{x})] + (\lambda \cdot \mathcal{GP}) \tag{6}$$

1.3 Residual Network

The generator and critic architectures used were implemented in PyTorch based on the design proposed in [3] and the TensorFlow implementation presented in the authors' official GitHub repository [4]. Both networks use pre-activation residual blocks each with two 3x3 convolutional layers and ReLU non-linearities. Residual blocks used in the generator use nearest neighbour up-sampling prior to the second convolution and some residual blocks in the critic perform mean average pooling for down-sampling. Residual blocks in the generator also use conditional batch normalisation and residual blocks in the discriminator use layer normalisation. This allows for the use of Auxiliary Classification (see next section).

Generator			
	Kernel Size	Re-sample	Output Shape
z	-	-	128
Linear	-	-	128 x 4 x 4
Residual Block	$[3 \times 3] \times 2$	Up	128 x 8 x 8
Residual Block	$[3 \times 3] \times 2$	Up	128 x 16 x 16
Residual Block	$[3 \times 3] \times 2$	Up	128 x 32 x 32
Conv, Tanh	-	-	3 x 32 x 32

Figure 1: Generator ResNet architecture

Critic			
	Kernel Size	Re-sample	Output Shape
Residual Block	$[3 \times 3] \times 2$	Down	128 x 16 x 16
Residual Block	$[3 \times 3] \times 2$	Down	128 x 8 x 8
Residual Block	$[3 \times 3] \times 2$	-	128 x 8 x 8
Residual Block	$[3 \times 3] \times 2$	-	128 x 8 x 8
ReLU, mean pool	-	-	128
Linear	-	-	1

Figure 2: Critic ResNet architecture

1.4 Auxiliary Classification

The model incorporates properties of Auxiliary Classifier (AC) GANs [5] by training the critic to also predict which class from the dataset the input image belongs. This works by giving the generator network a list of class labels to use when generating a batch of images, and then updating the loss of the generator and critic to incorporate the class loss (correctness of the critic to correctly guess the images class), which is given by:

$$L_{AC} = E\left[\log P\left(C = c \mid X_{\mathrm{real}}\right)\right] + E\left[\log P\left(C =\mid X_{\mathrm{fake}}\right)\right] \tag{7}$$

Unlike with $\mathcal{L}_{\mathrm{G}}$ and $\mathcal{L}_{\mathrm{C}}$, where G and C are trying to maximise each others losses, G and C both benefit from L_{AC} being as small as possible and so actively aim to minimise its value.

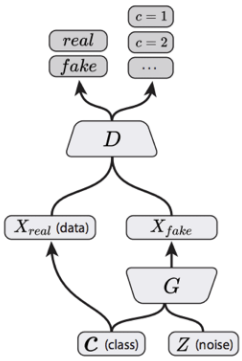

Figure 3: Architecture diagram of AC-GAN (taken from **??**)

2 Parameters

The method outlined above was trained on both CIFAR-10 and STL-10 resized to 32x32 resolution using the parameters suggested in [3] and the authors official GitHub implementation. The model was trained for 100k generator iterations, with 5 critic iterations per generator iteration. A batch size of 64 was used when training the critic with double that (128) used when training the generator. β values were set to 0.0 and 0.999 and ADAM optimisation was used with a learning rate of 2×10^{-3} linearly decayed to 0 over the 100k iterations.

3 Results

The results appear quite positive, with a number of realistic images being generated for both CIFAR-10 and STL-10. Some classes of each dataset are more successful than others with cars, horses, trucks and planes appearing most accurate. A random batch of non-cherry picked samples looks like this:

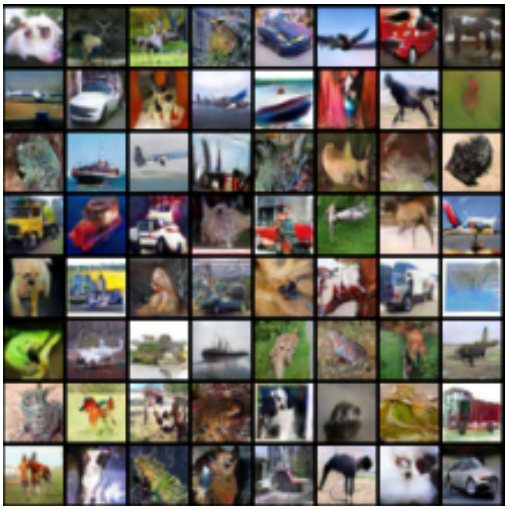

Figure 4: Random samples generated from the CIFAR-10 dataset

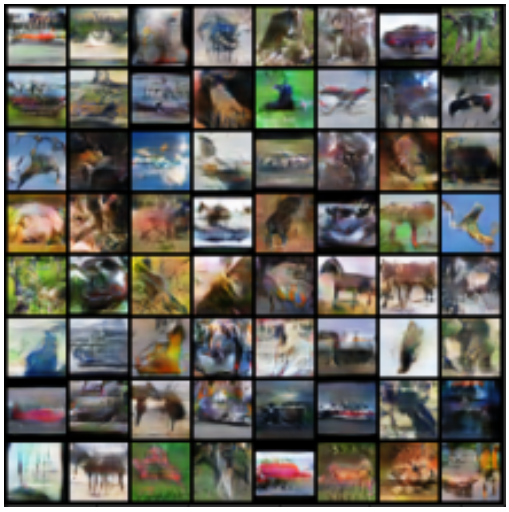

Figure 5: Random samples generated from the STL-10 dataset resized to 32x32 resolution

The next results show images generated by interpolating between points in the latent space:

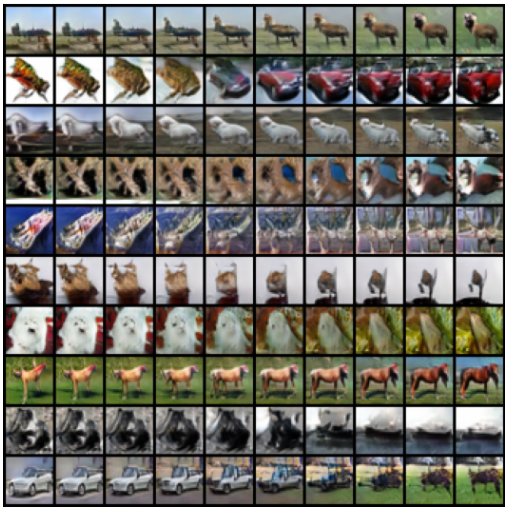

Figure 6: Linearly interpolated samples generated from the CIFAR-10 dataset

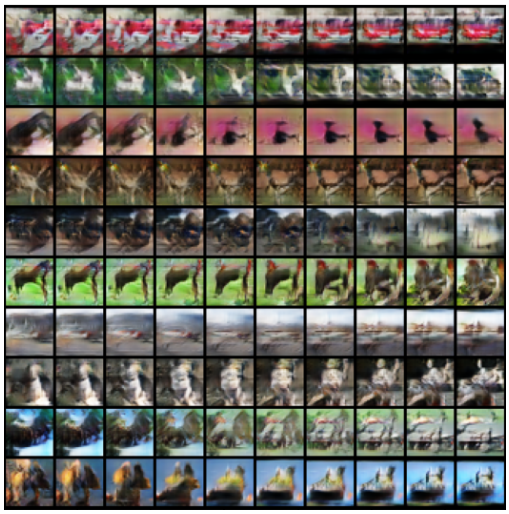

Figure 7: Linearly interpolated samples generated from the STL-10 dataset resized to 32x32 resolution

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

(a) horse (b) truck (c) car (d) dog (e) ship (f) plane (g) frog

Figure 8: Cherry picked samples generated from the CIFAR-10 dataset

4 Limitations

The CIFAR-10 samples look quite realistic and seem to replicate the results of the [3], however, the images generated from resized STL-10 do not quite reach the same quality. This is likely due to the effects of significantly downscaling the higher resolution images. Future work could look at changing the configuration of the residual blocks to accommodate the full resolution (96x96) STL-10 dataset. However achieving this will require a large amount of computational resources and a long training time (potentially up-to/over a week on a GPU) due to the amount of data needing to be processed and the number of input/output features that would be required.

Bonuses

This submission has a total bonus of -4 marks (a penalty), as it makes use of a GAN architecture but uses both CIFAR-10 and STL-10 datasets at a resolution of 32x32.

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: `1701.07875 [stat.ML]`.

[2] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: `1406.2661 [stat.ML]`.

[3] Ishaan Gulrajani et al. *Improved Training of Wasserstein GANs*. 2017. arXiv: 1704.00028 [cs.LG].

[4] Ishaan Gulrajani. *Improved Training of Wasserstein GANs*. https://github.com/igul222/improved_wgan_training. 2017.

[5] Augustus Odena, Christopher Olah, and Jonathon Shlens. *Conditional Image Synthesis With Auxiliary Classifier GANs*. 2017. arXiv: 1610.09585 [stat.ML].