Generating Pegasi Using a DCGAN

Anonymous author

Abstract

This paper details using a DCGAN approach with the CIFAR-10 and STL-10 datasets to generate images which look like a Pegasus. Numerous techniques have been implemented to improve the quality of the generated images, including: dropout layers, noisy labels, and one-sided smoothing amongst others. BCE loss has been used with the calculations of the objective functions. Some experimentation was required where the values of hyperparameters needed to be determined. These methods have been combined to successfully produce a desired best white Pegasus amongst a batch of 64 winged-horses.

1 Methodology

The method chosen was a DCGAN approach as detailed in [1], where a generative adversarial network is used with a generator which is attempting to create similar images to those in the given dataset, and a discriminator which is attempting to distinguish between real and fake images. Convolutional-transpose and convolutional layers are used in the generator and the discriminator. With these layers, batch normalization is applied to standardise each of the four-dimensional inputs for each mini-batch. This normalization is expected to noticeably reduce the time of the learning process [7] and also resolve sparse gradients [9]. Furthermore, ReLU activation functions were used in the generator and Leaky ReLU activation functions, with a slope of 0.2, were used in the discriminator, as negative values are allowed, to improve the learning speed of the model [1] and promote healthy gradient flow. The final layer in the generator and discriminator uses an element-wise Tanh and Sigmoid function respectively. Furthermore, Adam Optimization with a learning rate of 0.002, with beta values of 0.5 and 0.999, were used as a result of the findings in [1]. A weights-initialisation function was used to randomly initialise all weights, the function used can be found at [8], where the weights are initialised from a random distribution with a mean of 0 and standard deviation of 0.02 as recommended in [1].

Throughout the development of the model, results were compared both visually and by binary cross entropy loss values. A Torch implementation of this BCE loss was implemented with a mean reduction and no optional weights tensor, creating a comparable measure between the target and the output. The loss is calculated by:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) = mean(\mathcal{L}) = \{l_1, ..., l_N\}^\top, l_n = -[\boldsymbol{y}_n \cdot log\boldsymbol{x}_n + (1 - \boldsymbol{y}_n) \cdot log(1 - \boldsymbol{x}_n)] \quad (1)$$

[10] where N is the batch size, 64 in this case, $\boldsymbol{x}$ are the inputs, and $\boldsymbol{y}$ are the target outputs. The two inputs to the BCE loss function are the input tensor and the target value. The BCE loss in this approach is used to enforce the objective functions, that is to say, the functions which the generator, G and discriminator, D are attempting to maximise over data $\boldsymbol{x}$ with defined noise variables $p_z(\boldsymbol{z})$. As determined in [11], G should be trained to maximise $logD(G(\boldsymbol{z}))$, whereas D should instead be trained using $logD(\boldsymbol{x}) + log(1 - D(G(\boldsymbol{z})))$. The full algorithm for the ascending and descending of the stochastic gradient of D can be found in [11]. The BCE loss then enforces the objective functions as, for example, the output of D on some input $\boldsymbol{x}$ needs to be the probability that $\boldsymbol{x}$ came from the real data instead of the generated data, which is found by the BCE loss's measure of the difference between the target and the given output. The gradient descent of the loss is then used to optimise the parameters through backward propagation, this has been implemented using Torch.

Earlier tests resulted in mode collapse before long and so first the size of the dataset available to the DCGAN needed to be expanded. Primarily, both the CIFAR-10 and STL-10

datasets were duplicated but with a horizontal flip, doubling the size of the available training data. Other additional techniques were attempted to increase the size of the training data, for example, greyscale copies of the datasets were tested to see if it could also increase the number of white horses generated but unfortunately the combination of coloured and greyscale images poorly impacted results. Other copies involved added noise, rotation, and cropping with translation or slight rotation. All of these were successfully implemented but ultimately worsened the quality of the generated images and so were removed. I also found that the normalization of images during the training process greatly improved results.

Initially, testing was performed solely on a training set of horse images so that the effects of various adjustments could be evaluated more clearly. The first of these was the introduction of dropouts, this helps introduce randomness into the DCGAN to help prevent it from suffering over-fitting and improving the generalisation error as it is less likely for the model to become stuck in an equilibrium [2]. Dropout layers were tested after being implemented in both the generator and discriminator, and also solely on the discriminator, resulting in the latter method being chosen. Furthermore, several different dropout values were tested, an initial probability value of 0.2, or 20%, was used as a result of the findings in [3], before a probability of 0.15, or 15%, was settled on. This was a Torch implementation and so the elements of the input tensor are randomly sampled using a probability of 0.15 from a Bernoulli distribution [4].

With this current existing method, the loss of the discriminator would greatly increase as it was outperformed by the generator. An original remedy was to give the discriminator a headstart when training, inspired by the method of alternating between k steps of optimising D for every one step of optimising G in [11], but this was later replaced by the introduction of noisy labels [5]. This technique involved introducing some small probability (5% was chosen) that the label used would be incorrect, instead of simply using a 1 or 0 for a real or fake label, improving the robustness of the network. In addition to this, one-sided label smoothing was added to the "real labels", to assist the discriminator in avoiding overly confident classification, which would increase its loss, but rather by calculating soft probabilities [6]. For this project, the smoothed real labels were generated within the range of 0.85 and 1.10.

After realising that more weight could be placed on specific indices for each of the CIFAR-10 and STL-10 datasets where the image was more desirable, a simple program was created to allow manually selecting particular indices. After some experimentation, the indices for good CIFAR-10 planes were duplicated in the training data whereas specifically selected STL-10 indices were used as a replacement for selecting by class names due to the original lack of labelled data.

Next, planes and birds were introduced to the training set with the intention of combining them with the horses so that pegasi may be generated. It should be noted that for each epoch, the number of iterations used was the product of the size of the dataloader and the size of the batch to be generated. This kept the number of iterations as a suitable multiple of the batch size and reduced the risk of the training processes of different images overlapping and interfering with each other. The first attempts at combining the wings to the horses simply involved adding these various categories into a single dataloader and running the DCGAN identically to as if it had just been training on horses again. Predictably, this produced unfavourable and distorted results, the majority of which could not clearly be identified to belong to any category. The development following this involved limiting the size of the training data of the winged images with respect to the size of the horse training data. The thought process behind this was that the distorted images currently being produced were a split of 50% horse, and 25% of both plane and bird, whereas the percentage of the horse data should be significantly more prominent. This led to defining a 7:3 horse-wing ratio giving slightly improved results but still far from the desired final outcome. This idea of introducing size limitations on the winged training data was kept. Another unsuccessful approach involved giving the network a certain number of epochs where it would train solely on the horse training data before the wings training data was then incorporated afterwards. Again, this proved mixed results with a few Pegasus-like images being generated, however they were too infrequent to establish any confidence in the approach.

The final attempt at mixing the two sets of training data, which was eventually kept and used to produce the results shown, involved introducing rounds, where each round trained a defined number of epochs on either the dataloader containing the horses or the dataloader containing the winged images, either bird or plane. Due to the already large number of iterations and the restricted memory and time usage on Google Colab, the DCGAN was trained for 3 epochs on the horse images initially, followed by 3 further epochs on the winged images (this being a considerably smaller dataloader and so required fewer iterations), finally followed by one further round on the horse images where the best results produced would usually be within the first epoch of the final round. This approach outputted realistic and impressive horse images by the end of the first round, reasonable winged images by the end of the second round, and then the reintroduction of the horse images for the third round of training created the right balance of a predominantly horse-focused generated image but with residual winged-features from the previous round. Numerous factors including: ending on a horse-trained round, having twice as many horse-trained rounds as wing-trained rounds, and having less training time spent on the wing-trained round, all contributed to a weighting between the different sets of training data which produced Pegasus-like results. For one final improvement to this method, the current results of the second round were subject to a considerable amount of change due to the learning process trying to train on two different classes, planes and birds. The birds were removed in favour of the planes so that more accurate winged images could be generated in this second round and so have more of a discernible impact on the results. The rationale behind choosing the planes instead of the birds include the acknowledgement that there were a larger number of suitable plane indices due to wings constantly being outstretched, and also the planes were, unlike the birds, often white, which would help to generate white winged horses in particular.

The final structure of the DCGAN is as below:

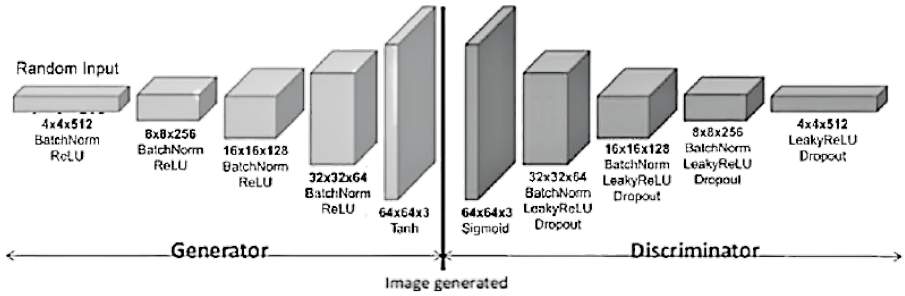

Fig.1: DCGAN Layers, edited from [12]

2 Results

From the images generated by the DCGAN in its best batch, the following image has been selected as the single best image of a white pegasus. A vague horse can be seen facing to the right, with wings attached to the bulk of its body but with no other features from the plane training data negatively impacting the image.

Fig.2: Best generated Pegasus

Below is the full batch of the 64 generated images. Within this batch, there are several images which could resemble winged-horses and so the methodology chosen has, at least to some extent, been successful. The discriminator loss calculated at this stage was 1.730 whereas the generator loss was calculated as 0.403. Further training would expect to see

a continued increase in the discriminator loss and a decrease in the generator loss which could perhaps be fixed through adjustments to dropout values or reintroducing the headstart for the discriminator. It must be noted that the resultant images have a noticeably lower contrast than the training data which may be a result of either too many iterations within the epochs, or perhaps a fault with the normalization of inputs and eventual de-normalization of the outputs.

Fig.3: Best batch of 64 generated winged-horses

3 Limitations

It is likely that the results produced could be improved by further training. This could be implemented by adding some number of pairs of rounds to the training, the first in the pair being another round of training on the winged-image's dataloader, and then the second round being trained on the horse dataset, so that the final round will always be training on the horse dataloader. Alternatively, the current round structure could be maintained but more than the current 3 epochs could be used within each. This increase in training time was not possible due to both natural time constraints with the project deadline, and in particular, time constraints imposed by Google Colab for GPU usage.

Furthermore, better results may be achievable through increasing the sizes of the datasets through applying additional transformation, but Google Colab must also limit RAM usage. This restriction would need to be taken into consideration if more time was to be spent on training.

Bonuses

This submission has a total bonus of $-4 + 1 + 1 = -2$ marks (a penalty), as it uses an adversarial training method (DCGAN), but it has been trained not only on CIFAR-10, but also STL-10 resized to 64x64 pixels.

References

[1] Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." In: *arXiv preprint arXiv:1511.06434* (2015).

[2] Brownlee, Jason. "A Gentle Introduction to Dropout for Regularizing Deep Neural Networks" *machinelearningmaster.com* [online] Available at: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/ (Accessed: 2 January 2021)

[3] Mordido, Gonçalo, Haojin Yang, and Christoph Meinel. "Dropout-gan: Learning from a dynamic ensemble of discriminators." In: *arXiv preprint arXiv:1807.11346* (2018).

[4] PyTorch "torch.nn" *pytorch.org* [online] Available at: https://pytorch.org/docs/stable/nn.html#dropout-layers (Accessed: 27 December 2020)

[5] Brownlee, Jason. "How to Implement GAN Hacks in Keras to Train Stable Models" *machinelearningmaster.com* [online] Available at: https://machinelearningmastery.com/how-to-code-generative-adversarial-network-hacks/ (Accessed: 2 January 2021)

[6] Goodfellow, Ian. "Nips 2016 tutorial: Generative adversarial networks." In: *arXiv preprint arXiv:1701.00160* (2016).

[7] Versloot, Christian. "What is Batch Normalization for training neural networks?" *machinecurve.com* [online] Available at: https://www.machinecurve.com/index.php/2020/01/14/what-is-batch-normalization-for-training-neural-networks/ (Accessed: 14 February 2021)

[8] InkaWhich, Nathan. "DCGAN Tutorial" *pytorch.org* [online] Available at: https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html (Accessed: 27 December 2020)

[9] Briggs, James. "Beating the GAN game" *towardsdatascience.com* [online] Available at: https://towardsdatascience.com/beating-the-gan-game-afbcce0a20be (Accessed: 28 December 2020)

[10] PyTorch "BCELoss" *pytorch.org* [online] Available at: https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html (Accessed: 15 February 2021)

[11] Goodfellow, Ian J., et al. "Generative adversarial networks." In: *arXiv preprint arXiv:1406.2661* (2014).

[12] Suh, Sungho, et al. "Generative oversampling method for imbalanced data on bearing fault detection and diagnosis." In: *Applied Sciences 9.4* (2019): 746.