Efficient CIFAR-100 Modelling

Anonymous author

Abstract

We propose a parameter-efficient classification and generative models for the CIFAR-100 dataset. The proposed classifier utilises depthwise separable convolutions [6] and squeeze-and-excitation (SE) blocks [7], optimising the model for both computational efficiency and enhanced feature representation. We drastically minimise the model's computational demands and parameter size by integrating depthwise separable convolutions. The SE blocks effectively re-calibrate channel-wise feature responses, boosting the accuracy. We achieve a testing accuracy of 50.2%. The generative model produces an FID [5] score of 62.134, whilst being very simple in its architectural design. Both models are under 10k and 50k optimisation steps for classification and generative tasks respectively.

Part 1: Classification

1 Methodology

The method involves a Convolutional Neural Network (CNN) containing Depthwise Seperable Convolutions [6] with Squeeze-and-Excitation blocks [7].

1.1 Model Architecture

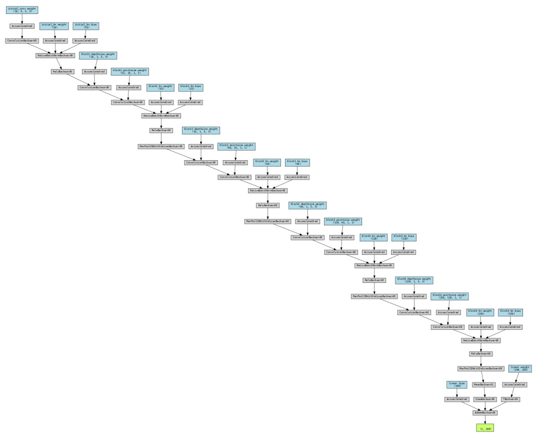


Figure 1: Visualization of the Model Processing a Tensor of Size [1,3,16,16]

Figure 1 above showcases the processing of a tensor by the model. The model consists of an SE block [7], a Depthwise-Separable block [6] and the Classifier - the main network. In the SE block, Adaptive Average Pooling [8] is applied to each input channel. It is then transformed through two dense layers with the ReLU activation function $\varphi(x) = \max(0, x)$, introduced in 1975 [2]. Sigmoid, σ, is then applied afterwards.

Depthwise-Separable block contains a depthwise convolution which applies a single kernel to a single input channel, given by $D_c(h, w) = \sum_{m,n} K_c(m, n) \cdot X_c(h+m, w+n)$. Pointwise convolution is then used to combine the outputs of the previous layer. Batch Normalisation, SE block, and non-linear ReLU activation function are then applied respectively.

The main Classifier initially takes the input through a convolutional layer followed by batch normalisation and ReLU activation, given by the expression $\mathrm{ReLU}(\mathrm{BN}(\mathrm{Conv}_{3\to16}(X)))$. These are then followed by 4 Depthwise Separable blocks with a Maximum Pooling layer after each, followed by Adaptive Average Pooling with a size of 1 and a dense layer for classification using Softmax, $\frac{e^{z_i}}{\sum_j e^{z_j}}$, to calculate the probability for each class. The proposed model contains 84,194 parameters in total.

1.2 Training Process

Our model's training leveraged data augmentation on the CIFAR-100 dataset to enhance its generalization capability. We applied random horizontal flips, random resized crops (scale of 0.8 to 1.0, size 32x32 pixels), and random rotations (±15 degrees) to the input images during training. These techniques introduced a more diverse training set without increasing the dataset size and resulting in no underfitting or overfitting, as detailed in Section 3: Results.

The proposed model uses 10,000 optimisation steps, each comprising of a prediction, an evaluation using Cross-Entropy Loss, as given by $L = -\sum_i t_i \log(p_i)$, and updating parameters through backpropagation via AdamW optimiser [9] with a learning rate of 0.0015, an empirically determined value close to optimal for this context.

2 Results

The model, with 84,194 parameters, achieved a training accuracy of 52.5% $\pm$ 4.5% and testing accuracy of 50.2% $\pm$ 4.2% after 10,000 steps, with a train loss of 1.742, as shown in figure 2.

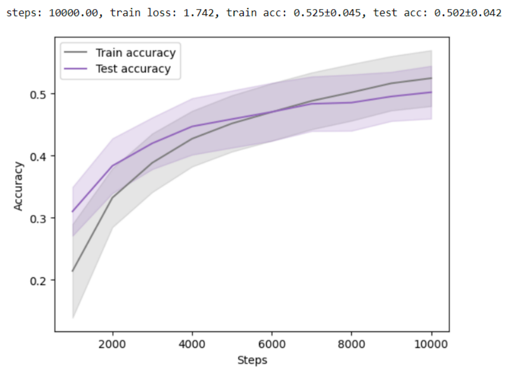


Figure 2: Training and Testing Accuracy of the Proposed Model

Depthwise Convolution without Pooling (No Data Augmentation): Test accuracy at 41.2%, train accuracy at 43.0%.

Depthwise Convolution with Pooling (No Data Augmentation): Increases test accuracy to 44.3%, train accuracy to 56.2%, introducing overfitting.

All Transformations: Achieved 50.2% test accuracy, showing balanced generalization.

Without Rotation: Slight improvement to 50.4% test accuracy but introduced some overfitting.

Without Crop: Test accuracy dropped to 49.8%, with increased overfitting compared to no rotation.

3 Limitations

The accuracy is low, only being 50.2% and relying on accuracy overlooks other types of errors (false positive/negative). As per the no free lunch theorem, this model would likely not perform well on other datasets. CIFAR-100's 32x32 pixel images may not provide sufficient detail for models to learn nuanced features necessary for higher accuracy in a real-world scenario with finer details. Image augmentation alone may not suffice to prevent overfitting, as indicated by the increasing overfitting trend observed in Figure 2. This necessitates exploring additional regularisation methods and incorporating dropout strategies to enhance model generalisation further, which may potentially inhibit accuracy in return.

Part 2: Generative model

4 Methodology

The proposed model uses the Generative Adversarial Network (GAN) framework, introduced by [3] and further advanced in the domain of deep convolutional networks by [11]. While our approach is inspired by these papers, it deviates from the constraints recommended in the DCGAN [11] study to explore alternative strategies. Our proposed model is loosely based on the DCGAN implementation by Aladdininpersson [10].

4.1 Model Architecture

GANs, introduced by Goodfellow et al. [3], consist of a Generator (G) that creates data resembling the real dataset, and a Discriminator (D) that differentiates real from generated data. These adversarial components iteratively improve through training, with G aiming to fool D, and D trying to classify data as real or fake (see Section 4.2: Training Process).

4.1.1 Discriminator Architecture

D is tasked with classifying images as real or fake through a series of convolutional layers, specified as follows:

1. **Conv1:** $\text{Conv2d}(\text{channels\_img}, \text{features\_d}, 4, 2, 1)$ followed by $\text{LeakyReLU}(0.2)$.
2. **Conv2:** $\text{Conv2d}(\text{features\_d}, \text{features\_d} \times 2, 4, 2, 1)$, $\text{BatchNorm2d}(\text{features\_d} \times 2)$, then $\text{LeakyReLU}(0.2)$.
3. **Conv3:** $\text{Conv2d}(\text{features\_d} \times 2, \text{features\_d} \times 4, 4, 2, 1)$, $\text{BatchNorm2d}(\text{features\_d} \times 4)$, followed by $\text{LeakyReLU}(0.2)$.
4. **Conv4:** $\text{Conv2d}(\text{features\_d} \times 4, 1, 4, 1, 0)$ with a Sigmoid activation function for output.

4.1.2 Generator Architecture

G maps a latent space vector z through a series of transposed convolutional layers.

1. **TransConv1:** $\text{ConvTranspose2d}(100, \text{features\_g} \times 4, 4, 1, 0)$, $\text{BatchNorm2d}(\text{features\_g} \times 4)$, and $\text{ReLU}(True)$. This layer initiates the process of mapping the latent space vector to the data space.
2. **TransConv2:** $\text{ConvTranspose2d}(\text{features\_g} \times 4, \text{features\_g} \times 2, 4, 2, 1)$, $\text{BatchNorm2d}(\text{features\_g} \times 2)$, with $\text{ReLU}(True)$.
3. **TransConv3:** $\text{ConvTranspose2d}(\text{features\_g} \times 2, \text{features\_g}, 4, 2, 1)$, $\text{BatchNorm2d}(\text{features\_g})$, followed by $\text{ReLU}(True)$.
4. **TransConv4:** $\text{ConvTranspose2d}(\text{features\_g}, \text{channels\_img}, 4, 2, 1)$, concluding with a Tanh activation to produce the final image.

4.2 Training Process

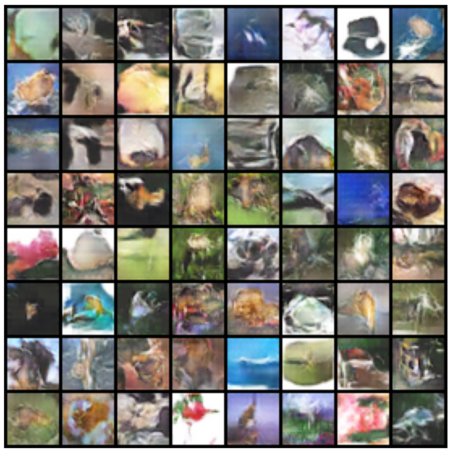
$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{1}$$

The objective function above is the min-max game played between D and G where D is to maximise its ability to classify real and generated data as such, and G aims to minimise D's classification accuracy. We use the Adam optimiser (lr=0.0002), optimising both D and G in 50,000 steps. Binary Cross Entropy (BCE), the loss function quantifies the performance of both D and G throughout the training process. The parameters of both D and G are updated iteratively by Adam's gradient descent based on the BCE loss for D and G.

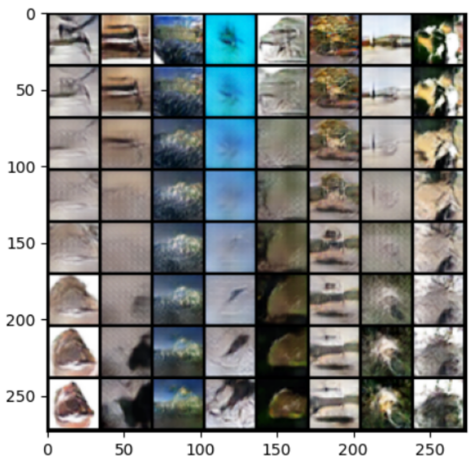
D is trained first by classifying real and generated data as such; calculating the BCE loss to measure its accuracy. Adam optimiser updates D's parameters based on the BCE loss to improve D's performance. Iteratively, G is trained to generate data that D misclassifies as real; based on the BCE loss of D's classification $D(G(z))$, Adam updates G's parameters. These interactions count as 1 optimisation step in our training.

5 Results

The proposed generative model contains 980,724 parameters, achieving an FID score of 62.134 after being trained on 50,000 steps. Below, we see 64 non-cherrypicked images generated by the GAN (left) and latent space interpolations between points in the latent space (right).



```
Step 50000/50000, Discriminator Loss: 0.2511, Generator Loss: 2.1110
```



The Fréchet Inception Distance (FID) [5] tells the similarity between real and generated images using feature vectors from the Inception model [12], with lower scores indicating better similarity. Latent space interpolations between two points, z_1 and z_2, are defined as $z_{\text{interp}} = (1 - \alpha)z_1 + \alpha z_2$, showcasing the model's generative continuity.

6 Limitations

Whilst our proposed model only provides a foundational approach to designing a GAN, advancements like WGAN [1] and WGAN-GP [4] have significantly improves the GAN architecture. These architectures acknowledge mode collapse; providing enhanced training stability. These models introduce a novel loss function based on the Wasserstein distance, providing a more meaningful and smooth gradient signal for the generator's training process. WGAN-GP further uses the Lipschitz constraint. Controlling the gradient's magnitude ensures smoother and more stable training, resulting in an improved quality of generated images.

References

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

[2] Kunihiko Fukushima. "Cognitron: A self-organizing multilayered neural network". In: *Biological cybernetics* 20.3-4 (1975), pp. 121–136.

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

[4] Ishaan Gulrajani et al. "Improved Training of Wasserstein GANs". In: *CoRR* abs/1704.00028 (2017). arXiv: `1704.00028`. URL: `http://arxiv.org/abs/1704.00028`.

[5] Martin Heusel et al. "GANs Trained by a Two Time-Scale Update Rule Converge to a Nash Equilibrium". In: *CoRR* abs/1706.08500 (2017). arXiv: `1706.08500`. URL: `http://arxiv.org/abs/1706.08500`.

[6] Andrew Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *arXiv preprint arXiv:1704.04861* (Apr. 2017). Available online at https://arxiv.org/abs/1704.04861. eprint: `1704.04861`.

[7] Jie Hu et al. *Squeeze-and-Excitation Networks*. 2019. arXiv: `1709.01507 [cs.CV]`.

[8] Shu Liu et al. "Path Aggregation Network for Instance Segmentation". In: *CoRR* abs/1803.01534 (2018). arXiv: `1803.01534`. URL: `http://arxiv.org/abs/1803.01534`.

[9] Ilya Loshchilov and Frank Hutter. "Fixing Weight Decay Regularization in Adam". In: *CoRR* abs/1711.05101 (2017). arXiv: `1711.05101`. URL: `http://arxiv.org/abs/1711.05101`.

[10] Aladdin Persson. "DCGAN Implementation in PyTorch". In: *GitHub repository* (2021). URL: `https://github.com/aladdinpersson/Machine-Learning-Collection/blob/master/ML/Pytorch/GANs/2.%20DCGAN/model.py`.

[11] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: `1511.06434 [cs.LG]`.

[12] Christian Szegedy et al. "Going Deeper with Convolutions". In: *CoRR* abs/1409.4842 (2014). arXiv: `1409.4842`. URL: `http://arxiv.org/abs/1409.4842`.