Advanced Machine Learning: Optimizing Image Classification with Generative Model-Driven Data Augmentation

Nicoli Leal$^{1[2021272755]}$ and Stefano Trenti$^{2[2023252452]}$

University of Coimbra, PT

1 Introduction

This report presents the main methods, results, and analysis for the second practical assignment within the scope of the Advanced Machine Learning course. The assignment aimed to introduce the concept and applicability of generative models within computational learning, with a special focus on image processing.

A significant challenge identified in the previous project was the limited amount of data provided, which is insufficient for training deep neural networks for image classification effectively, as these models typically require large datasets to perform well. To address this issue, the primary objective is to enhance the performance of a pre-determined image classifier by augmenting the dataset with images generated by generative models. Specifically, we will use the dataset provided by the supervising professor and augment it with synthetic images generated by Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs). As an extra, we also implemented Denoising Diffusion Probabilistic Models (DDPM).

In this report, we will discuss the implementation and evaluation of these generative models, assessing their reliability and effectiveness in generating high-quality images to improve the performance of the classifier.

2 Experimental Setup

2.1 Dataset Description

The dataset utilized in this project is derived from the "Traffic Sign Dataset", originally comprising 54 distinct classes. However, for the purposes of this study, only 10 classes will be analyzed.

The images within the dataset have a resolution of 75x75 pixels and are organized into two main folders: "Train" and "Test". The "Test" folder, containing 340 unlabeled images, was specifically allocated for evaluating the performance of the trained CNN model on kaggle. On the other hand, the "Train" folder, comprising 277 images, is subdivided into folders corresponding to each class. These images were utilized for training the generative models developed by us.

In order to solve the lack of diversity in the dataset, we applied data augmentation to compensate for the limited data, mitigate overfitting, and improve model robustness. The data augmentation pipeline, defined using the Albumentations library, enhances both the diversity and robustness of the training dataset. This approach ensures that the model is exposed to a wide variety of image transformations, promoting better generalization and performance.

The Albumentations transform pipeline enhances image diversity for training by applying several augmentations. It rotates images up to 15 degrees with border reflection, ensuring no content loss. ColorJitter introduces slight random variations in brightness, contrast, saturation, and hue. Perspective transformation adjusts image corners by 5% to 10% while maintaining size and reflecting borders. Finally, ToTensorV2 converts images into PyTorch tensors. Each transformation is applied with a 100% probability, ensuring consistent augmentation across all images.

An enlargement factor was also defined to control the size of the final available dataset. Different values were tested but the best compromise in terms of results and processing time was achieved with the augmented dataset having 3 times the original size with a final number of 3108 images available.

However, the augmentation pipeline was not used during the VAE training process as it was found to negatively impact performance. The reason for that can be related with the fact that Augmented images can introduce variations that the VAE struggles to learn, making it harder for the model to capture true underlying data distribution.

2.2 Model Design and Evaluation

Among various architectures and hyperparameters, this section will present and explain the configurations that achieved the best performance in image generation for DCGANs, VAEs, and DDPMs. Initially, the models started with simpler, more common architectures, and complexity was incrementally added throughout the process until achieving the optimal results. Each choice will be thoroughly explained here, with the corresponding results discussed in the following section. Conditional VAEs and Conditional DCGANs were implemented and tested but due to their higher complexity (both of conditional inputs and latent space description) and their difficulties with mode collapse and training instability were very hard to tune considering our computational resources and time constraints. Their lack of performance when comparing them to their per-class counterpart was the determining factor for our choice in avoiding the conditional variants of these architectures.

Variational Autoencoder Autoenconders (AEs) and Variational Autoenconders (VAEs) are both neural network architecture used for unsupervised learning and data compressions. They consist of an enconder and a decoder, where the encoder compresses input data into a latent representation, and the decoder reconstructs the original input from this representation. The VAEs, however, extend AEs by imposing a probabilistic structure on the latent space and it allows smoother and more structured latent representations, making them well-suited for tasks such as generative modeling and data synthesis.

The encoder part of the network consists of six convolutional layers, progressively downsampling the input image to extract features. The sizes of the convolutional filters increase from 3 channels to 1024 channels, capturing increasingly abstract representations of the input image. The final layer flattens the feature maps and passes them through fully connected layers to compute the mean and logarithm of the variance of the latent distribution. The decoder part of the network mirrors the encoder's structure, using transposed convolutional layers to upsample the latent representation and reconstruct the original image (3x64x64). The kernel size was set as 3, the stride as 2 and the padding 1.

The activation functions chosen were: (i) LeakyReLU, (ii) SELU, and (iii) Sigmoid. LeakyReLU was applied between the convolutional layers in both the enconder and decoder, with a small negative slope of 0.02, due its ability to mitigate the vanishing gradient problem. By allowing a small, non-zero gradiente when the input is negative it helps prevent dead neurons during training, unlike traditional ReLU. SELU is utilized after the fully connected layers in the Encoder, and helps to stabilize the network's activations and accelerate convergence. Finally, the Sigmoid activation function is employed at the end of the Decoder to ensure the output pixel values are constrained between 0 and 1, making it suitable for image reconstruction tasks.

For the loss function a combination of two components were used: the reconstruction loss (BCE) and the Kullback-Leibler Divergence (KLD). While BCE measures the difference between the reconstructed output and the original input through Mean Squared Error (MSE), KLD measures the divergence between the learned latent distribution and a predefined prior distribution (usually a standard Gaussian distribution). By combining these two components yields the total loss function that balances reconstruction fidelity with regularization of the latent space.

The training process involved using the Adam optimizer with a learning rate of 0.0001 for training each class separately. Each training session lasted for 10,000 epochs, resulting in the creation of 10 different models, one for each class. These trained models were saved and later reloaded for the image generation phase. During the image generation process, the decoder component of each loaded model received a random vector sized as Nx512. This random vector was used to generate N samples, each representing a generated image.

The table(1) summarizes the architecture used.

Deep Convolutional Generative Adversarial Network Generative Adversarial Networks (GANs) and Deep Convolutional Generative Adversarial Networks (DCGANs) are both frameworks for training generative models, but they have some key differences in their architecture and applications. A GAN consists of two neural networks: a generator and a discriminator. The DCGANs, however, is a specific type of GAN that incorporates convolutional neural networks (CNN) in both generator and discriminator. Based on the literature, DCGANs produce higher quality images and more stable training compared to basic GANs, especially for image generation tasks.

In this sense, the DCGAN architecture implemented consists of a generator and a discriminator, both implemented using convolutional layers. The Generator features five transposed convolutional layers (ConvTranspose2d), progressively increasing the spatial resolution from a latent vector of 256 to a

Component	**Layer Type**	**Output Size/Channels**
Encoder		
Layer 1	Conv2d, LeakyReLU, BatchNorm2d	32
Layer 2	Conv2d, LeakyReLU, BatchNorm2d	64
Layer 3	Conv2d, LeakyReLU, BatchNorm2d	128
Layer 4	Conv2d, LeakyReLU, BatchNorm2d	256
Layer 5	Conv2d, LeakyReLU, BatchNorm2d	512
Layer 6	Conv2d, LeakyReLU, BatchNorm2d	1024
Flatten	Flatten	4096
Layer 7	Linear, SELU, BatchNorm1d	1024
Output Layer	Linear (mu, log_var)	512
Decoder		
Layer 1	Linear, SELU	4096
Layer 2	ConvTranspose2d, LeakyReLU, BatchNorm2d	512
Layer 3	ConvTranspose2d, LeakyReLU, BatchNorm2d	256
Layer 4	ConvTranspose2d, LeakyReLU, BatchNorm2d	128
Layer 5	ConvTranspose2d, LeakyReLU, BatchNorm2d	64
Layer 6	ConvTranspose2d, LeakyReLU, BatchNorm2d	32
Output Layer	ConvTranspose2d, Sigmoid	3

Table 1: Summary of the Encoder and Decoder Architecture

64x64 image with 3 color channels (3x64x64), using batch normalization and ReLU activations at each step. The final layer uses a Tanh activation to scale the output image pixels between -1 and 1. The Discriminator employs five convolutional layers (Conv2d) to downsample the input image, with feature map sizes doubling at each layer and using batch normalization, LeakyReLU activations, and dropout of 0.2 to improve robustness. The final layer produces a single scalar output with a Sigmoid activation, indicating the probability of the input image being real or fake.

The weights were manually initialized in order to improve the model's convergence during training, ensuring stability. For that we defined the weights with a normal distribution, having mean of 0.0 and a standard deviation of 0.02 for convolutional layer, and mean of 1.0 and standard deviation of 0.02 for batch normalization layers. It was also implemented a technique to introduce noise into labels, potentially making the model more robust by preventing overfitting.

Binary Cross-Entropy (BCE) was used as the loss function for both the generator and discriminator in the training process. The Adam optimizer was employed with a learning rate of 0.0002, along with beta1 set to 0.5 and beta2 set to 0.999, to balance the influence of past gradients and stabilize the training. The model was trained in a per-class manner, with each training session consisting of 10,000 epochs to ensure thorough learning of the data distribution for each class.

The table(2) summarizes the architecture used, with FEATURES_GEN and FEATURES_DIS set as 64.

Component	**Layer Type**	**Output Size/Channels**
Generator		
Layer 1	ConvTranspose2d, BatchNorm2d, ReLU	`FEATURES_GEN * 8`
Layer 2	ConvTranspose2d, BatchNorm2d, ReLU	`FEATURES_GEN * 4`
Layer 3	ConvTranspose2d, BatchNorm2d, ReLU	`FEATURES_GEN * 2`
Layer 4	ConvTranspose2d, BatchNorm2d, ReLU	`FEATURES_GEN`
Output Layer	ConvTranspose2d, Tanh	3
Discriminator		
Layer 1	Conv2d, LeakyReLU, Dropout	`FEATURES_DIS`
Layer 2	Conv2d, BatchNorm2d, LeakyReLU, Dropout	`FEATURES_DIS * 2`
Layer 3	Conv2d, BatchNorm2d, LeakyReLU, Dropout	`FEATURES_DIS * 4`
Layer 4	Conv2d, BatchNorm2d, LeakyReLU, Dropout	`FEATURES_DIS * 8`
Output Layer	Conv2d, Sigmoid	1

Table 2: Summary of the Generator and Discriminator Architecture

Denoising Diffusion Probabilistic Model Denoising Diffusion Probabilistic Models (DDPMs) are a specialized implementation of diffusion models that utilize a probabilistic framework to formalize the denoising process. This framework ensures that each denoising step is modeled as a probabilistic distribution, making DDPMs distinct in their approach to noise addition, reverse process, and training objectives. Due to their rigorous, effective methodology and easier implementation with respect to other diffusion models, we have chosen to use DDPMs for this project with the support of the *diffusers* package.

To implement this we used *UNet2DModel*, which is a specialized U-Net architecture designed for diffusion models. The model was initialized with specific parameters: a sample image size of 64x64 pixels, three input and output channels for RGB images, and two layers per block. The model's architecture includes a series of down-sampling and up-sampling blocks, with output channels increasing from 64 to 256 to capture fine details and broader features. The choice of channel sizes was conditioned by the amount of GPU memory required for larger channel sizes of the models. Attention mechanisms are incorporated in specific down-sampling and up-sampling blocks to enhance feature representation. These choices ensure a robust model capable of effectively handling the complexities of the diffusion process.

The training process was conducted per class over 500 epochs. The AdamW optimizer, which combines Adam with weight decay, was used with a learning rate of 0.0001. The Mean Squared Error (MSE) loss function was employed to measure the difference between the predicted noise and the actual noise added to the images. A color histogram based loss was also implemented during testing to try and improve the final output color of the generated images but this didn't have much impact on the final result. For each class, a cosine learning rate scheduler with warmup was implemented, featuring 500 warmup steps and total training steps calculated as the number of images per class multiplied by the number of epochs.

An accelerator was used to manage mixed precision and gradient accumulation, enhancing training speed and reducing memory usage. Mixed precision was set to 'fp16', and the gradient accumulation steps were set to 1. A DDPMScheduler was utilized to control noise levels throughout the training timesteps, with the timestep count set to 1000. For generate the images we used a pipeline that encapsulates the model and the noise scheduler to streamline the image generation process.

The table(3) summarizes the architecture use

Layer Type	**Output Channels**	**Details**
Input Layer		
Input	3 (RGB)	Image size: 64x64
Down-sampling Blocks		
DownBlock2D	64	Layers per block: 2
DownBlock2D	64	Layers per block: 2
DownBlock2D	128	Layers per block: 2
DownBlock2D	128	Layers per block: 2
AttnDownBlock2D	256	Layers per block: 2, with attention
DownBlock2D	256	Layers per block: 2
Up-sampling Blocks		
UpBlock2D	256	Layers per block: 2
AttnUpBlock2D	256	Layers per block: 2, with attention
UpBlock2D	128	Layers per block: 2
UpBlock2D	128	Layers per block: 2
UpBlock2D	64	Layers per block: 2
UpBlock2D	64	Layers per block: 2
Output Layer		
Output	3 (RGB)	Image size: 64x64, with Tanh activation

Table 3: UNet2DModel Architecture

Metrics To evaluate the performance of the generated images, and comprehensively assess its quality and reliability, three metrics were chosen: Inception Score (IS), Fréchet Inception Distance (FID), and Structural Similarity index (SSIM). Despite the similarities between IS and FID, such as the fact that both use a pre-trained Inception v3 model to extract features from images, they differ in their approach and focus. Training the CNN model with the generated images and evaluate its performance on real ones also helps to determine how realistic the images are.

The Inception Score measures the quality and diversity of generated images by evaluating the confidence and diversity of the predictions made by an Inception-v3 classifier. For that we imported *Inception-Score* from *torchmetrics.image*, and calculated it per class and then for the whole dataset. The range of values for the IS theoretically falls between 0 and infinity. However, in practice, they are usually bounded within a certain range depending on the dataset. Generally speaking, higher IS values indicate better performance in terms of both image quality and diversity.

The Fréchet Inception Distance, on the other hand, focus on the similarity between the distribution of features in real and generated images, by calculating the Fréchet distance between two Gaussian distribution fitted to the features of real and generated images extracted by an Inception-v3 network. For that we imported *FrechetInceptionDistance* from *torchmetrics.image.fid*, and calculated it per class. The range of values can also falls between 0 and infinity, however higher values indicates, for this metric, a poor performance.

The Structural Similarity index, as the name indicates, evaluated the structural similarity between two images based on luminance, contrast, and strucute, providing a perceptual similarity measure. The values for the ssim are tipically between [-1, +1], where +1 indicates that the 2 given images are very similar or the same while a value of -1 indicates the 2 given images are very different. For that we used *structural_similarity* from *skimage.metrics*, and compute it for all pairs of image between real and generated. The final index, presented later in this report, are the mean of all combination for each class.

Training the CNN The final step involved training the provided CNN using the generated images. For each of the three generative models (GAN, VAE, and DDPM), we generated two datasets consisting of 50 and 100 images each. Additionally, for the GAN and VAE models, a third dataset containing 500 generated images was created. Unfortunately, due to computational limitations, we were unable to generate the 500-image dataset for the diffusion model. Balanced accuracy was chosen as a metric of goodness of our results with respect to the validation set, this is because our starting training set is heavily unbalanced (78 images for class 6 and only 8 images for class 24) and balanced accuracy takes into account the number of samples for each class therefore providing a more fair assessment of the model's ability to correctly identify examples from all classes, not just the majority ones. Still when enlarging our dataset with generated images we chose to add the same amount of new images to each class thus reducing the total class unbalances.

The CNN was subsequently retrained using newly generated images together with the original training set and evaluated on the provided test set through Kaggle. Additionally, the pre trained baseline model provided was also evaluated on Kaggle for comparison.

3 Experimental results

The following sections will explore the challenges and results associated with each generative model implemented. We will present the scores for the evaluation metrics, as well as the accuracy achieved by the CNN when trained with the newly augmented datasets.

3.1 Variational Autoencoder

Upon analyzing the images generated by the VAE models, it is evident that the model succeeded in its task. Generally speaking, through visual inspection, the VAE model showed better results and more consistent when compared with the other models. However, the inability to augment data for training the VAE resulted in generated images that are quite similar to each other, and to the original ones, leading to a dataset with low diversity. This is a typical issue with Autoencoders when having a small training set. The figure(1) shows the best(left) and worst(right) results for the one we considered the best (top) and worst(bottom) class.

The table below presents the scores for each class. Lower values for the Inception Score (IS) indicate a lack of diversity in the dataset. However, the Fréchet Inception Distance (FID) suggests that the generated images are similar to the original dataset. By analyzing these results, we can conclude that diversity may be also a problem associated with the type of images we are dealing with. The SSIM indicates that there are a significant perceptual discrepancy between the real and generated images, likely due to noise.

CLASSES	12	13	24	38	39	44	46	49	50	6
INCEPTION SCORE	1.87	1.78	1.84	2.11	2.06	2.17	1.91	2.13	1.62	1.67
FRECHET INCEPTION DISTANCE	0.96	0.86	0.87	0.82	0.81	0.87	0.84	0.82	0.83	0.75
STRUCTURAL SIMILARITY INDEX	0.06	0.08	0.07	0.05	0.06	0.06	0.06	0.06	0.06	0.05

Table 4: Evaluation of the generated images with VAE

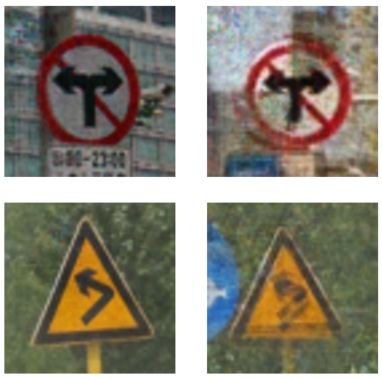

Fig. 1: Best and worst images for VAE

3.2 Deep Convolutional Generative Adversarial Network

The GAN results appeared perceptually worse compared to the VAE, as evident when examining figure(2), yet they exhibited higher diversity. Shape consistency, noise and lack of detail are the more evident issues. However, despite these perceptual differences, the metrics indicated similar issues to those observed with the VAE.

CLASSES	12	13	24	38	39	44	46	49	50	6
INCEPTION SCORE	1.90	1.66	1.81	1.86	1.80	2.05	2.28	2.29	1.54	1.55
FRECHET INCEPTION DISTANCE	1.17	1.05	0.98	0.90	0.88	0.89	0.86	0.84	0.84	0.76
STRUCTURAL SIMILARITY INDEX	0.05	0.05	0.06	0.04	0.05	0.05	0.05	0.05	0.05	0.04

Table 5: Evaluation of the generated images with DCGAN

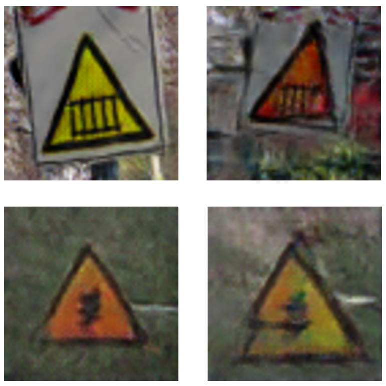

Fig. 2: Best and worst images for DCGAN

3.3 Denoising Diffusion Probabilistic Model

The resulting images generated by the DDPM model show good details and very low noise when looking at the smaller details of the road signs and a good image to image variance. These images however show much less consistency when looking at the shape and overall color of the road signs, some results even lacking a clear image of a road sign. This lack of consistency is also evident by looking at the testing result where, the images generated by this model achieved a worse result when compared to other models with the same amount of samples both in the kaggle score and in the validation set metrics.

CLASSES	12	13	24	38	39	44	46	49	50	6
INCEPTION SCORE	2.00	2.06	2.01	2.17	2.29	2.24	2.28	2.10	1.90	2.07
FRECHET INCEPTION DISTANCE	1.18	1.04	1.01	0.89	0.83	0.79	0.77	0.74	0.77	0.70
STRUCTURAL SIMILARITY INDEX	0.01	0.01	0.02	0.02	0.01	0.02	0.02	0.02	0.02	0.02

Table 6: Evaluation of the generated images with DDPM

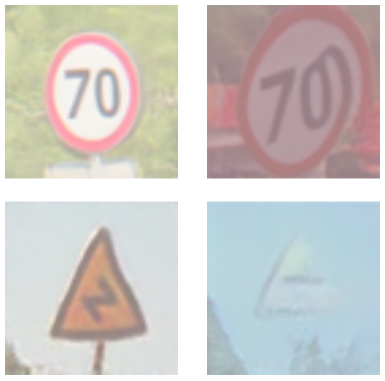

Fig. 3: Best and worst images for DDPM

3.4 Convolutional Neural Network Performance

The results observed and commented in the previous section were confirmed when we evaluate the CNN using the generated images. The baseline model given by the responsible professor achieved a balanced accuracy of 55% with test 28 images from the original dataset, and a kaggle score of 0.2676.

The dataset augmentation proved effective for all three implemented models. The VAE demonstrated higher accuracy for the initial test sets: 98.33% for 78 test images, 100% for 128 test images, and 100% for 528 test images. However, the Kaggle scores were 0.5909, 0.5513, and 0.6050, respectively, indicating potential overfitting.

Similarly, the GAN showed effectiveness with accuracies of 72.98% for 78 test images, 97.41% for 128 test images, and 100% for 528 test images. Despite lower initial test set values, the Kaggle scores indicated better consistency and generalization, achieving 0.5067, 0.6476, and 0.6670, respectively.

In comparison, the DDPM exhibited the worst performance but still outperformed the baseline model. It achieved balanced accuracies of 82.84% for 78 test images and 90.84% for 128 test images, with Kaggle scores of 0.3968 and 0.5536, respectively.

The results are graphically illustrated below. It is evident that both the DCGAN and DDPM models exhibit improvement as the number of images used increases. However, the VAE model appears to plateau, indicating a potential limitation associated with the lack of diversity in the generated images. This stagnation in performance may be attributed to the model's struggle to capture diverse image representations. The values in the graph serve as a reflection of the overall quality of the generated images and their ability to generalize across different datasets.

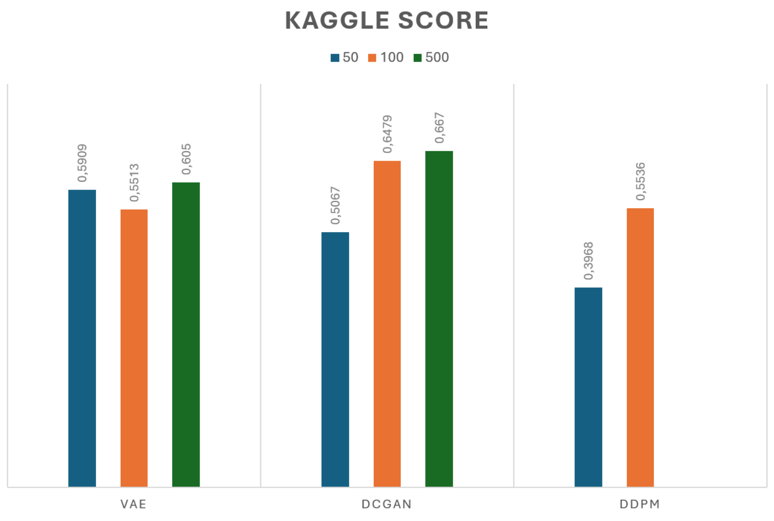

Fig. 4: Kaggle Scores

4 Discussion and Conclusion

Overall, the generative models demonstrated efficiency in image generation, each with specific characteristics associated with its architecture that significantly improved the performance of the CNN. Regarding the quality of the generated images, the Inception Score indicated a lack of diversity for all three models, possibly due to the inherent similarity in the street sign images used as input. Moreover, a good Fréchet Inception Distance (FID) but a poor Structural Similarity Index (SSIM) suggests that while the generated images closely match the statistical distribution of real images, they may still lack visual similarity in terms of structural details, textures, or perceptual quality. In essence, although the generated images may have similar statistical properties to real images, they may still exhibit noticeable differences in appearance, as indicated by the low SSIM score. These findings highlight the need for further exploration and refinement in generative models to achieve both statistical fidelity and visual realism in image generation tasks.

5 References

- Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- An Introduction to Variational Autoencoders
- Denoising Diffusion Probabilistic Models