Efficient Discriminative and Generative Modelling

Will Farmer

Abstract

This paper proposes using a Convolutional Neural Network (CNN) with Inverted Residual blocks, Swish activation and Squeeze-Excitation blocks to classify images and a Deep Convolutional Generative Adversarial Network (DCGAN) with StepLR scheduling to generate images. The classification model suggests potential for efficient learning but has scalability issues, while the generative model produces low-quality images, highlighting the need for refinement.

Part 1: Classification

1 Methodology

The method involves using the AddNIST dataset to train a CNN by minimising the cross-entropy loss:

$$\mathcal{L}_{data} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c}) \tag{1}$$

where N is the number of samples, $y_{i,c}$ is a binary indicator (1 if class c is correct for sample i, else 0) and $\hat{y}_{i,c}$ is the probability for class c for sample i. To optimise the CNN, I implemented Swish activation, Squeeze-and-Excitation (SE) blocks, and Inverted Residual blocks. The Swish activation function $Swish(x) = x * \frac{1}{1+e^{-x}}$ enables a smoother gradient flow during training. After testing my model's performance with the Swish, ReLU and Softplus $\zeta(x) = log(1 + exp(x))$ activation functions, I found that Swish activation consistently provided the highest test accuracy.

However, the model's test accuracy after training was only 43.2%. I then read about MobileNetV2's [7] success using Inverted Residual blocks as well as how effectively SE blocks generalise across various datasets [6].

This led to my CNN having an single convolution layer with batch normalisation and Swish activation, before being forwarded onto my Inverted Residual blocks to extract complex features. The input channels were expanded by a factor of 4 with 1x1 convolution, before a 3x3 depthwise convolution was applied to each channel individually. The output of the depthwise convolutions was passed into my SE block. This performed global average pooling on the input (squeezing) before reducing and re-expanding the dimensionality of the input to reduce computation and extract the most critical information (exciting). The output feature map is then projected into a single vector, before being passed into a fully connected layer, determining which of the twenty classes the image is in.

This led to my model having a test accuracy of 65.0% after training. Thus far I had been using a learning rate value of 0.0001, no dropout layers and no L2 regularisation:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda\sum_{i} w_i^2 \tag{2}$$

I wanted to implement these regularisation techniques to improve the generalisability of my model and reduce the chance of overfitting. Using Optuna [1], I performed hyperparameter tuning, narrowing the search space after every fifty trials to find the optimal hyperparameter values. This dramatically increased the accuracy of my model.

2 Results

The network has 86,692 parameters and was trained for 10 epochs, each with 1000 optimisation steps for a total of 10,000 steps.

The fully-trained model achieves a test accuracy of 84.95%, which demonstrates the potential the model possesses in efficient learning.

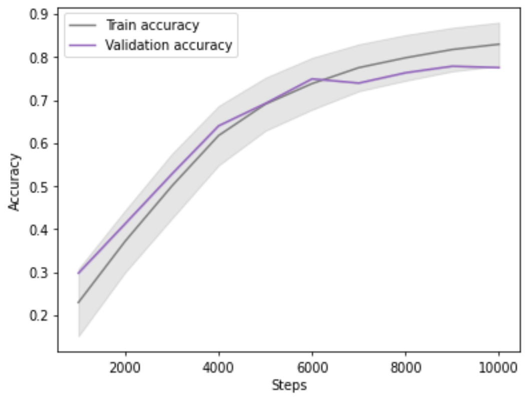

Figure 1: Validation accuracy and training accuracy after each epoch

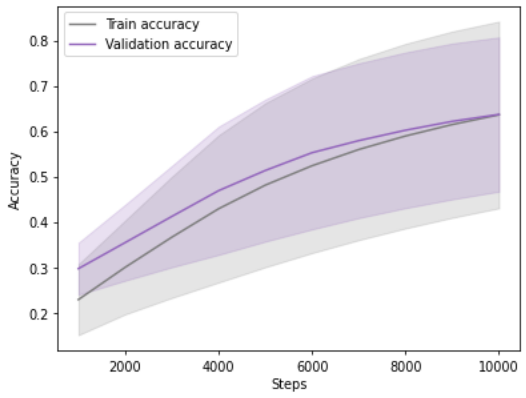

Figure 2: Mean validation accuracy and training accuracy throughout training

The mean training accuracy was 0.637±0.206 and the mean validation accuracy was 0.638±0.170. In the final epoch, the model attains 84.3% accuracy on the training set and 80.8% accuracy on the validation set.

3 Limitations

The results are promising. In such a confined learning environment the model is able to correctly classify 84.95% of the images in the test data and generalises well across all three sets. However, it has a large number of limitations. While ultimately improving the generalisability and performance of the trained model, the regularisation techniques cause accuracy to be critically low at the start, hence the low mean accuracy compared to the high test accuracy at the end of the training process. This can also result in the model having varying accuracies if re-trained; it typically fluctuates around 80% - 85%, but has sparsely and sporadically recorded accuracies of ≈ 71%. This is further evidenced by the extremely large standard deviations on the training and validation sets. When trained on 20,000 optimisation steps, the model's training accuracy continually increases (though more gradually), but the test accuracy remains at 83.1%. This is likely due to my hyperparameter tuning optimising my model's performance for 10,000 optimisation steps. Again, when adding an additional Inverted Residual block to my model (increasing the number of parameters to 104,804), the training accuracy increases with the test accuracy remaining at 83.0%. Therefore, while my model shows great promise in efficient learning, my hyperparameter tuning has caused it to overfit to its search space, which has made it less scalable.

Part 2: Generative model

4 Methodology

The method is to train a DCGAN by minimising binary cross-entropy loss:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right] \tag{3}$$

I initially considered using a Wasserstein GAN with gradient penalties (WGAN-GP) [2] [4]. However, the model kept experiencing mode collapse, and the resulting images quality was terrible. I attempted to add minibatch discrimination to the model to remedy this, but could not find a satisfactory solution that fit within the parameter limit. The original GAN model produced images that I deemed to be too noisy, so I opted to use a DCGAN instead, using convolutional layers rather than fully-connected layers to generate higher-quality images.

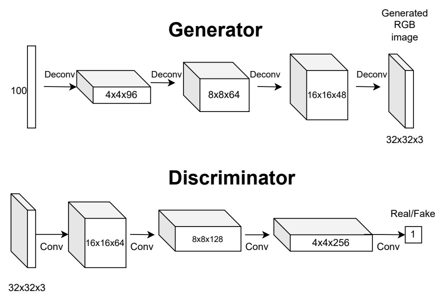

Figure 3: Architecture of the proposed DCGAN

The DCGAN is comprised of a generator with four deconvolutional layers and a classifier with four convolutional layers. Batch normalisation and ReLU activation occurs after each layer in the generator, except the final layer which has no normalisation and uses tanh activation

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4}$$

In the discriminator, Leaky ReLU activation occurs after each layer:

$$LeakyReLU(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0.2x & \text{if } x < 0 \end{cases} \tag{5}$$

with the exception of the final layer, which uses sigmoid activation:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{6}$$

The generator initially produces random noise. The discriminator predicts whether an image is fake (made by the generator) or real (from the dataset). The generator attempts to trick the discriminator and the discriminator attempts to correctly guess whether the image is real or fake. Their capabilities improve each step, and hence so do the generated images. It is important that an equilibrium is found between the two, as otherwise the generator will simply just produce random noise.

I used the StepLR scheduler [3] to gradually decrease the learning rate of the generator and discriminator after a set number of epochs. This helped stabilise the network during training, help keep a balance between the generator and discriminator as well as improving the likelihood of convergence. Tuning of the scheduler's gamma and step size, along with the generator and discriminator's learning rates reduced the FID score [5] heavily. While using transformations on the training dataset at the start, I discovered that they were harshly reducing image quality and consequently removed them.

5 Results

The network has 967,072 parameters and achieves an FID score of 44.46 against the CIFAR-100 test dataset after being trained for 50,000 optimisation steps.

The generated images look slightly blurry and it is difficult to discern what is being depicted in each one.

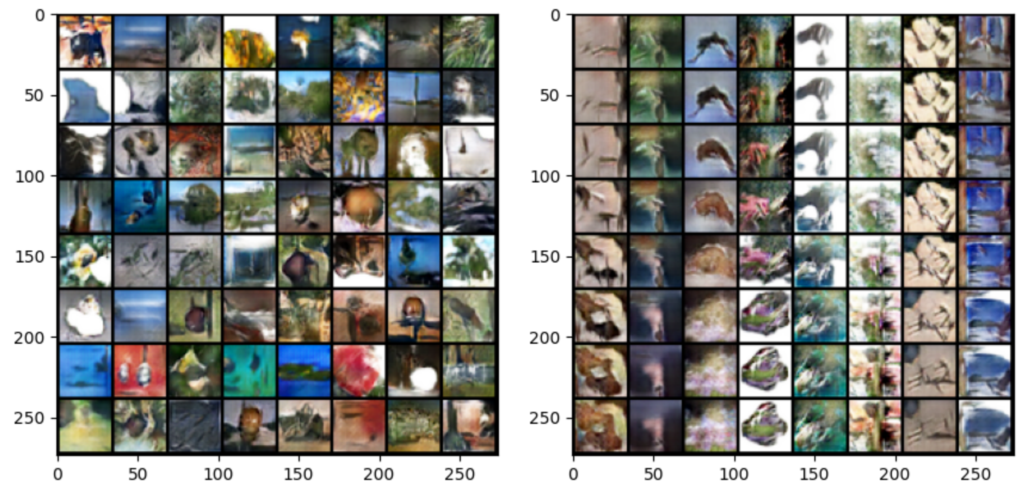

Figure 4: Non-cherry picked samples (left) and interpolants between points in latent space (right)

Fig. 4 shows a diverse sample of images. This is likely due to the latent dimension value of 100, giving the generator a large space to explore. The interpolants between points in latent space show a gradual transition from one point to another. The high diversity of generated images is further evidenced by the mean LPIPS score [8] of 0.686±0.09, which also shows that the images are dissimilar to their nearest neighbours in the training dataset.

6 Limitations

The results look slightly blurry and not very realistic. While some images look clear, even then they resemble abstract art rather than realistic images. Furthermore, the quality of the interpolants is poor and very blurry. An extremely small sample of generated images have an LPIPS score of 0, which means they are identical to images in the training set.

References

[1] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML]. URL: https://arxiv.org/abs/1701.07875.

[3] PyTorch Contributors. *StepLR - Learning Rate Scheduler*. 2023. URL: https://pytorch.org/docs/stable/_modules/torch/optim/lr_scheduler.html#StepLR.

[4] Ishaan Gulrajani et al. *Improved Training of Wasserstein GANs*. 2017. arXiv: 1704.00028 [cs.LG]. URL: https://arxiv.org/abs/1704.00028.

[5] Martin Heusel et al. *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. 2018. arXiv: 1706.08500 [cs.LG]. URL: https://arxiv.org/abs/1706.08500.

[6] Jie Hu et al. *Squeeze-and-Excitation Networks*. 2019. arXiv: 1709.01507 [cs.CV]. URL: https://arxiv.org/abs/1709.01507.

[7] Mark Sandler et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. arXiv: 1801.04381 [cs.CV]. URL: https://arxiv.org/abs/1801.04381.

[8] Richard Zhang et al. *The Unreasonable Effectiveness of Deep Features as a Perceptual Metric*. 2018. arXiv: 1801.03924 [cs.CV]. URL: https://arxiv.org/abs/1801.03924.