Efficient CIFAR-100 Modelling

Vincent Kenny

Abstract

This paper proposes a convolutional neural network, with a small number of parameters, for achieving a high classification accuracy on the CIFAR100 dataset. The model uses residual blocks, with depthwise separable convolutions to achieve 56.9% test accuracy with 98,068 parameters. This paper also outlines a DCGAN architecture for generation of unique images from the CIFAR100 dataset. This implementation uses 980,076 parameters to generate 10,000 new images with an FID score of 46.85 when compared to the test dataset.

Part 1: Classification

1 Methodology

For the problem of classification in the CIFAR100 dataset a Convolutional Neural Network (CNN) was used. As described in [5] the model is composed of multiple layers. First is the input layer followed by ten convolutional layers. In this case each convolutional layer has been expanded into a residual block. Next is a global average pooling layer which reduces the dimensionality of the input feature map to a single value per channel: saving parameters. The last layer is fully connected, and maps the model's output to each of the 100 classes.

Each convolutional layer in the CNN is replaced with a residual block as described in [2]. Each residual block has an input x upon which it performs some transformations F(x) such as a convolution. The block then outputs F(x) + x as opposed to just F(X). The result of this is that a model can be constructed with an increased depth of layers without the problem of vanishing gradients. Figure 1 shows a visualisation of each residual block.

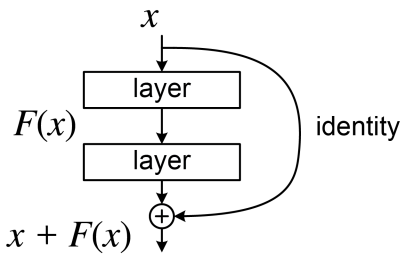

Figure 1, [2]

The structure of each convolutional block is as follows:

- Depthwise separable convolution
- Batch normalisation
- ReLU activation
- Depthwise separable convolution
- Batch normalisation
- Addition of identity
- ReLU activation

This architecture is implemented as described in the above paper, with one key change. Each standard 2D convolution is replaced with a depthwise separable convolution as described in [1]. First a depthwise convolution is performed, where a convolutional filter is applied to each colour channel separately. A pointwise convolution with a 1x1 kernel is then applied across the resulting outputs of each channel. This method achieves similar performance to a standard 2D convolution but with a greatly reduced number of parameters, hence it is useful given the constraints.

Increasing the number of channels from 42 to 128 in the last two residual blocks improved the performance of the model, without incurring too high of a parameter cost. The first 8 convolutional layers all have 42 channels: the highest number that I could achieve within the parameter limit. Random horizontal flips, as well as random crops of up to 4 pixels in each direction were applied to each image in the training set. These are common methods to reduce overfitting of the model and proved effective in this task. The Adam optimiser was used to train the model, with cross-entropy loss as the loss function.

2 Results

The network has 98,068 parameters. It attained 68.1 ± 4.3% training accuracy and 56.9 ± 4.6% testing accuracy at 10,000 optimisation steps. The final value of training loss was 1.099. Figure 2 shows the training graph over 10,000 optimisation steps. The test accuracy began to plateau towards the end of training which leads me to believe that more steps would not have yielded a significant improvement in performance. 1.099

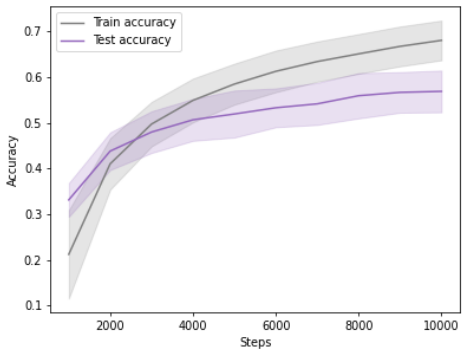

Figure 2

3 Limitations

The current state-of-the-art model for low parameters [7] achieves 71.35% test accuracy with 520,000 parameters. This model has more than 5 times as many parameters, for approximately 14 percentage points of improved accuracy over my model. Comparing my implementation to this it is clear that there is room for improvement: however, the performance is satisfactory.

I briefly experimented with using dropout before the final fully connected layer in my CNN, however this reliably decreased test accuracy. This could suggest that my CNN architecture is not particularly robust, and that it may not generalise well if there is any noise or distortion in the input.

I did not experiment extensively with the hyperparameters of my model. A grid search could be implemented to explore optimal parameter values more extensively; however, due to the large runtime of over 25 minutes on my machine, the search space would have to be significantly constrained. Using more powerful compute hardware would facilitate more parameter exploration. Further to this, learning rate scheduling could be implemented to help to fine tune the model in later optimization steps, where my model began to plateau.

Part 2: Generative model

4 Methodology

The code used for my generative model was taken from [3] and modified for CIFAR100.

Figure 3 shows the architecture of a typical Deep Convolutional Adversarial Neural Network (DCGAN) as proposed in [6]. The generator takes a random noise vector as an input; this vector is then passed to a series of four convolutional transpose layers that progressively upscale it into a larger spatial dimension. The output of the generator is a 32x32 image.

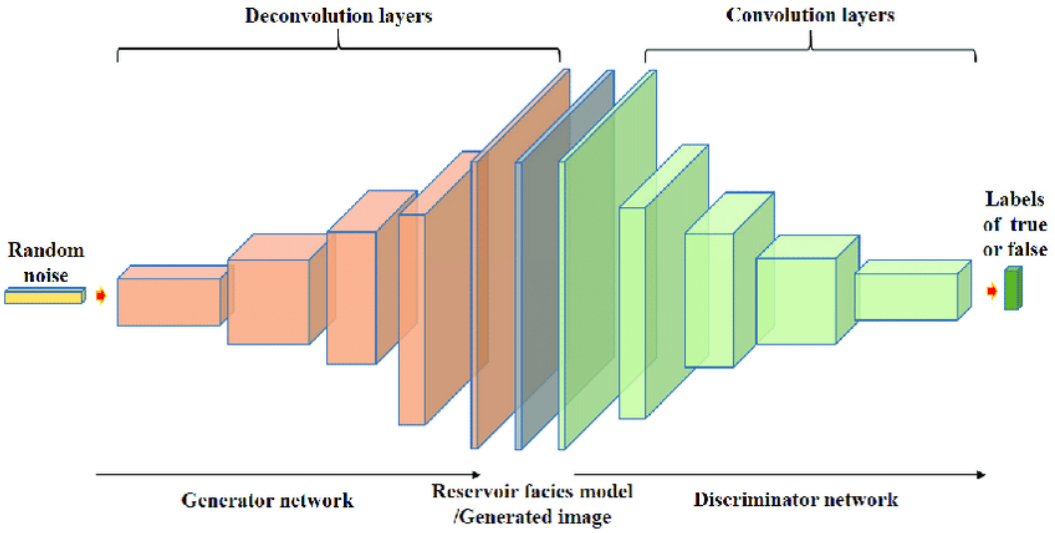

Figure 3, [4]

The structure of each generator layer is as follows:

- 2D convolutional transpose
- Batch normalisation
- ReLU activation

The discriminator architecture is similar to that of the generator, but in reverse. Each layer uses a 2D convolution with an increasing number of filters to capture the features within the image, reducing the number of spatial dimensions. The final layer outputs a single scalar, which is passed through a sigmoid activation function to determine the likelihood that the image is a real member of the dataset.

The structure of each discriminator layer is as follows:

- 2D convolution
- Batch normalisation
- Leaky ReLU activation

Binary Cross Entropy Loss (BCELoss) was used to measure the discrepancy between the discriminator's predictions and the ground truth labels for each image, real or fake.

$$\text{BCELoss}(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N} [y_i \log(\hat{y_i}) + (1 - y_i) \log(1 - \hat{y_i})]$$

During the training phase we update the weights for the discriminator and generator each step with the aim of minimising the BCELoss. Through this process the generator learns to trick the discriminator by creating increasingly realistic images.

5 Results

The DCGAN network achieved an FID score of 46.85 against the CIFAR-100 test dataset. It was trained for 46,000 optimisation steps. I chose to use a snapshot of the model taken at 46,000 steps as the images were more stable and less noisy than the samples taken at 50,000 steps. The generator model had 635,812 parameters and the discriminator had 344,264 for a total of 980,076.

The resulting images have good variety, with a range of colours, shapes, textures and styles. Figure 4 shows a batch of non-cherry picked samples taken from the model. Some of the images clearly resemble one of the labels from the dataset: There is a recognisable rocket, volcano and cow. However the majority of images are not recognisable.

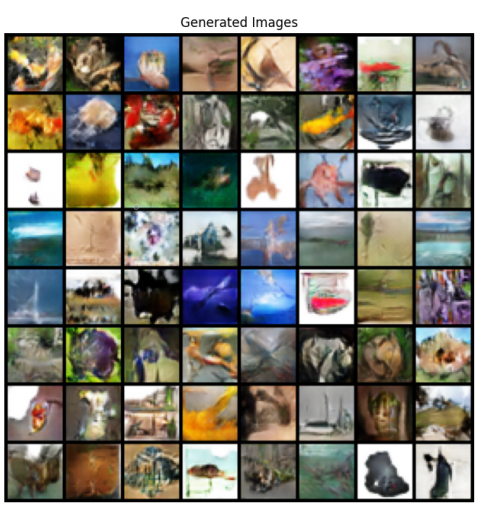

Figure 4

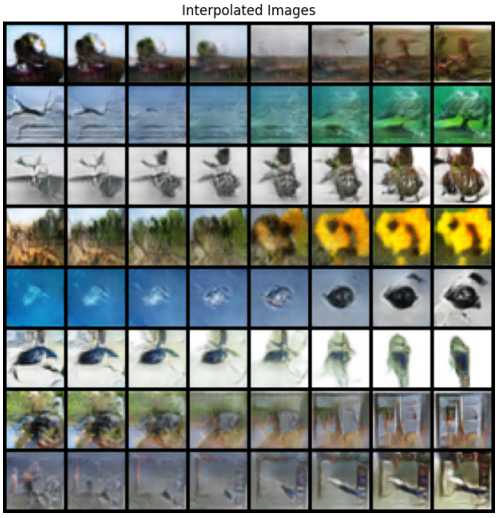

Figure 5

And here are some cherry-picked samples that show four of the better images my model was able to generate:

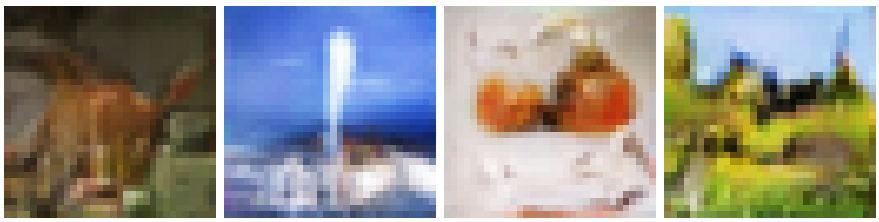

Figure 6

6 Limitations

During training of the DCGAN model, mode collapse was a frequent issue. Once this had happened in a training cycle the model would often not recover and the entire process would have to be restarted. A diffusion model would not have been susceptible to this issue, and therefore might have given a more stable training loop.

To improve the visual clarity of the model's outputs I could have trained the DCGAN on each subset of labels, giving a tighter spread of data for the generator to target. With 100 classes, the dataset may too broad for my model to achieve convincing results in any one category.

References

[1] François Chollet. "Xception: Deep learning with depthwise separable convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.

[2] 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.

[3] Nathan Inkawhich. *DCGAN Faces Tutorial*. 2018. URL: `https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html` (visited on 01/26/2024).

[4] Qingbin Liu et al. "An improved method of reservoir facies modeling based on generative adversarial networks". In: *Energies* 14.13 (2021), p. 3873.

[5] Keiron O'Shea and Ryan Nash. "An introduction to convolutional neural networks". In: *arXiv preprint arXiv:1511.08458* (2015).

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

[7] Joao Paulo Schwarz Schuler et al. "Grouped pointwise convolutions reduce parameters in convolutional neural networks". In: *Mendel*. Vol. 28. 1. 2022, pp. 23–31.