Efficient CIFAR-100 Modelling

Anonymous author

Abstract

This paper presents an in-depth study on CIFAR-100 image classification and generation, leveraging advanced deep learning techniques within parameter limits by utilising model pruning. For classification, it employs convolutional neural networks enhanced by residual blocks, batch normalisation, and Adam optimisation, achieving significant accuracy improvements. In generative modelling, a Deep Convolutional Generative Adversarial Network (DCGAN) is tailored for CIFAR-100, focusing on adversarial training dynamics and architectural optimisations. Despite notable successes, the study identifies limitations like overfitting and suggests future exploration in advanced regularisation methods to boost model performance and generalisation capabilities further.

Part 1: Classification

1 Methodology

At the core of our model is the convolution operation fundamental to CNNs, enabling feature extraction and hierarchical organisation from images, defined as:

$$F(i,j) = (G * H)(i,j) = \sum_{m}\sum_{n} H(m,n)G(i-m, j-n), \tag{1}$$

Where F is the output feature map, G is the input image, and H is the kernel or filter, epitomises the process of applying filters across the input data to produce feature maps. These operations capture the spatial hierarchies inherent in images, enabling the model to learn abstract data representations as it progresses through sequential layers [7].

Enhancements via residual blocks and bottleneck structures [4] address the vanishing gradient issue by facilitating alternative gradient pathways ($y = F(x, \{W_i\}) + x$) and optimising input channel dimensionality for computational efficiency. Our 'MyCNN' architecture, inspired by [12], incorporates these elements alongside batch normalisation and ReLU activation within its convolutional layers, ensuring depth without excessive computational demands.

Optimisation utilises the Adam optimiser for its adaptive learning rates, proving advantageous over conventional methods for rapid convergence [5], augmented by a cyclic learning rate scheduler for optimal learning rate adjustment throughout training. The hyperparameters are generally kept the same as used in their respective original papers and implementations.

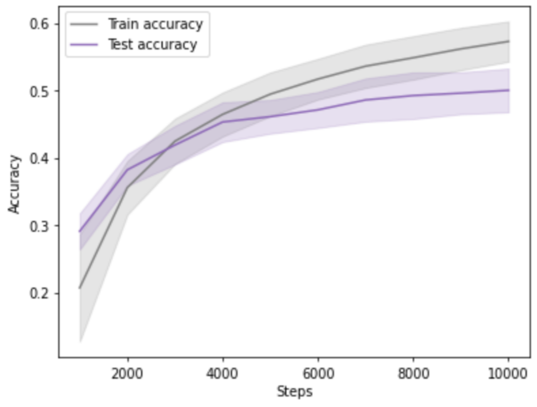
2 Results

The network has 99,440 parameters.

It attains 57.3% training accuracy and also 50.01% testing accuracy at 10,000 optimisation steps, which is a moderate result, moderate variance; this over-fitting is due to a larger batch size being used (train loss: 1.479, train acc: 0.573±0.030, test acc: 0.501±0.032).

The dataset under review suggests our model is verging on high variance, as depicted in the ensuing graph, hinting at its nearing optimisation ceiling; hence, any further advancements might plateau or wane due to overfitting. any more Optimising of the models performance necessitates a judicious assessment of training duration and parameter intricacy. While prolonging the training phase can potentially bolster performance to a degree, prevailing evidence intimates an impending rise in variance, thus heralding the onset of diminishing returns and heightened overfitting risks. Augmenting the model's parameters can amplify its capacity, yet it concurrently mandates the employment of sophisticated regularisation strategies to safeguard generalisation. Achieving a strategic equilibrium is crucial, as elucidated by Goodfellow et al. [2] and further explored by Smith and Topin [10] regarding the training duration's impact on model efficacy.

Here is the training graph:



3 Limitations

The results are moderate but compare poorly to the state of the models and would not be a very useful classifier. In the future, other more expressive functions could be used, such as advanced regularisation techniques like mixup or CutMix, adopting dynamic learning rate schedulers such as Cosine Annealing, and enhancing data augmentation with colour jitter and random crop is essential. These strategies aim to reduce overfitting, optimise training efficiency, and improve the model's generalisation capabilities, effectively addressing its key limitations.

Part 2: Generative model

4 Methodology

The methodology involves training a Deep Convolutional Generative Adversarial Network (DCGAN), necessitating an initial understanding of the Generative Adversarial Networks (GANs) framework. GANs, conceived by Goodfellow et al [3], instantiate a game-theoretic model in which two neural networks, the generator (G) and the discriminator (D), engage in a minimax competition. The generator strives to create data indistinguishable from authentic data, aiming to deceive the discriminator, which, in turn, aims to discern between real and generated data accurately. The objective function governs this adversarial process:

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

where x denotes an instance drawn from the real data distribution $p_{\text{data}}(x)$, and z is a noise vector sampled from a predetermined distribution $p_z(z)$. The generator's output, $G(z)$, is a synthetic data point derived from the noise vector. The discriminator's output, $D(x)$, represents the probability estimate that a given input x is drawn from the real data distribution as opposed to being generated by G.Theoretically, this game's equilibrium is achieved when $P_g = P_{data}$, rendering the discriminator unable to differentiate between real and synthetic inputs, essentially making random guesses.

DCGAN is a direct extension of the GAN described above through the strategic substitution of fully connected layers with convolutional layers [8]. This design choice is underpinned by the premise that convolutional kernels are adept at learning spatial features that echo the distribution characteristics of the input dataset, thereby facilitating a more nuanced synthesis and discrimination of images. In the DCGAN architecture, the generator and discriminator are configured as mirror images of each other, employing 2D transposed convolutions and 2D convolutions, respectively, to achieve their goals.

The generator embarks on the task by transforming an $n_l \times 1 \times 1$ latent vector into a $3 \times 64 \times 64$ image representation through a series of five transpose-convolutional layers, effectively unscaling the latent representation. This process is augmented by implementing batch normalisation and ReLU activations to ensure the stability and efficiency of the training process. Conversely, the discriminator undertakes the reverse operation, methodically condensing a $3 \times 64 \times 64$ image representation into a scalar value (fig 1). This value indicates the discriminator's assessment regarding the origin of the image, distinguishing between those synthesised by the generator and those originating from the training set.

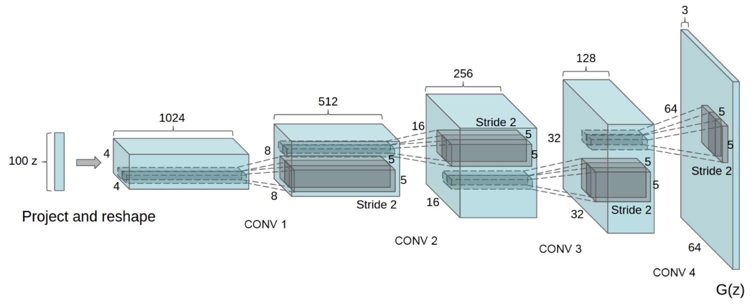


Figure 1: Diagram illustrating the DCGAN process [3]

4.1 Implementation Specific Details

The architectural blueprint of the DCGAN tailored for the CIFAR-100 dataset uses the principles delineated by Radford et al. [6]. To ensure the DCGAN architecture adheres to parameter limits, the network undergoes model pruning, a methodical reduction of parameters. To balance the generator and discriminator's complexity without affecting the generator to effectively upscale latent noise vectors into discernible images and the discriminator to evaluate their authenticity accurately, all within the parameter budget.

4.2 Optimisation Details

Incorporating the `StepLR` scheduler [11], the DCGAN architecture adeptly modifies the learning rate at specific intervals, significantly bolstering the training regimen. This, combined with the tailored Adam optimisers for the generator and discriminator, meticulously calibrated against a binary cross-entropy loss function[1], stabilises the learning trajectory and mitigates overfitting[8]. Preprocessing interventions, including random horizontal flips, strategic resizing, and meticulous normalisation, augment generalisation and secure feature invariance within the CIFAR-100 dataset. This confluence of strategies exemplifies a methodically optimised approach to deep learning tasks, ensuring both precision and rigour in the model's execution and theoretical foundation. The hyperparameters are generally kept the same as used in their respective original papers and implementations.

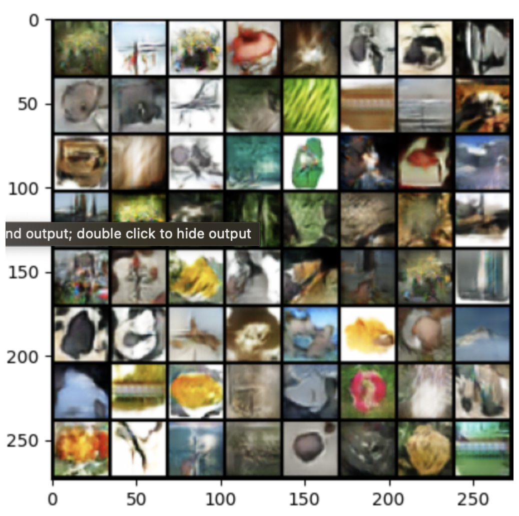
4.3 Interpolation Technique

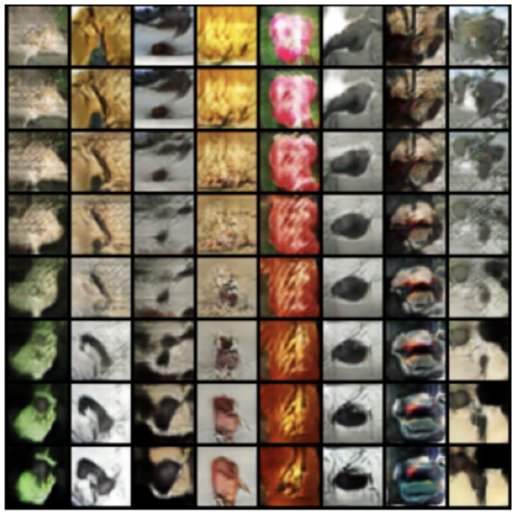
The spherical linear interpolation (SLERP) technique ensures smooth transitions in the latent space, crucial for generating images that evolve naturally without abrupt visual changes, known as "alpha bending." This method preserves the essence of the original images, facilitating a seamless flow in the generated image space. For a deeper mathematical insight into SLERP and its applications in image synthesis, refer to Shoemake's seminal work [9] on animating rotation with quaternion curves, which lays the foundation for understanding SLERP's principles and applications in computer graphics and deep learning models.

5 Results

The network has 1,000,052 parameters and achieves an FID of 31.56 against the CIFAR-100 test dataset. It was trained for 50,000 optimisation steps.

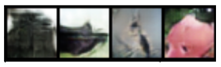
The results were varied, some images are quite clear, and their subjects are easily discernible, while others are blurry or obscured, making it challenging to identify the content, where a random batch of non-cherry picked samples looks like this (left) and interpolants between points in the latent space look like this (right):





These results are also quite similar to their nearest neighbours in the training data. This is highlighted with an LPIPS Similarity score of 0.146.

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



6 Limitations

The outcome of the model to create images varies, with some pictures looking realistic and others appearing blurry. In the future, we may need to use a different generative model that has a better architecture to achieve lower FIDs, or improve the model's capability to work with diverse datasets or use higher resolution images as well and refine preprocessing methods to enrich data quality, exploring advanced architectural frameworks that optimise computational efficiency.

Zero Penalties incurred

References

[1] *BCELoss documentation in PyTorch*. `https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html`. Accessed: YYYY-MM-DD.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

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

[4] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[5] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[6] LabML. *DCGAN Implementation*. 2021. URL: `https://github.com/labmlai/annotated_deep_learning_paper_implementations/blob/master/labml_nn/gan/dcgan/__init__.py` (visited on 02/14/2024).

[7] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

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

[9] Ken Shoemake. "Animating rotation with quaternion curves". In: *ACM SIGGRAPH Computer Graphics* 19.3 (1985), pp. 245–254.

[10] John Smith and Nicholas Topin. "The implications of training duration for the efficacy of neural networks". In: *Journal of Artificial Intelligence Research* 65 (2019), pp. 485–499.

[11] *StepLR documentation in PyTorch*. `https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.StepLR.html`. Accessed: YYYY-MM-DD.

[12] Yeonwoo Sung. *ResNet Implementation for CIFAR-100 in PyTorch*. `https://github.com/YeonwooSung/PyTorch_CNN_Architectures/blob/master/models/resnet.py`. 2021.