Efficient Discriminative and Generative Modelling

Aleksandar Yordanov

Abstract

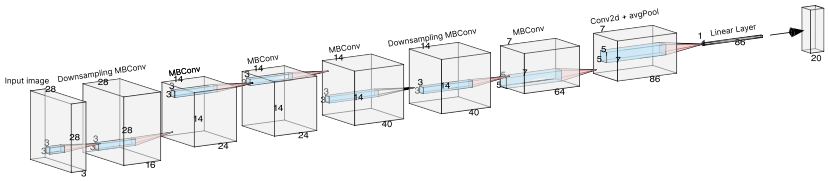
This paper proposes using a modified Efficient-Net model to classify images in the AddNIST database and a modified GAN based on the DCGAN architecture to generate images from the CIFAR100 dataset.

Part 1: Classification

1 Methodology

The method involves using the AddNIST dataset with a 9-layer deep Efficient-Net [12] architecture with a fully connected layer for classification and the Sigmoid Linear Unit activation function [1], $SiLU(x) = x * \sigma(x)$ where x is the logistic sigmoid function. The architecture can be seen below.

Efficient-Net architecture:



2 Efficient-Net Architecture

First introduced in the MobileNetv2 [11] architecture, Inverted Residual blocks play a crucial role in the efficiency of the EfficientNet architecture [12]. Unlike Residual blocks in ResNet [4] which maintain a wide → narrow → wide structure, Inverted Residual blocks maintain the opposite. First channels are expanded with a pointwise convolution then a depth-wise separable convolution is applied to efficiently capture spatial features. This is followed by a squeeze-and-excite block to perform channel-wise feature recalibration [5], and finally, a bottleneck layer that reduces the number of channels to the desired output.

I decided to go with a 9-layer-deep model with several residual connections as opposed to a wider, shallower network as I wanted to capture the complex relationship between each channel and the label $(r + g + b) - 1$. A deeper network will have more expressive power and be able to capture such relationships as proven by Telgarsky. [11]

3 Architecture Modifications and Experimental Setup

The modifications that I made to the architecture compose of: Using SiLU instead of ReLU6 as SiLU has been documented to work better on deeper networks [3] and not using the compound scaling method mentioned in [12] to allow for more granular control of each layer in my model.

Due to the lack of research on augmentation strategies for AddNIST, I went with commonly used MNIST data augmentation practices, a rotation of 15 degrees, a translation of 0.15, a

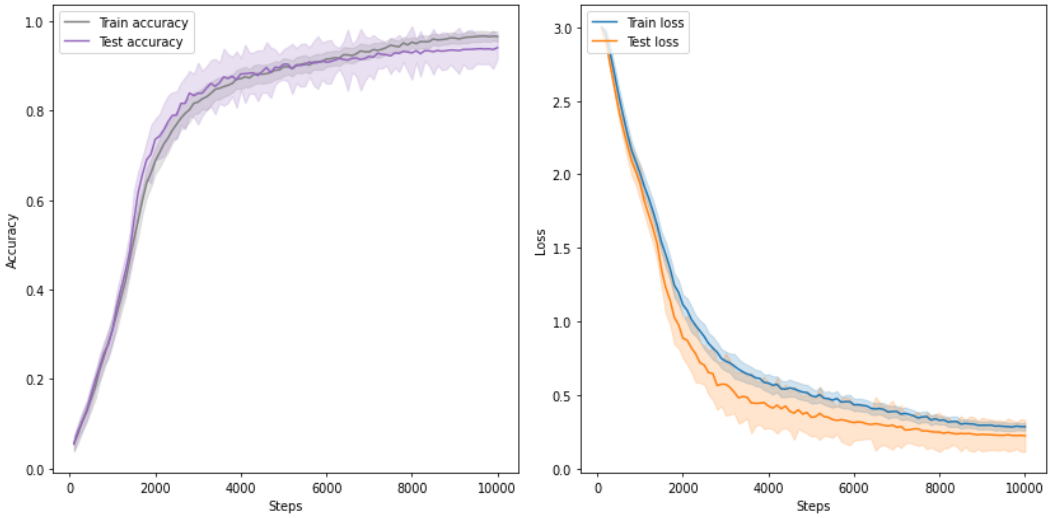
scaling between 0.85 to 1.15 and a 5 degree shear. Image Flipping was excluded to avoid digit misinterpretations, e.g. a flipped 6 resembling a 9.

For experimental setup, The model was trained using cross-entropy loss with label smoothing [8], the Adam optimiser [6] and a OneCycleLR with cosine-annealing to dynamically adjust learning rates. [7]

A batch size of 256 from $[64, 128, 256]$, a min/max learning rate of $\{0.001, 0.004\}$ from $[\{0.0008, 0.003\}, \{0.0004, 0.001\}, \{0.002, 0.006\}]$, a weight decay of 0.0001 from $[0.00001, 0.0001, 0.001]$ and label smoothing of 0.02 from $[0.01, 0.02, 0.05, 0.1, 0.2]$ were selected via a manual grid search and found to be the best hyper-parameters for this model.

4 Results

The network has 98,720 parameters. It attains 96.6% training accuracy and also 94.1% testing accuracy at 10,000 optimisation steps (train loss: 0.284, train acc: 0.966±0.011, test acc: 0.941±0.025).



From these results we can infer that the model is effectively learning the underlying pattern in the dataset. Test accuracy is consistently close to training accuracy, showing that the model generalises well to unseen data. The test and training curves are smooth and converge, suggesting a stable training process. The lower test loss is interesting to note, this is most likely due to regularisation being disabled on evaluation, leading to lower test set loss.

5 Limitations

The model achieves a high test accuracy of 93.8%, however, it plateaus at that accuracy at 9000 steps while training accuracy continues to grow, this suggests some light over-fitting and that the model lacks the capacity at 98720 parameters to fully learn the pattern in the AddNIST database, $(r + g + b) - 1$.

Part 2: Generative model

6 Methodology

My method is to train a Generative Adversarial model based on [2], the goal of which is to play a mini-max game in which a generator is producing images for which a discriminator

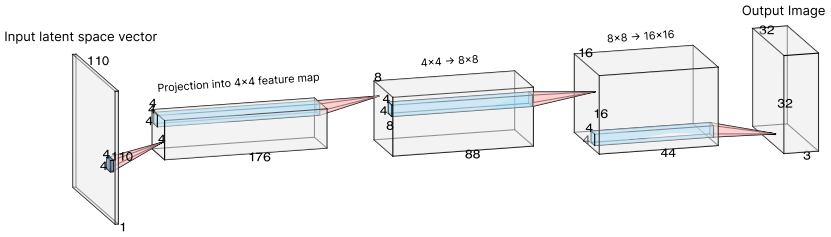
classifies as real based on the following equation:

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

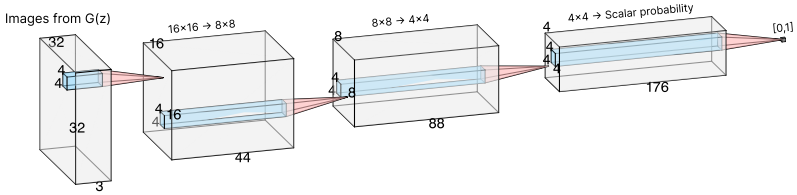
The role of the discriminator in this equation is to discriminate between real data x sampled from p_{data} and fake data $G(z)$ where z is a latent space vector sampled from a normal distribution. The discriminator effectively attempts to maximise $log\ D(x)$. The role of the generator is to minimise the probability that the discriminator will predict that its outputs are fake: $log(1 - D(G(z)))$. [2]

Given that this is a pure computer vision problem, I decided to implement the DCGAN architecture [10] as it is specifically designed for image generation tasks, excels in capturing spatial information, hierarchies and features and is easily scalable to the CIFAR100 Dataset. An overview of my DCGAN architecture can be seen below:

Generator Architecture:



Discriminator Architecture:



As seen in the diagrams, I have scaled the model from [10] down to generate and discriminate the 32 by 32 images in CIFAR-100. Latent space size (110) and base channel count (44) were chosen to fit parameter limits.

7 Architecture and Training Modifications

To improve training stability and image quality, I have made a couple modifications based on notes from ganhacks. The first modification I made was to use soft and noisy labels as a form of regularisation for the discriminator. This prevents the discriminator from becoming overconfident about its predictions. [8]

The second modification I made was to add gradually decaying noise over time based on this instance noise trick, this smooths the distribution of z and prevents the discriminator from focusing on high level features, such as fine textures, early on. This essentially acts as a low-pass filter and forces the discriminator to learn lower-level features before higher level ones. From my own testing this improved image quality by removing some artefacting seen by misplaced high-level details.

8 Experiment setup

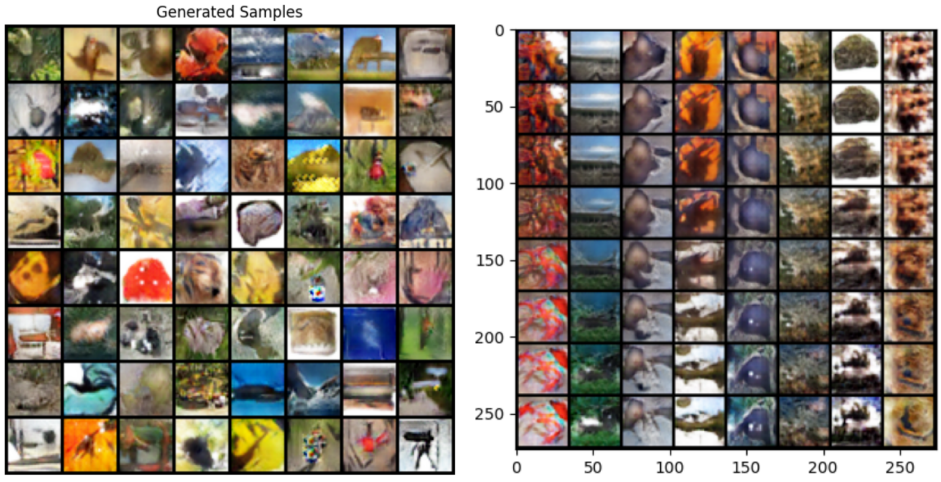
For experimental setup, The model was trained using BCE loss with the Adam optimiser [6] and weights initialised according to [10]. Recommended values for betas of $(0.5, 0.999)$ were used. [13]

A batch size of 256 from $[64, 128, 256]$, learning rates of 0.0002 from $[0.00001, 0.0001, 0.0005]$, soft label boundaries of 0.1 from $[0.1, 0.3]$, an initial noise value of 0.25 from $[0.1, 0.25, 0.5]$, and a noise decay rate of 0.9997 from $[0.999, 0.9993, 0.9995, 0.9997, 0.9999]$ were selected via a manual grid search and found to be the best hyper-parameters for this model.

9 Results

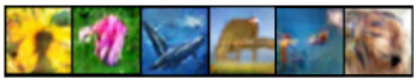
The network has a total of 998553 parameters and achieves an FID of 50.283 against the CIFAR-100 test dataset. It was trained for 50,000 optimisation steps.

The image on the left shows a set of random images, the image on the right shows a latent interpolation.



From the latent interpolation, we can see that, within a single interpolation path, there is variety of features, suggesting little to no mode collapse. This is also supported by the variety in the random sample. We can also see that these features shift gradually, suggesting that the generator has learned meaningful latent representations.

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



We can see from these results, that the model has learned to generate partially recognisable images (such as flowers, fish or animals). Indicating it has learned important features about those classes from the training data.

10 Limitations

One limitation that this model presents is in chequer-boarding artefacts that can be seen early on in the model's training. This could be solved by using upscaling along with convolutional layers that maintain spatial dimensionality [9], however this method lead to complete mode collapse as well as detail loss in the generated images until the point of mode collapse. However, these chequerboard artefacts can not be seen in the final images.

References

[1] Doya Elfwing Uchibe. "Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning". In: *arXiv.org* (2017).

[2] Mirza Goodfellow Pouget-Abadie. "Generative Adversarial Nets". In: *arXiv.org* (2014).

[3] Zhou Guo Wang. "Monitoring and Detection of Driver Fatigue from Monocular Cameras Based on Yolo v5". In: *CAA International Conference on Vehicular Control* (2022).

[4] Ren He Zhang. "Deep Residual Learning for Image Recognition". In: *arXiv.org* (2015).

[5] Albanie Hu Shen. "Squeeze-and-Excitation Networks". In: *arXiv.org* (2019).

[6] Lei Ba Kingma. "ADAM: A Method For Stochastic Optimization". In: *ILCR 2015* (2015).

[7] Hutter Loshchilov. "SGDR: Stochastic Gradient Descent With Warm Restarts". In: *arXiv.org* (2017).

[8] Hinton Müller Kornblith. "When Does Label Smoothing Help?" In: *arXiv.org* (2019).

[9] Olah Odena Dumoulin. "Deconvolution and Checkerboard Artifacts". In: *distill.pub* (2016).

[10] Chintala Radford Metz. "Unsupervised Representation Learning With Deep Convolutional Generative Adverserial Networks". In: *arXiv.org* (2016).

[11] Zhu Sandler Howard. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *arXiv.org* (2018).

[12] Le Tan. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *arXiv.org* (2020).

[13] Alwidian Zhou Mahmoud. "Evaluation of GAN-Based Model for Adversarial Training". In: *mdpi.com* (2023).