Efficient CIFAR-100 Modelling

Anonymous author

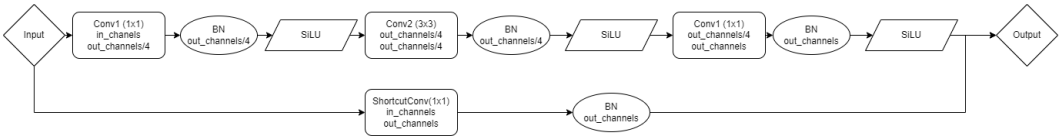
Abstract

This paper proposes a ResNet architecture with Bottleneck blocks, Convolutional Layers and a SiLU activation function for classification of photos in the CIFAR-100 database. The paper also proposes a vanilla GAN architecture which takes advantage of convolutional layers and purposeful network design to generate CIFAR-100 like images.

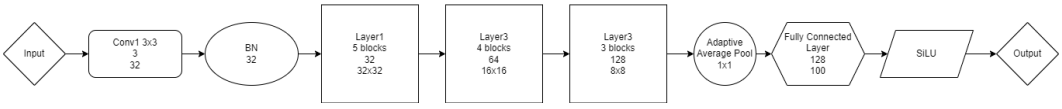
Part 1: Classification

1 Methodology

The classifier uses a standard ResNet architecture containing Bottlenck blocks. These blocks serve to increase the depth of the network without increasing parameter size nearly as much. This is done by first decreasing the dimensionality to 1/4th of the desired output channels, performing a 3x3 convolution on the input and then increasing the dimensionality back to the desired output channels. There is also a shortcut connection which helps to prevent the vanishing gradient problem and maintain complexity without a large increase in the number of parameters. [2]



The rest of the network contains large layers with multiple Bottleneck blocks in each. It was adapted from an online tutorial. [5] This architecture aims to maintain the most depth possible while maintaining under 100,000 parameters. By incrementing the number of features across 3 layers we further increase the complexity of the network allowing as much feature information to be gathered as possible before downsizing the image for more abstract feature detection.



The network uses an activation function called SiLU which was settled on after testing a wide range of activation functions and looking at prominent research. SiLU works by multiplying the Sigmoid function by the input tensor. [3]

$$x * \sigma(x)$$

This creates a smoother approximation of ReLU which improves classifier performance.

The training set is preprocessed with random horizontal flips and color jitter to prevent over-fitting. The loss is assessed using cross entropy, a common method of multi-class loss calculation that uses logarithmic curves to account for both having the correct class and having high certainty in that prediction.

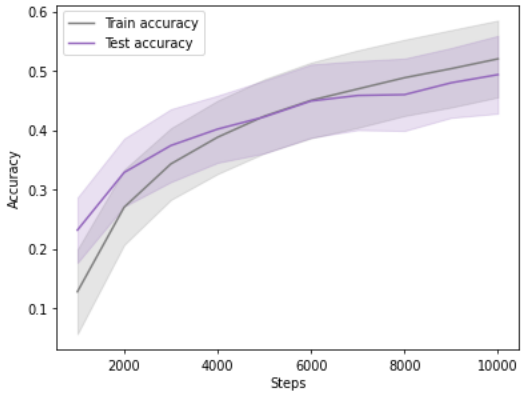
$$-\sum_{c=1} y_{o,c} \log(p_{o,c})$$

The model uses the Adam optimiser with a 0.002 loss rate, a value settled on after assessing a range of values. Adam uses a moving average of gradients across steps, allowing for stable training.

2 Results

The network has 99,364 parameters.

It attains 52.1% training accuracy and also 49.4% testing accuracy at 10,000 optimisation steps(train loss: 1.812, train acc: 0.521±0.065, test acc: 0.494±0.066).



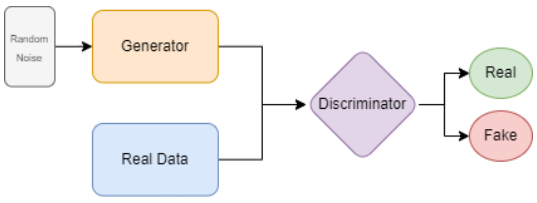
3 Limitations

The training process of the model is exceptionally long and memory intensive due to complexity of ResNets. The structure of many layers at the same depth could reduce the model's ability to distinguish more abstract features that could be extracted at higher depths. There is also some minor overfitting with a 2.7% difference between the test and training accuracy. In my implementation I tested various methods of preventing this such as dropout layers and more training set processing, however the current solution had both the best training and testing accuracy. In the future I could potentially implement alternative classification network architectures, use different activation functions, or try different methods of image preprocessing.

Part 2: Generative model

4 Methodology

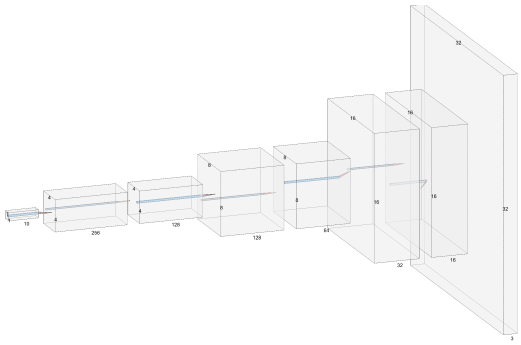
The model is a DCGAN which follows the usual architecture of a GAN but uses convolutional layers. I adapted my model from code by Lornatang on github. [4]



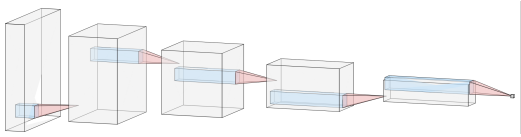
The aim is for the discriminator to maximize the number of real images it labels real and fake images it labels fake while the Generator aims to trick the discriminator and maximize the number of fake images that it labels as real. Both of these use Binary Cross Entropy which is similar to Cross Entropy but works with binary classification tasks.

$$-(y \log(p) + (1 - y) \log(1 - p))$$

Both the Generator and the Discriminator use convolutional layers in opposite directions. The generator model I have chosen appears below. The generator works by taking as input a 64x1x1x10 tensor and gradually increasing spatial dimensions using 4,2,1 layers with additional 3,1,1 layers after to better capture the features at that spatial dimension. This is repeated untill it generates a tensor of shape 64x32x32x3 This approach worked best out of the tests performed due to its ability to generate diverse images that capture CIFAR best. Alternative approaches either failed to properly generate more abstract features or were subject to extreme artifacts.



The Discriminator works by starting with 32x32 images with 3 feature layers and gradually expanding the feature space while decreasing the spatial dimensions. Spatial dimensions are decreased until the discriminator creates a 64x1x1x1 tensor containing the value assignment of all images in the batch. The discriminator and generator need to have somewhat similar loss to prevent mode collapse in a vanilla GAN. As such, the discriminator is far less complicated.



Both the generator and the discriminator use the Adam optimiser with a learning rate of 0.002 and betas of 0.5 and 0.999. Both optimisers have learning rate schedulers that reduce the learning rate by 5% every 1000 steps. This is to prevent mode collapse and to allow finer adjustments to the weights to converge more gently. Assigning the beta values allows fine tuning for the decay of the mean of the gradients and square gradients. By prioritizing the mean of the squared gradients the noise is reduced. Using a beta1 value of 0.5 for Adam allows the model to better adapt to changes in the gradient increasing speed of convergence without sacrificing too much stability. [1]

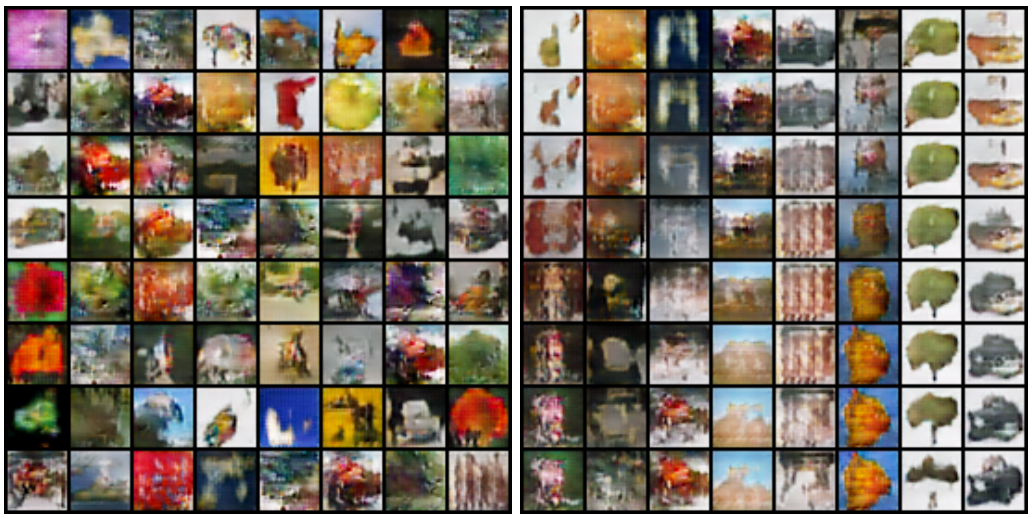
This generative model was decided upon after intensive testing of a wide variety of GAN as well as diffusion methods. The most promising of these was ACGAN, a GAN which embeds class information into the generator and has the discriminator also work as a classifier. This model was not used as the limited number of parameters led to poor performance.

5 Results

The network has 929472 parameters and achieves an FID of 94.53 against the CIFAR-100 test dataset. It was trained for 50,000 optimisation steps.

The results look somewhat odd. The shapes of items is the main concern, they mainly lack the smoothness expected of real images.The colors appear no different from CIFAR however the changes between colors are often not very smooth. There are also some slight grid artifacts which is a consequence of using convolutional layers for the model. The images are quite sharp and don't appear to be blurred at all which could occur with other approaches such as using Upsampling. A few categories are much more prominent than others. In

the samples to the left it is clear that flowers are over-represented. This is likely due to flowers having a less well defined shape and color making it difficult for the discriminator to distinguish between fake and real flowers incentivising flower generation.



These appear to be the best images output in the 2 batches above.

6 Limitations

The results are somewhat realistic but lack smoothness as well as containing poorly defined shapes and not representing all classes equally. A cGAN could likely solve the class representation issue and using upsampling to some degree could help with the smoothness. A diffusion model would likely capture shapes better as well. Alternate generator designs could also possibly improve the FID score such as additional later layers to fine tune images. The discriminator parameters could be cut down in order to accomodate for this. The colors look strong at a glance but are occasionally applied to the wrong shapes. For instance in the second to last interpolated column it appears to be trying to generate an animal of some sort however it is using a grassy texture. This could likely also be improved by using a cGAN.

Note: All Images used in this paper were created by me using draw.io and NN-SVG

References

[1] Pierre-Antoine Bannie. *Understanding Adam : how loss functions are minimized ?* 2019. URL: https://towardsdatascience.com/understanding-adam-how-loss-functions-are-minimized-3a75d36ebdfc.

[2] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[3] Dan Hendrycks and Kevin Gimpel. "Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units". In: *CoRR* abs/1606.08415 (2016). arXiv: 1606.08415. URL: http://arxiv.org/abs/1606.08415.

[4] Lornatang. *DCGAN-PyTorch*. https://github.com/charlespwd/project-title. 2021.

[5] Nouman. *Writing ResNet from Scratch in PyTorch*. 2022. URL: https://blog.paperspace.com/writing-resnet-from-scratch-in-pytorch/.