Conference Paper Title

Abelino Sepúlveda Estrada
Universidad EAFIT
Medellin, Colombia
asepulvede@eafit.edu.co

Olga Lucía Quintero Montoya
Universidad EAFIT
Medellin, Colombia
oquinte1@eafit.edu.co

***Abstract*—This study explores popular machine learning models, including autoencoders, convolutional neural networks (CNNs), and generative adversarial networks (GANs). The first part focuses on utilizing autoencoders to uncover patterns in hidden layers and apply them to practical problems, emphasizing dimensionality reduction. Next, a LeNet-5 CNN is trained and tested on the MNIST dataset for handwritten digit recognition. The study also incorporates GANs, using convolutional layers to generate realistic images and perform style transfer. Through a series of experiments, the research underscores the importance of selecting the appropriate model for specific application requirements, emphasizing the integration of machine learning into real-world problem-solving scenarios.**

***Index Terms*—Autoenconder, Convolutional Neural Networks, LeNet 5, Generative Adversarial Networks, MNIST**

I. Introduction

Numerous algorithms and architectures have emerged in the dynamic fields of artificial intelligence and machine learning to handle challenging data-related tasks. The use of autoencoders is one such method. A group of neural networks called autoencoders is essential for feature extraction and data compression. They accomplish this by shrinking the size of the data (encoding) and then expanding it back to its original dimensions (decoding). This procedure enables the extraction of valuable features from the data in addition to aiding in data compression. When discussing autoencoders, there are two essential aspects to take into account: Autoencoding in low dimensions, which concentrates on effectively representing data in a low-dimensional space, and Autoencoding in high dimensions, which increases the dimensionality of the problem, allowing for more prosperous feature extraction.

Convolutional Neural Networks (CNNs) are a noteworthy neural network architecture that has significantly impacted computer vision. LeNet-5, a groundbreaking model created for handwritten digit recognition, is one of the notable examples of CNNs. LeNet-5 is a great candidate for tasks like character recognition because its success is closely related to its capacity to quickly process and analyze image data. CNNs like LeNet-5 display remarkable performance and accuracy when paired with datasets like MNIST, which contain an extensive collection of handwritten digits, demonstrating the effectiveness of deep learning for image classification tasks.

Generative Adversarial Networks (GANs), which are generative models, have attracted much interest. GANs comprise two neural networks engaging in adversarial training: discriminators and generators. This dynamic encourages the production of synthetic data that closely resembles real-world data distributions. GANs have proven to be a flexible tool in the AI toolbox, finding use in various fields such as image synthesis, data augmentation, and style transfer.

The Style-Based GAN architecture (StyleGAN) advances GAN technology. By introducing a novel method of managing the creation process, StyleGAN makes it possible to manipulate particular attributes of produced images. StyleGAN provides unmatched control and creativity in producing realistic and varied graphics by separating the style and content of photos.

The incorporation of these many neural network architectures into hybrid supervised algorithms is now what unites them. These algorithms combine the powers of autoencoders, CNNs, GANs, and other neural network designs to tackle challenging tasks using various supervised learning methods. The robustness and adaptability of AI systems are improved by this integration, opening the door for more effective and potent solutions across various applications. We will go into greater detail about these approaches in the following sections, examining their distinct benefits and potential uses in the constantly developing field of artificial intelligence.

The rest of this article has the following structure. We begin in section 2 with the methodology, where we define the architectures, the data of the problems, and experiments. Section 3 will present the results of each model's implementation, testing, and validation. In section 4, a discussion of the results will be presented. In section 5, the conclusions will be presented. Finally, we have the references.

II. Methodology

A. Autoencoder

A crucial architectural idea called the Autoencoder balances dimensional reduction and feature extraction. This method, known as nonlinear principal component analysis, provides dimensionality reduction and expansion flexibility to maximize information retention from input data. Autoencoders are crucial for feature extraction because they are skilled at mapping nonlinear relationships between original and modified data. Understanding neural networks (NNs) is crucial before getting into the specifics of autoencoders. NNs comprise successive layers of perceptrons or neurons, each processing input data or the layer's output before it. These layers, which consist of the input layer (which receives starting data), the output layer (which produces responses), and hidden layers (which

are intermediary layers between input and output), linearly integrate stimuli and may alter dimensionality using weight matrices.

NNs may model input data as a function of itself, which may seem counterintuitive but is functional. Although NNs commonly model variables dependent on input data, this capability is not limited to this. By using fewer perceptrons than input features in a single hidden layer, this capacity aids dimensionality reduction, and by using more perceptrons than input features in this hidden layer, it supports feature extraction. Figure **??** shows how this hidden layer, known as the code, captures the core of the input by encoding data in the input layer and decoding it in the output layer. The paradigms for data processing and transformation are redefined by these numerous uses of autoencoders in the context of neural networks.

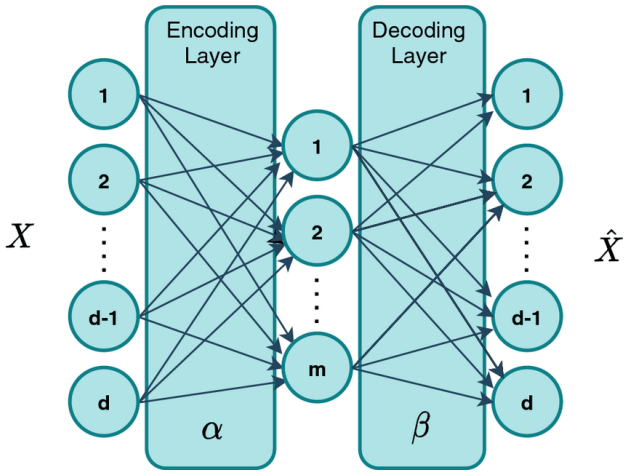


Fig. 1: Autoencoder Architecture Link to source

1) Data: The data selected consists of three input features and an output with 3001 observations of a dynamic phenomenon. The nature and the actual context of the data are currently unknown. However, it is observed that all the variables are numerical and that they are on different scales.

As all features have variable scales, the data are taken to the hyperplane $[0, 1]$, i.e., normalized. This is to avoid numerical problems, improve convergence, and make the training less dependent on the initialization of the data.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{1}$$

The equation (1) refers to the normalized data by means of min-max normalization. Where X_{norm} refers to the normalized data, X the original data and X_{min}, X_{max}, the minimum and maximum of the features respectively.

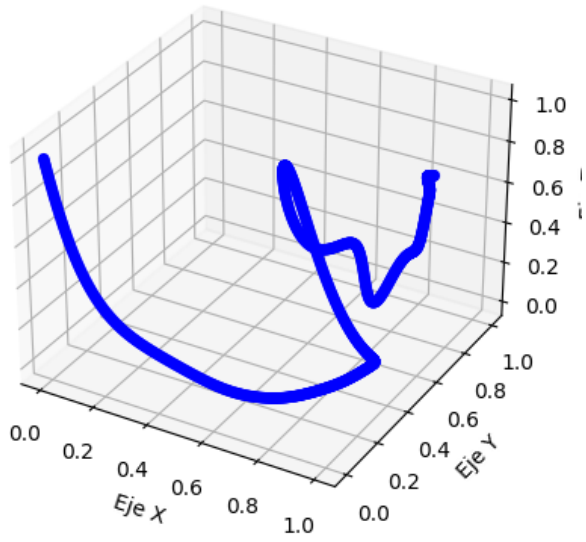


Fig. 2: Data visualization

Subsequently, the data are separated into training, testing, and validation sets in a proportion of $60\%, 20\%$, and 20% of the original data, respectively. This avoids over-fitting and allows us to assess the model's generalization ability to new data. Finally, this partitioning was done based on the uniform distribution, i.e., the probability of selecting each data set was the same to guarantee a higher entropy and that the sets are representative of the general distribution of the original data.

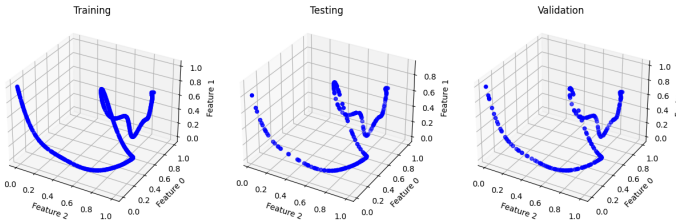


Fig. 3: Set partitioning of data

Figure 2 shows the original data in the hyperplane $[0, 1]$ and Figure 3 shows that the sampling of the different sets has the same behavior as in Figure 2, so we can say that they are a representative sample of the data.

2) Experimentation: The experimentation exercise will be divided into Autocoding in low dimensions, i.e., in the hidden layer using fewer neurons than in the input layer as $X_{norm} \in R^3$, $l_j = [1, 2]$. For Autocoding in high dimensions $l_j = [4, 5]$, it is decided to test with the same increment to check which of the two performs better. The activation functions used are linear for the input and hidden layers and sigmoid for the output layer, with a learning rate of 0.01 and 50 input epochs. Finally, the results of these models will be compared with a model of the Python Keras package.

B. Convolutional Neural Network (CNN): LeNet 5

Convolutional Neural Networks (CNNs) are a pivotal class of deep learning models explicitly tailored for visual data processing tasks like image classification, object detection, and segmentation. They draw inspiration from the human visual system, adeptly capturing local patterns and hierarchies of features in images. This capability has led to groundbreaking advancements in computer vision applications, fundamentally changing how we approach tasks related to visual data.

LeNet-5, a seminal CNN architecture by Yann LeCun and collaborators in the early 1990s, comprises seven layers, including two convolutional layers, two pooling layers, and three fully connected layers. It was meticulously crafted for digit recognition, showcasing CNNs' potential in image classification. LeNet-5 remains a cornerstone in deep learning, paving the way for more sophisticated models. It was initially designed to automate banking processes like check reading, playing a vital role in digitization and remaining a venerated milestone in computer vision and deep learning (see Figure 4).

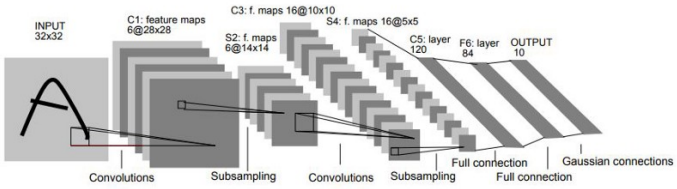


Fig. 4: LeNet 5 architecture Link to source

As mentioned above, the CNN LeNet 5 was developed for digit recognition; for this work, we seek to train a CNN with this architecture with the MNIST database (Modified National Institute of Standards and Technology database), which is a database of 70,000 images of handwritten digits (0-9) in black and white and then try to discriminate a database of our own made in class.

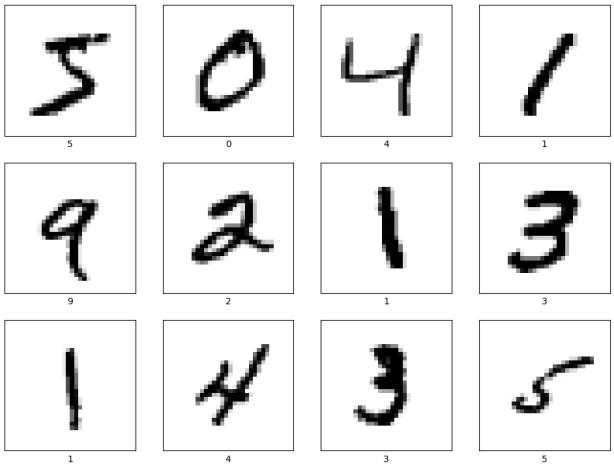


Fig. 5: MNIST digits

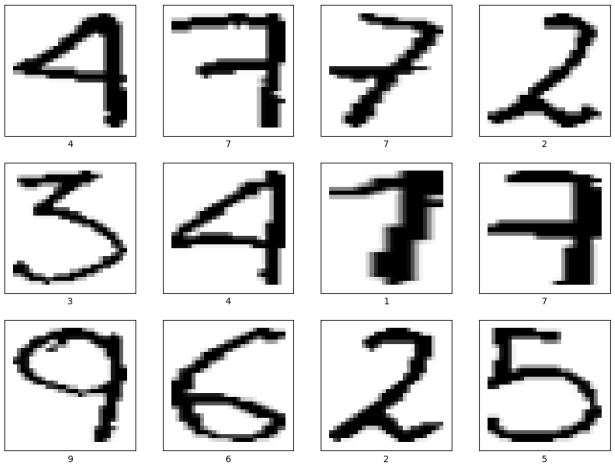


Fig. 6: Course digits

Figures 5 and 6 shows the comparison between the MNIST database and the digits generated by the course.

C. Generative Adversarial Networks (GAN)

The primary goal of GANs is to create data indistinguishable from authentic data. The generator takes random noise and crafts synthetic data (like images or text) resembling real examples. Meanwhile, the discriminator distinguishes between authentic and generated data. These networks engage in constant adversarial competition: as the generator improves, so does the discriminator, and vice versa. This process enhances both networks, yielding highly realistic data.

The architecture of a GAN includes two key components: a generator and a discriminator. The generator uses random noise as input, employing neural network layers (often convolutional or recurrent) to generate data resembling the training set. In contrast, the discriminator assesses real and generated data, producing a probability score indicating data authenticity. During training, the generator aims to deceive the discriminator while the discriminator seeks real discrimination. This iterative adversarial process refines both networks until the generator produces data nearly indistinguishable from real examples, fulfilling GANs' primary objective: realistic data generation (see Figure 7).

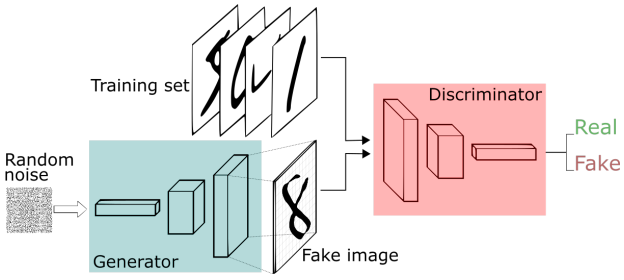


Fig. 7: GAN architecture Link to source

The primary objective is to train a generator network to produce synthetic data nearly indistinguishable from real data. This is achieved through an adversarial process where the

generator creates data, typically starting from random noise, while the discriminator evaluates whether the generated data is real or fake. The generator aims to improve over time by creating increasingly convincing data, while the discriminator strives to become more proficient at distinguishing real from generated data. As this adversarial competition progresses, the generator generates data samples that closely match the distribution of the real training data, making GANs a powerful tool for high-quality data generation and a wide range of applications in fields such as computer vision, natural language processing, and more.

D. Style-based GAN (StyleGAN)

StyleGANs are a groundbreaking development in generative AI and computer vision. They excel at producing incredibly lifelike, high-resolution images, often indistinguishable from photos. StyleGANs are renowned for their ability to generate diverse and imaginative content, making them indispensable in creative fields like art, entertainment, and fashion.

At their core, StyleGANs are a Generative Adversarial Network (GAN) comprising a generator and a discriminator. The generator crafts synthetic data, while the discriminator discerns real from fake. This adversarial interplay continually drives the generator to create more realistic content. What sets StyleGAN apart is its introduction of ”style.” Unlike earlier GANs that manipulated images through fixed input noise, StyleGAN operates with two distinct latent spaces: the latent code, controlling content, and the style code, governing appearance and artistic traits. This separation allows precise control over features like facial expressions and hairstyles.

Moreover, StyleGAN employs a progressive training approach that enhances images incrementally, beginning with low resolution and progressing to higher resolutions. This method ensures the gradual addition of details and contributes significantly to producing high-quality, lifelike images. StyleGANs have diverse applications, from crafting deepfakes to generating realistic artwork and aiding in data augmentation for computer vision tasks, making them an essential tool for AI researchers and creative professionals.

III. Results

A. Autoenconder

Tables I and II show the results obtained from the experimentation, where the training, testing and validation errors for each of the 4 models are shown.

		Own			Library		
l_j	# Parameters	T_{error}	t_{error}	V_{error}	T_{error}	t_{error}	V_{error}
1	7	0.296	0.295	0.27	0.047	0.043	0.032
2	10	0.251	0.248	0.275	0.013	0.01	0.004

TABLE I: Results for autoencoder in low dimensions

		Own			Library		
l_j	# Parameters	T_{error}	t_{error}	V_{error}	T_{error}	t_{error}	V_{error}
4	16	0.275	0.273	0.279	0.008	0.005	0.001
5	19	0.296	0.296	0.263	0.007	0.005	0.001

TABLE II: Results for autoencoder in high dimensions

The Figures 8, 9 and 10 show the obtained results for the best model, i.e., the high-dimensional autoencoder with five neurons in the hidden layer (extract features by dimensionality broadening). Figure 8 shows the behavior of the testing and training errors across epochs and the validation error. Figure 9 compares the density distributions of the real data and those obtained by my model and the Keras model (TensorFlow). Finally, Figure 10 shows the data in a 3D graph where the same results are compared.

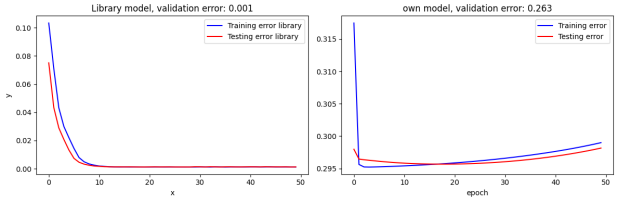


Fig. 8: Errors

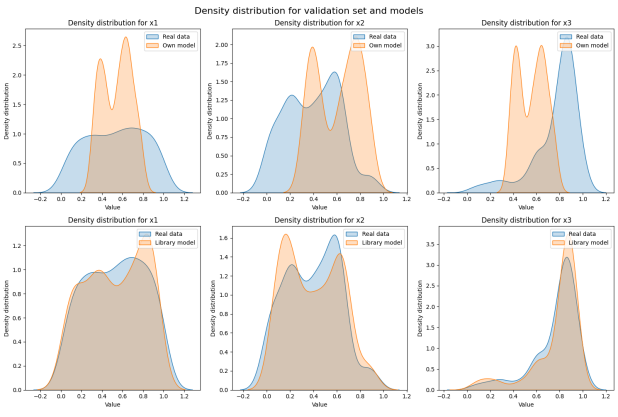


Fig. 9: Density distributions

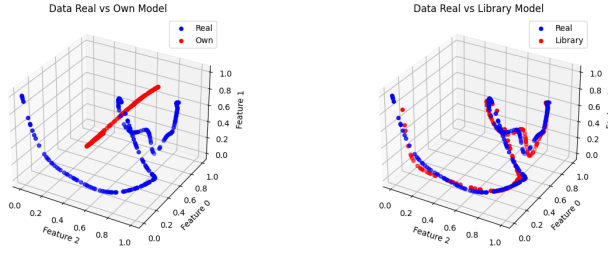


Fig. 10: Desired vs Model output

B. LeNet 5

In Figure 11, we track the accuracy, revealing how the model's performance gradually improves as it learns from the training data, demonstrating its ability to discern complex patterns. Meanwhile, Figure 12 focuses on precision, a vital metric for applications with significant consequences for false positives. Precision's trajectory throughout training and testing offers insights into the model's capacity to reduce errors and false alarms, ensuring its real-world reliability.

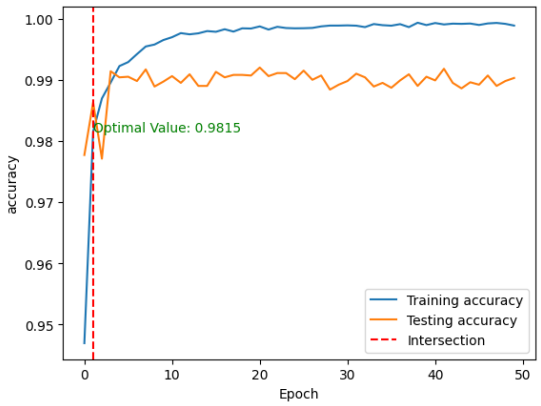


Fig. 11: Accuracy of training and testing through epoocs

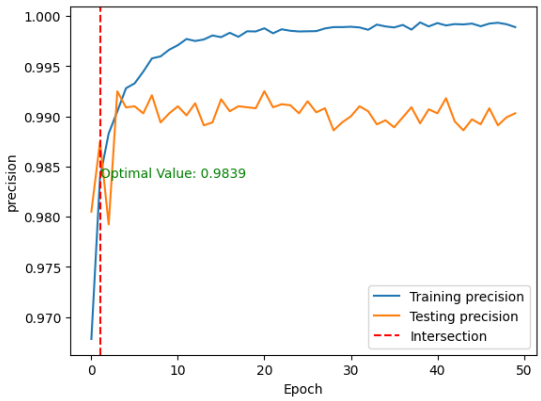


Fig. 12: Precision of training and testing through epoocs

Figure 13 shows the predictions made to certain digits of the validation set. Figure 14 shows the confusion matrix of the testing predictions. The values represent the proportion of the predictions made to certain digits of the validation set. Analogous to Figures 15 and 16

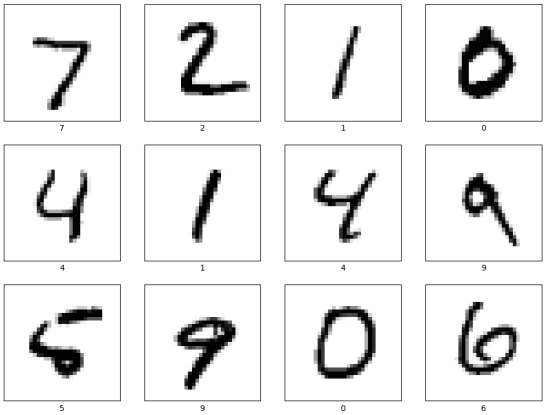


Fig. 13: Predicted values for testing set

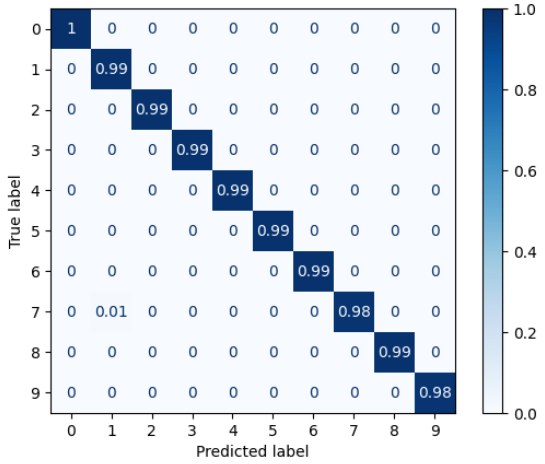


Fig. 14: Confusion matrix for testing set

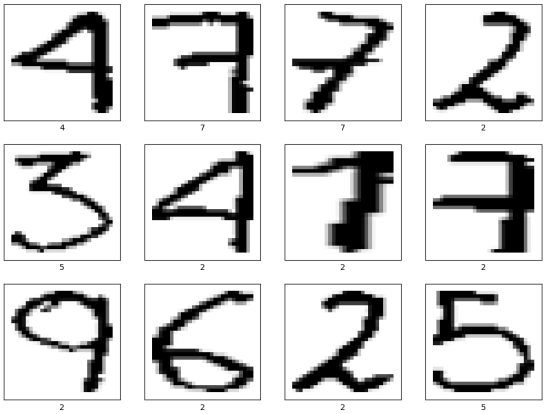


Fig. 15: Predicted values for course dataset

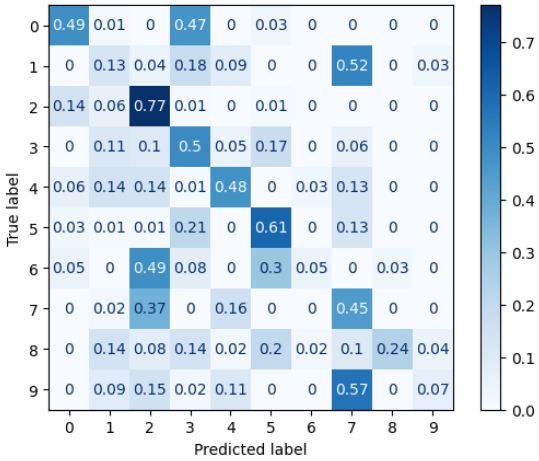


Fig. 16: Confusion matrix for course dataset

C. GANs

The GAN was trained using the MNIST database over 50 epochs. Figures 17-22 illustrate the evolving quality of generated digits at epochs 1, 10, 20, 30, 40, and 50. Initially, at epoch 1, the GAN produced noisy and abstract digits, but as training progressed, it steadily improved its ability to generate more realistic and recognizable handwritten digits. By epoch 50, the GAN reached its peak performance, generating digits that closely resembled those in the MNIST dataset. These snapshots showcase the GAN's learning process, highlighting its capacity to capture intricate patterns and refine its generative output with prolonged training.

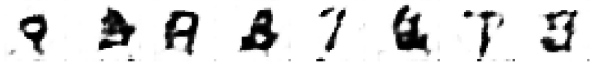


Fig. 17: Digits generated by GAN epoch 1

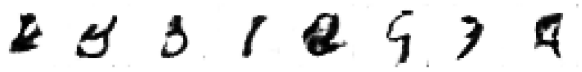


Fig. 18: Digits generated by GAN epoch 10

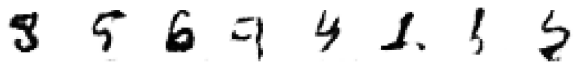


Fig. 19: Digits generated by GAN epoch 20

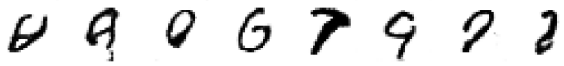


Fig. 20: Digits generated by GAN epoch 30

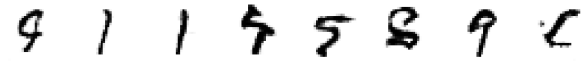


Fig. 21: Digits generated by GAN epoch 40

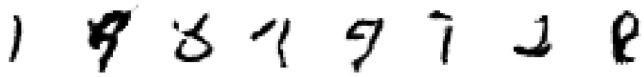


Fig. 22: Digits generated by GAN epoch 50

The Figure 23 tracks the generator and discriminator loss functions across training epochs.

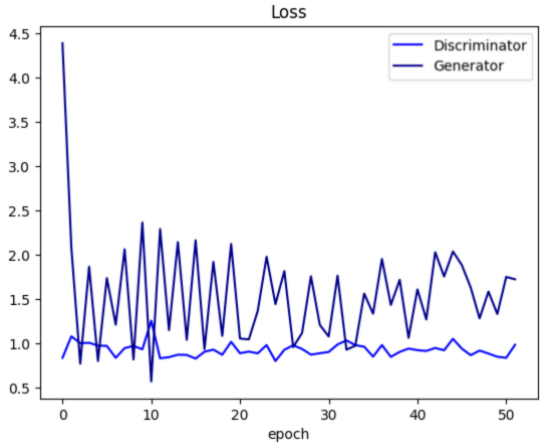


Fig. 23: Generatos

Additionally, the discriminator received the course digits dataset to classify it as authentic or fake data. The results obtained are presented in Table III.

Probability	Course Dataset
<0.5	0.38
>0.5	0.62

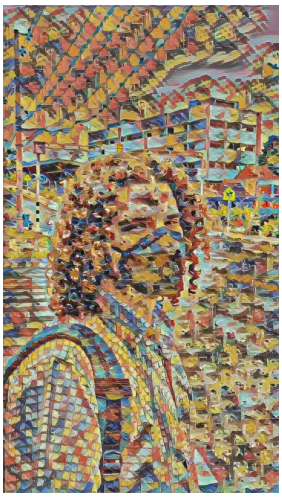
TABLE III: Discriminator of course dataset

D. StyleGAN

For the styleGAN, we used a TensorFlow Hub module called "Arbitrary Image Stylization v1-256/2," a pre-training. From it, we styled the video for the activity "Arbitrary Image Stylization v1-256/2". Figure 24 shows a frame example of the styled video.

(a) Style

(b) Original frame



(c) Styled frame

Fig. 24: Result of stylleGAN for a specific frame

IV. Discussion and Conclusion

A. Autoencoder

As can be seen in the tables I and II, a comparison was made between high and low-dimensional autoencoding to look at the behavior of these autoencoders when there was a variation in the dimensionality, number of parameters, extraction or compression of the data. From the results, it is observed that the best models obtained are those of extraction of characteristics using enlargement of the dimensionality of the problem, that is to say, the autoencoding in high dimensions, being thus, the best model one of 5 neurons in the output layer. However, it is essential to highlight that the improvement among these high dimensional models is not so significant if compared with the low dimensional ones, so it could be said that it could reach a point where increasing the dimensionality could be counterproductive compared to the gain in learning level against the complexity of the model. Additionally, we can observe that the model generated by the Keras package has a better performance than the model itself, so it fits very well with the data. That is to say, it learns the patterns of this one. Finally, it is recommended to make a thorough variation and parameter tuning of the proprietary model to make it competitive against the package.

B. LeNet 5

At the level of training and testing of the CNN LeNet with the MNIST database, we can say that the performance is perfect since the accuracy and precision for training and testing during the epochs are very good, besides the confusion matrix shown in Figure 14 tends to show a ratio of 1 in all correct predictions (the diagonal). However, we can see that the model cannot classify very well with the course database. However, the justification for this problem is apparent. In the figures 5 and 6, we can observe the big difference between the digits of both databases. This is due to how old MNIST is and how the writing process has changed nowadays. Another main reason can be how the course database is structured and digitized.

C. GANs

Figure 22 shows the digits generated in the last training epoch (50), and we can see that the digits are not sufficiently differentiable, which would give us an indication to increase the epochs because, of course, 50 epochs are very few for a GAN. However, it can be said that there are certain digits whose generated pattern can be recognized. Additionally, in figure 23, we can see the discriminator and generator loss, which are sought to decrease during training. We can see that their behavior is wildly oscillating, so it could be expected that it is wholly stabilized in a more significant number of epochs. We can see that despite oscillating, both decrease, i.e., both the generator and the discriminator are meeting their objectives of reducing the discriminator error when evaluating the generated samples and reducing its error in the classification of the samples, respectively.

D. StyleGAN

In figure 24c, we can see that the image style of figure 24a was applied very well, where the patterns are entirely differentiable within the frame and the video, that is, both the person and all the figures and objects can be differentiated within the styled frame.

V. Acknowledgments

I want to thank Miguel Roldan, Paulina Perez, and Ana Sofia Gutierrez for helping to consolidate the course digit database and Olga Lucia Quintero for this work that allows us to learn more about the vast field of applications of neural networks.