Generating blood cells with ML

José Miguel Matos[1]

Department of Informatics Engineering of the University of Coimbra, Coimbra, Portugal

Abstract. This work explores the generation of synthetic low resolution blood cell images using generative machine learning models, including the development and comparison of: Convolutional Variational Autoencoders (CVAE), Deep Convolutional Generative Adversarial Networks (DCGAN), and Denoising Diffusion Probabilistic Models (DDPM). Experiments are conducted on the BloodMNIST dataset of 28x28 pixel RGB images. After extensive hyperparameter tuning and architectural optimizations, the DDPM achieved the lowest average FID score of 122.26 $\pm$ 0.76, outperforming the DCGAN and the CVAE, demonstrating it superior capability to generate high-quality, realistic and diverse medical images.

Keywords: Generative Models · Diffusion Models · GANs · Autoencoders

1 Introduction

Generative machine learning models are powerful tools, able to learn complex data distributions and come up with realistic data samples that closely resemble real data. Unlike discriminative models, generative ones focus on capturing the statistical structure of the datasets, allowing new data instances to be created. This capacity is particularly important in the medical field where real-world data is often scarce, making it harder to train capable models.

This work focuses on the development and comparison between three different generative machine learning approaches: Autoencoders, Generative Adversarial Networks, and Diffusion Models, when presented with the task of generating novel medical data instances.

2 Dataset

The dataset targeted is the BloodMNIST dataset, part of MedMNIST v2 [1]. It contains over 17,000 RGB 28 by 28 pixel images of blood cells, spanning 8 different blood cell types.

2.1 Considerations

For this work, the whole dataset was used as the training set, so there was no data splitting and all of the instances were available. All training images were normalized to the [-1, 1] range.

3 Methods

3.1 Convolutional Variational Autoencoder (CVAE)

Autoencoders are neural networks with the same number of inputs and outputs, which are trained by minimizing the error between input and output data to learn a latent representation z that can represent the distribution of the training data. It is separable into 2 components: encoder and decoder. The training process forces the network to reconstruct what was fed as input, ensuring the latent space captures essential information from the dataset. One problem is that it tends to learn a non-regularized latent space, which makes it unusable for generation.

Variational Autoencoders (VAE) address that issue and provide greater generation capability by having the encoder output parameters of a pre-defined gaussian distribution that represents the latent space. To generate new instances, vectors are sampled randomly from the distribution given by the encoder and fed to the decoder.

The Convolutional Variational Autoencoder (CVAE) is a variation of a VAE, but with convolutional blocks in both the encoder and decoder, so it is able to better capture patterns in images and therefore have better performance in image processing tasks.

VAE Loss Function The loss function for the VAE has 2 components, defined by the network's objectives. On one side, it needs to be able to reconstruct the input, while on the other it needs to make sure the latent space follows a normal distribution. To make sure inputs are being reconstructed, binary cross-entropy loss was used. To ensure that we are approximating a true and normal distibution, *Kullback-Leibler Divergence* is used. If KL divergence reaches 0, it means that both the distributions are equal, so our objective is to minimize it. To tweak the importance of this parameter in the model's loss function a weight parameter was used as shown in equation 1.

$$\mathcal{L}_{VAE} = \text{BCE}(\hat{x}, x) + \beta \cdot \text{KLD}(q(z|x)||p(z)) \tag{1}$$

3.2 Deep Convolutional Generative Adversarial Network (DCGAN)

Generative Adversarial Networks consist of 2 separate networks, which are trained by playing a min-max game against each other. The Generator produces synthetic data samples, having the goal of fooling the discriminator that they are real. The Discriminator learns about real data and tries not to be fooled by the synthetic samples. A Deep Convolutional GAN [2] incorporates convolutional blocks into both the Generator and the Discriminator to boost capabilities in image processing tasks.

3.3 Denoising Diffusion Probabilistic Model (DDPM)

DDPMs learn to create new data by modeling a gradual process of noise removal. They work by first learning how to systematically add noise to data, then to

reverse this process so new samples are created. It operates through a forward diffusion process where Gaussian noise is progressively added to training data over a series of timesteps until it reaches pure noise, and a reverse denoising process to remove this noise step by step, eventually generating realistic samples from random noise. This process typically uses a U-Net architecture network.

The key insight that separates it from the other two models is that instead of directly learning the complex data distribution, it learns to predict added noise at each step.

3.4 Evaluation

Fréchet Inception Distance (FID) Is a metric to quantitatively evaluate the quality of images generated by generative models by comparing the distributions of real and generated images features, extracted using a pre-trained InceptionV3 network. The implementation used is a wrapper of the *pytorch-fid* library, which allows for in-memory FID computation [3].

Evaluation Approach Each model is evaluated for total of 5 independent runs with different seeds to account for stochasticity in the process. In order to calculate the FID score, 10000 real samples are randomly sampled from the dataset and 10000 synthetic samples are generated by the model. The results presented consist of the average FID score over these 5 runs.

4 Experiments & Results

4.1 CVAE

The CVAE architecture consists of an encoder with three convolutional layers (32,64,128 filters) with ReLU activations, copying the convolutional section that performed best in the TP1 project, followed by fully connected layers that output parameters (mu and logvar) that describe the latent Gaussian distribution. The decoder reverses the process using a linear layer to map latent vectors to a reshaped feature space, then applying transposed convolutional layers to upsample back to 28x28 with 3 channels.

Parameters The starting parameters were a latent space dimension of size 64, a KL loss weight of 1 (equal importance to BCE loss), and 100 epochs of training with a batch size of 128. The optimizer used was Adam with a learning rate of 1e-4.

Experiments The first experiment was to see how the model behaved with different latent space dimensionalities, from which 64 achieved the best results. After this experimentation was made with different *Kullback-Leibler Divergence* weights, giving different importance to the training objectives. A weight of 0.75 compared to the BCE loss yielded the best results from the values tested. Finally, different learning rates were tested. All results are shown in Table 1.

Table 1: CVAE Experiment Results

Experiment	Parameter	Values	Average FID Score	Std. Dev.
Latent Space Dimensionality	Latent Dim	32	255.31	0.91
	Latent Dim	64	**248.83**	0.48
	Latent Dim	128	250.66	0.53
KL Divergence Weight	KL Weight	1	240.20	0.61
	KL Weight	2	261.02	0.45
	KL Weight	0.75	**238.17**	0.46
	KL Weight	0.25	242.01	0.90
Learning Rate	LR	0.001	**170.05**	0.64
	LR	0.005	190.71	0.74

4.2 DCGAN

The architecture implemented follows the original DCGAN architecture [2], with adaptations for 28x28 pixel images. With an extra layer when compared to the CVAE, the Generator transforms a noise vector through transpose convolutional layers (256, 128, 64, 32) with batch normalization and ReLU activations, ending in a Tanh output layer to produce 3 channel normalized RGB images. The Discriminator mirrors this, using regular convolutional layers, with batch normalization and LeakyReLU activations to classify real versus generated images. Comparing to the original, this implementation used fewer layers due to the smaller target image size, but maintains the core architectural principles including the use of strided convolutions instead of pooling, batch normalization in both networks, and the specific activation functions.

Parameters The baseline DCGAN was trained with a latent space of dimension 100, binary cross-entropy loss, over 40 epochs and with a batch size of 128. Both the Generator and Discriminator were optimized with Adam and a learning rate of 1e-3, and are updated once every iteration. The Adam optimizers had a Beta 1 of 0.5 as also specified in the original paper [2] to help stabilize training.

Experiments When training the baseline GAN, high oscillations were noted in generator and discriminator loss, both in early and late stages. To try and fix this, label smoothing was implemented, with real labels as 0.9 and fake labels as 0.1, as it has been shown to be a great regularizer [4]. The label smoothing worked as expected, with loss curves oscillating less; however, it is notable that there is a strong case of discriminator dominance occurring, and to combat it, the generator was trained 3 times per iteration and 5 times per iteration.

As expected, training the generator more often led to a better balance between the 2 networks. Next, the attempt was to reduce the learning rate and see its effects in the training process. Unexpectedly, a lower learning rate resulted in a much more volatile training, where stabilization was hard and the final results were poor compared to the original learning rate, as seen in Figure 1.

All results from experimentation are present in Table 2.

Table 2: DCGAN Experiment Results

Experiment	Parameter	Values	Average FID Score	Std. Dev.
Label Smoothing	Baseline	-	197.06	0.71
	Label Smoothing	0.1	**188.99**	0.62
Generator/Discriminator Ratio	G/D Updates	1:1	188.99	0.62
	G/D Updates	3:1	**166.85**	0.56
	G/D Updates	5:1	246.61	0.31
Learning Rate (G & D)	LR	1e-3	**166.85**	0.56
	LR	5e-4	302.13	0.42
Latent Space Dimensionality	Latent Dim	64	209.50	0.53
	Latent Dim	100	**166.85**	0.56
	Latent Dim	128	176.86	0.62

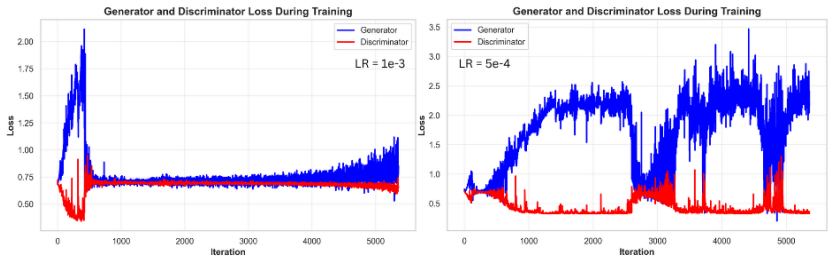

Fig. 1: Effect of learning rate changes in the training of the DCGAN networks: 1e-3 (left) and 5e-4 (right).

4.3 DDPM

The DDPM architecture employs a U-Net structure designed to predict noise at each denoising time step. The network takes as input the image at timestep t and the timestep itself, and outputs the predicted noise to be removed. The time embedding component encodes the timestep t, and is followed by fully connected layers with ReLU activations.

The encoder consists of downsampling blocks with convolutional layers: 64, 128 filters both followed by max pooling, ReLU activations and residual connections. The bottleneck layer between encoder and decoder employs 256 filters, capturing the most essential features in a low resolution.

The decoder mirrors the encoder for upsampling, with skip connections, reducing from 256 to 128, to then 64, while maintaining a symmetric structure to the encoder. A final convolutional layer outputs the predicted noise with the same dimensions as the input image. Compared to the original DDPM, this version utilizes a simpler U-Net with less layers and without self-attention modules [5].

Parameters Initially the DDPM was trained for 100 epochs at 1e-4 learning rate, using an Adam optimizer. The loss used was mean squared error loss. Beta scheduling matches the original paper [5] with 1e-4 minimum beta and 0.02

maximum beta. Decision on the number of timesteps to use was the objective of the first experiment.

Experiments In the original paper, a value of 1000 timesteps is used. To decide on what value to keep, experimentation was done with 4 different values (300,500, 750, and the original 1000). After each model was trained, samples were generated with the same number of timesteps as the training was done, and the results are in Figure 2. A value of T=500 was chosen, as it provided the lowest FID and also the samples which appear closer to the original.

After that an experiment was done with the depth of the U-Net. The baseline U-Net received more depth by receiving another block, so the encoder (and decoder) now has 3 blocks (64,128,256,512 filters), and the bottleneck also was increased to 512 filters. The deeper U-Net performed better, so it was given a larger number of epochs to train, as the last modification, as it has been shown that by training for a larger number of epochs, DDMPs can learn more accurately in less dense regions of the data space, like the background of the images in the dataset. Results for all experiments are present in Table 3.

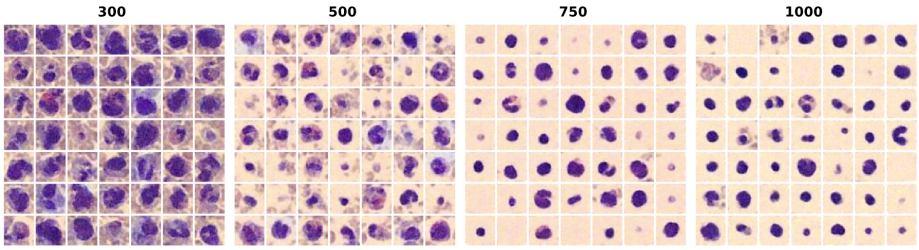

Fig. 2: Samples generated after training of the baseline DDPM with a different number of timesteps. Increasing from 300 up to 1000.

Table 3: DDPM Experiment Results

Experiment	Parameter	Values	Average FID Score	Std. Dev.
No. of Timesteps	T	300	295.70	0.43
	T	500	**238.40**	0.62
	T	750	265.36	0.59
	T	1000	247.78	0.71
U-Net Depth	Depth	Baseline	238.40	0.64
	Depth	Deeper	**202.09**	0.41
Epoch Number	Epochs	100	202.09	0.41
	Epochs	500	**122.26**	0.76

4.4 Summary of Results

After experimenting with all models, samples were generated by the models which achieved the lowest FID score during evaluation. The samples which they

generated are seen in Figure 3 and the FID scores achieved by the models in Table 4.

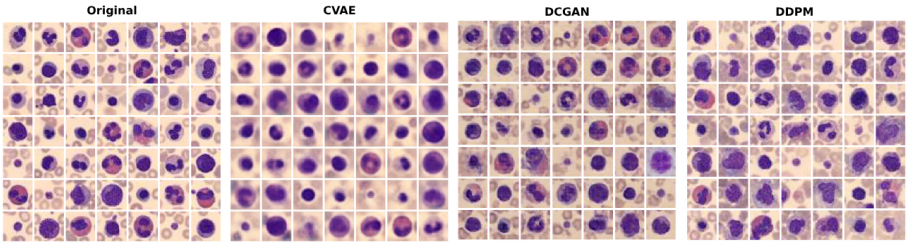

Fig. 3: Samples generated by the model with the lowest FID score for each model type against the original.

Table 4: Best FID scores achieved for different generative models (mean ± std over 5 runs)

Model	**FID Score**
Convolutional Variational Autoencoder	170.05 ± 0.64
Deep Convolutional Generative Adversarial Network	166.85 ± 0.56
Denoising Diffusion Probabilistic Model	**122.26 ± 0.76**

5 Discussion & Conclusion

Analyzing the results, while the FID scores for both the CVAE and the DCGAN are close, visually the samples generated by the DCGAN are much sharper and of much higher fidelity than those generated by the CVAE. The latter lack definition in both the cell and the background of the image, and produce mostly circle shapes, indicating slight mode collapse. The DCGAN and DDPM appear to produce close results when it comes to overall sharpness, but the DDPM generates more varied samples, which suggest it generalizes better to the whole dataset. This is also reflected by its lower FID score.

Experimentation with latent space dimension with the CVAE revealed that 64 dimensions led to the optimal FID score among others, which suggests that bigger is not always better for latent dimensions, and a balance between capturing of data and avoiding overfitting may be optimal. Also, a learning rate of 1e-3 proved optimal to allow our model to perform, heavily reducing it's FID score. A key advantage of CVAEs is their training stability, contrary to GANs. However, their primary weakness in image generation tasks is notable in the samples produced. They lack realism and sharpness and can be almost classified as "blurry".

As expected, the DCGAN presented unique training complexities. The implementation of one side label smoothing proved its effectiveness as a regularizer, preventing the discriminator from overpowering the generator, and stabilizing training. It also is to note that in this case, the generator and discriminator update ratio of 3:1 helped boost performance and lower FID score, which indicated the necessity of training the generator more often to maintain adversarial

equilibrium. Unexpectedly, a lower learning rate resulted in a more instable and worse training. Visual analysis of the samples confirms that DCGANs excel in providing sharp and realistic images when the training process is stable, adding that they are highly sensitive to changes in hyperparameters.

Finally, the DDPM proved heavily influenced by the number of timesteps and the architecture. In the experiments made, a T of 500 proved to be the optimal balance, reducing FID and producing the most realistic samples (Figure 2). Further improvements were made by increasing the depth of the network, demonstrating the need of more filters to better learn the intricate patterns of the dataset. However, the most significant gain was observed by training the network for an extended period of 500 epochs, which dramatically reduced the FID to 122.26 . This highlights that DDPMs benefit from more robust architectures and longer training to accurately learn all regions of the data distribution and generate high-quality synthetic images. Against this, a limitation of the DDPM is its high computational cost, when compared to the other 2 models, particularly during the sampling (inference) phase.

In conclusion, while the DCGAN achieved the lowest FID among the initial runs proving its efficiency, the DDPM with further architectural and training optimizations, ultimately outperformed both the DCGAN and the CVAE, demonstrating its potential for generating highly realistic blood cell images, even in environments with low data resolution. This highlights the importance models like DDPM can have in tasks like data augmentation in a medical setting.

References

1. Yang, J., Shi, R., Wei, D., Liu, Z., Zhao, L., Ke, B., Pfister, H., Ni, B.: MedMNIST v2-A large-scale lightweight benchmark for 2D and 3D biomedical image classification. Scientific Data **10**(1), 41 (2023)
2. Radford, A., Metz, L., Chintala, S.: Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv preprint arXiv:1511.06434 (2015)
3. vict0rsch: pytorch-fid-wrapper: A simple wrapper around pytorch-fid for in-memory FID computation. GitHub repository. `https://github.com/vict0rsch/pytorch-fid-wrapper`. Last accessed 1 Jun 2025
4. Goodfellow, I.: NIPS 2016 Tutorial: Generative Adversarial Networks. arXiv preprint arXiv:1701.00160 (2017)
5. Ho, J., Jain, A., Abbeel, P.: Denoising Diffusion Probabilistic Models. In: Advances in Neural Information Processing Systems 33 (NeurIPS 2020), pp. 6840–6851 (2020)