A Comparative Analysis of Different Image Generative Models Using Healthy Mango Leaf Dataset

Angeline S. Catapang
ascatapang@up.edu.ph

Patrick John A. Francisco
pafrancisco4@up.edu.ph

Justin Carl C. Sagun
jcsagun@up.edu.ph

Allen Christian P. Segovia
apsegovia@up.edu.ph

Mark Henry S. Alcantara
msalcantara4@up.edu.ph

Jerico Luis A. Ungos
jaungos@up.edu.ph

Andrei Joshua T. Gelaga
atgelaga@up.edu.ph

I. Introduction

A. Background

Generative Artificial Intelligence models, or GenAI models, are a subset of artificial intelligence techniques designed to generate new data based on patterns learned from existing datasets [1]. These models can produce outputs such as images, text, music, and videos by creating content that resembles the data used to train them [2]. GenAI plays a significant role in various applications, including creating realistic images, generating human-like text responses, and producing video game content.

Currently, several types of generative AI models have been developed and introduced to the public. Some notable examples include autoencoders, Generative Adversarial Networks (GANs), and Restricted Boltzmann Machines (RBMs). Autoencoders compress data into a compressed and semantically meaningful form, and then reconstruct it, allowing them to generate new data similar to the input [3]. GANs, on the other hand, consist of two networks, a generator and a discriminator, that compete with each other to produce realistic samples by learning the data distribution [4]. Lastly, RBMs, introduced way back in 1983, are energy-based models that learn through pairwise interactions between visible stochastic units and hidden stochastic units — restricted to have no connections from hidden to hidden or visible to visible nodes [5].

Although GenAI models have improved significantly over the past years, they still face challenges and limitations that hinder them from being more reliable, efficient, and scalable [4]. This study aims to compare the performance of these three models in image generation using a selected set of criteria.

B. Objectives

The main objective of this study is to compare the performance of three generative models: Autoencoders, Generative Adversarial Networks (GANs), and Restricted Boltzmann Machines (RBMs), on a healthy mango leaf image dataset. The study aims to implement basic versions of these models, train them on a selected dataset, and assess the performance of each model through several hyperparameter configurations and evaluating the quality of the generated images.

II. Methodology

A. Data Handling

1) Data Description: The dataset utilized in this study was the **"Plant Leaves for Image Classification"** dataset, sourced from Kaggle. This dataset consists of 4,502 images in JPG file format, representing both healthy and unhealthy plant leaves from 12 different plant species. For the purposes of this study, only the first 100 images from the Mango Healthy class of plant leaves were used.

Fig. 1. Mango Healthy leaves

2) Data Preprocessing: Augmentation was performed on the selected images to increase the dataset's size. Specifically, each of the 100 initial images was subjected to the following transformations: (a) application of Gaussian blur, (b) scaling by various factors, (c) rotation at different angles, (d) flipping in multiple directions, (e) brightness level adjustment, and (f) addition of random Gaussian noise. These augmentation techniques generated multiple variations for each image, resulting in a total of 1,620 images after augmentation.

Fig. 2. Mango Healthy leaves after preprocessing

Afterwards, the images were then imported into Google Colab as the training dataset. During this process, all images were

resized to a fixed dimension of 30x40 pixels. After resizing, the dataset was converted into a NumPy array. Following this conversion, the pixel values were normalized to the range of 0 to 1 by dividing each value by 255.0. Finally, the images were flattened into 1D arrays to meet the shape requirements for training the models.
For the RBM model, an additional layer of preprocessing by binarizing the images was done, this is because RBMs are traditionally used for binary data. This is done by capturing only a range of green hues for the binarization.

B. Model Implementation

We implemented a variety of models to compare the capability of each model to reconstruct existing images and generate new ones derived from the dataset.

1) Autoencoders
2) Generative Adversarial Networks (GANs)
3) Restricted Boltzmann Machines (RBMs)

C. Hyperparameter Optimization

The number of epochs for model training was adjusted for each run to find the optimal training configuration for better performance. The models were trained with different epoch values including but not limited to 10, 50, 100, 128 and 256, while maintaining a constant batch size at 64. This variation helped assess how the duration of training affected the quality and accuracy of the generated outputs, and to determine the most effective epoch count for the dataset. Another setup was also done with all models, keeping the amount of epochs constant at 128, and varying the batch sizes with values 64, 96, 128, and 256. These variations helped provide insight as to whether batch size has a significant effect in the training time and quality of the outputs.

D. Model Evaluation

Mean Squared Error (MSE) is a widely used metric in image processing and machine learning to quantify the difference between predicted and actual values. It calculates the average squared difference between the original image and its reconstruction, providing a measure of reconstruction error. A lower MSE indicates a closer match, reflecting high fidelity and minimal loss of information, while a higher MSE highlights greater discrepancies and potential quality degradation. The squaring of errors ensures that larger differences are penalized more heavily, making MSE particularly sensitive to significant outliers in the reconstruction. However, it is worth noting that MSE measures pixel-wise differences and may not always align with human perceptual quality, as it does not account for spatial or structural similarities in the image [6]. Despite these limitations, MSE remains a robust and interpretable metric for evaluating model performance in tasks involving image reconstruction and regression [7].

III. Results and Discussion

A. Autoencoder

Autoencoders consist of three main components: an encoder, a code or bottleneck, and a decoder. The implementation of the Autoencoder model used in this study includes four encoding layers, each utilizing the Rectified Linear Unit (ReLU) activation function to reduce the dimensionality of the input data. The compressed representation from the encoder is passed through a code or bottleneck layer and then reconstructed by four corresponding decoding layers. The first three decoding layers use ReLU activation functions, while the final layer uses a sigmoid activation function to produce the reconstructed output.

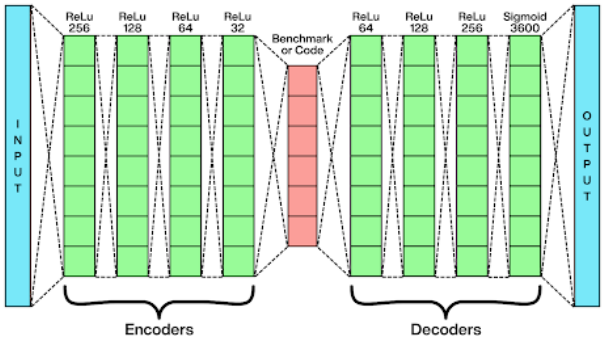

Fig. 3. Architecture of the Autoencoder model implemented in this study.

Additionally, a timer was integrated during the training and predicting stages of the model. This feature was used to record and display the total time taken to complete each respective step, providing insights into the computational efficiency of the implementation.

The hyperparameter optimization for the Autoencoder model involved modifying two specific hyperparameters: the number of epochs and the batch size. In the first setup, the batch size was held constant at 64, while the number of epochs was varied with values of 10, 50, 100, 128, 256, and 512. In the second setup, the number of epochs was kept constant at 128, while the batch size values varied from 64, 96, 128, 256, and 512. These variations allowed for the evaluation of the model's performance under different training durations to observe the effects of changing this specific hyperparameter on the model's learning and reconstruction capabilities.

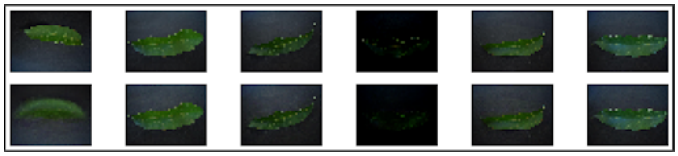

Fig. 4. Generated images from 10 epochs and constant batch size of 64

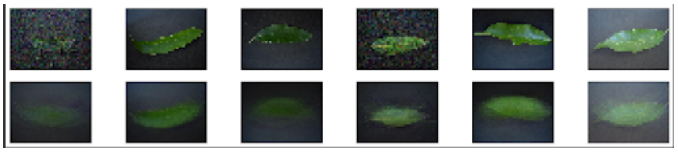

Fig. 5. Generated images from 50 epochs and constant batch size of 64

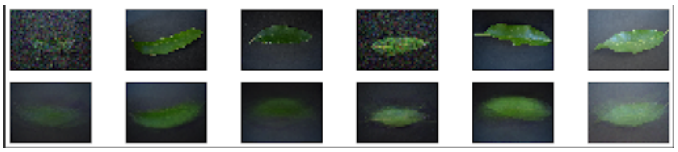

Fig. 6. Generated images from 100 epochs and constant batch size of 64

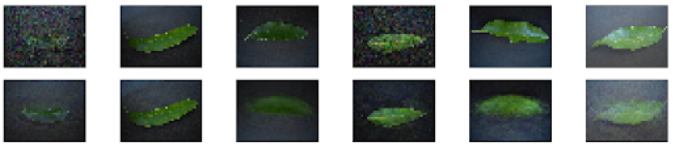

Fig. 7. Generated images from 128 epochs and constant batch size of 64

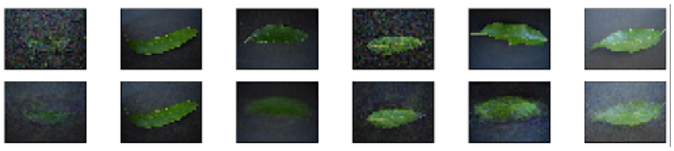

Fig. 8. Generated images from 256 epochs and constant batch size of 64

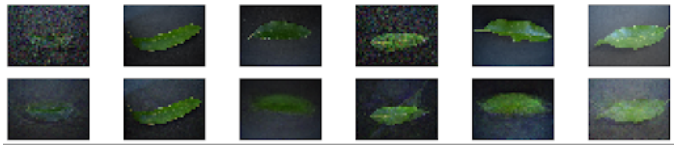

Fig. 9. Generated images from 512 epochs and constant batch size of 64

Epochs	Training time (seconds)	Prediction time (seconds)
10	35.4148s	0.2674s
50	176.0127s	0.3239s
100	258.0127s	0.3239s
128	358.4573s	0.2682s
256	702.3729s	0.2682s
512	1399.414s	0.6002s

TABLE I
TRAINING TIME AND PREDICTION TIME FOR EACH EPOCH WITH CONSTANT BATCH SIZE OF 64

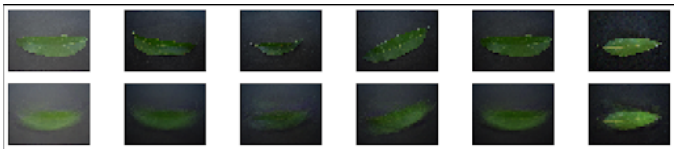

Fig. 11. Generated images from 96 batch size and constant 128 epochs

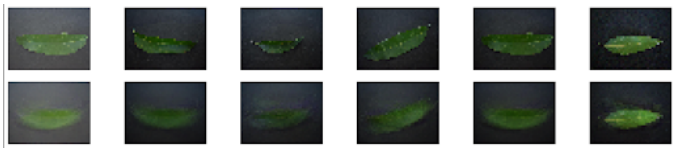

Fig. 10. Generated images from 64 batch size and constant 128 epochs

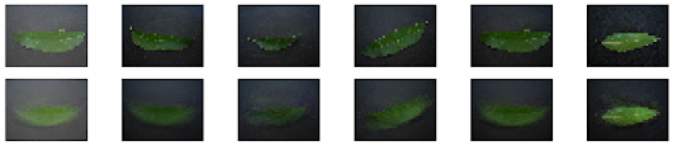

Fig. 12. Generated images from 128 batch size and constant 128 epochs

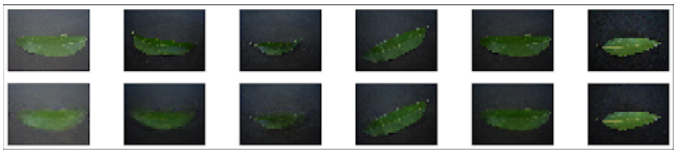

Fig. 14. Generated images from 512 batch size and constant 128 epochs

Batch Size	Training time (seconds)	Prediction time (seconds)
64	142.3701s	0.3846s
96	107.5204s	0.3846s
128	89.6792s	0.3846s
256	81.1284s	0.3846s
512	82.5746s	0.5103s

TABLE II
TRAINING TIME AND PREDICTION TIME FOR EACH BATCH SIZE WITH CONSTANT EPOCH OF 128

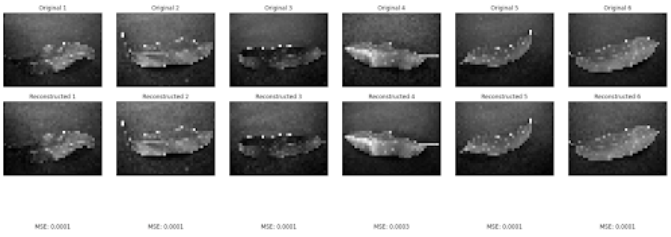

Fig. 15. Mean Squared Error

The results demonstrate that the model performs well in reconstructing images, with MSE values ranging from 0.0001 to 0.0003 and an average MSE of 0.0001 across all six images. The low MSE values indicate minimal differences between the original and reconstructed images, reflecting high fidelity in most cases. As shown in Figure 10, Images 1, 2, 3, 5, and 6 stand out with an exceptionally low MSE value of 0.0001, suggesting nearly perfect reconstruction, while Image 4 exhibits slightly higher MSE value of 0.0003, indicating minor imperfections. Despite these variations, the overall performance is consistent, and the differences are likely imperceptible to the human eye. These findings suggest that

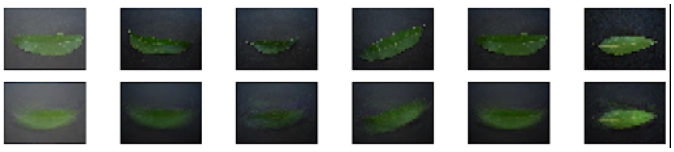

Fig. 13. Generated images from 256 batch size and constant 128 epochs

the model is reliable and effective for image reconstruction, though there is potential for fine-tuning to achieve even more uniform results across diverse images.

B. Generative Adversarial Networks (GANs)

This implementation of a Generative Adversarial Network (GAN) consists of a generator and a discriminator which generate RGB images. The generator runs it through multiple levels of LeakyReLU and BatchNormalization before producing an image using a tanh activation. The discriminator, on the other hand, ends with a sigmoid. TensorBoard logging is utilized to track losses and accuracies.

The hyperparameters modified for this model were: the number of epochs and batch size. The first setup involves keeping the batch size constant to 64 and changing the epochs with values 10, 50, 100, 128, and 256, as the capacity of the free tier of Colab was not able to handle running this model with 512 epochs. The second setup kept the number of epochs constant to 128, while changing the batch size with values 64, 96, 128, and 256. This model was not able to train with a batch size of 512, again due to the limitations of Google Colab's free plan.

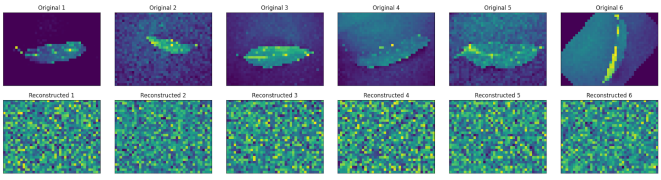

Fig. 16. Generated images from 10 epochs and constant batch size of 64

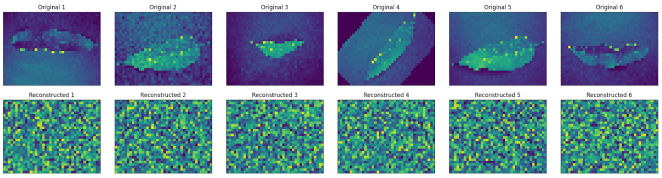

Fig. 17. Generated images from 50 epochs and constant batch size of 64

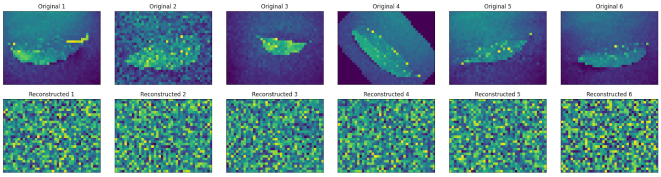

Fig. 19. Generated images from 128 epochs and constant batch size of 64

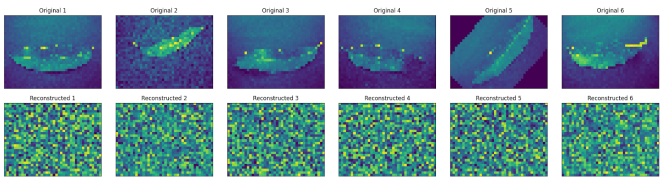

Fig. 18. Generated images from 100 epochs and constant batch size of 64

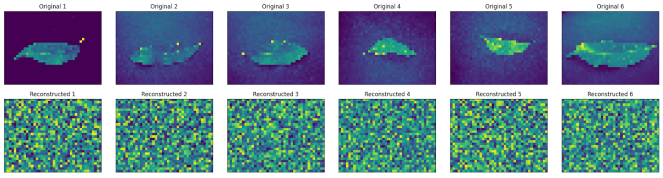

Fig. 20. Generated images from 256 epochs and constant batch size of 64

Epochs	Training time (seconds)
10	9.82s
50	28.58s
100	56.13s
128	75.76s
256	273.92s

TABLE III
TRAINING TIME FOR EACH EPOCH WITH CONSTANT BATCH SIZE OF 64

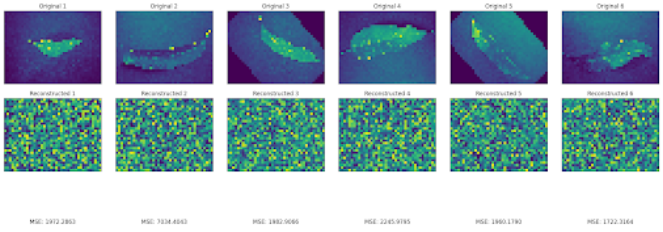

Fig. 21. Generated images from 64 batch size and constant 128 epochs

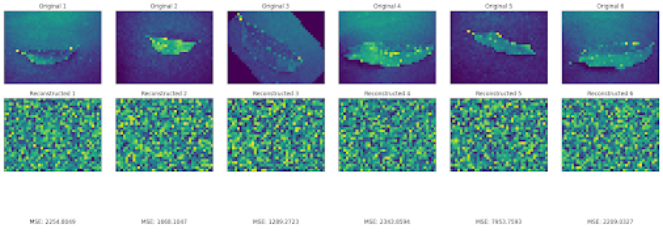

Fig. 22. Generated images from 96 batch size and constant 128 epochs

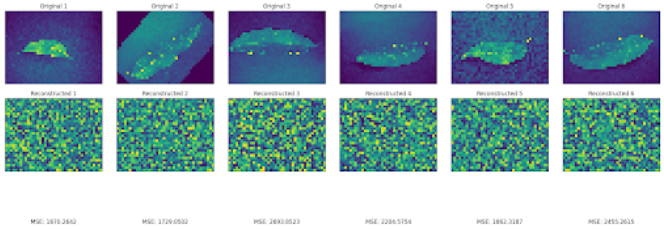

Fig. 23. Generated images from 128 batch size and constant 128 epochs

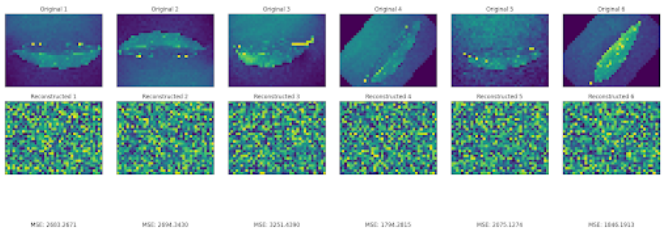

Fig. 24. Generated images from 256 batch size and constant 128 epochs

Batch Size	Training time (seconds)
64	69.40s
96	72.83s
128	80.66s
256	100.99s

TABLE IV
TRAINING TIME FOR EACH BATCH SIZE WITH CONSTANT EPOCH OF 128

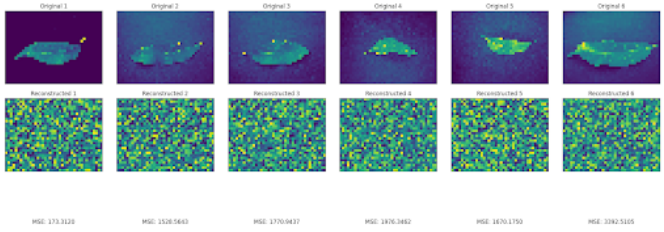

Fig. 25. Mean Squared Error

It can be seen from the results that the model struggled with generating images. MSE scores range from 173.3120 to 3392.5105, all indicating obvious differences between the original and reconstructed images. The overall performance of the model was poor, with issues also arising with training the model and generating images. TensorFlow is trying to load very large data components into memory, which exceeds the RAM budget. This issue occurs because the dataset contains high-resolution photos (3000x4000), which require large RAM to handle. The runtime also gets disconnected when the available runtime is exceeded due to higher epoch training.

C. Restricted Boltzmann Machines (RBMs)

The implementation of the Boltzmann Machine model utilizes the use of both binary visible and hidden layers. Using Xavier initialization, it is set up with random weights only within a certain range to maintain variance between them and to keep the model more stable during training. These weights are then modified during training through the use of contrastive divergence. The model alternates between a positive phase during training, in which it uses the visible data to calculate the hidden probabilities, and a negative phase, in which it uses the hidden layer to reconstruct the visible data and modify the weights accordingly. The sigmoid function is used as an activation function between phases. The input for the sigmoid function is clipped to be within a certain range to avoid calculation overflows.

The goal is to make the probability distribution of the model match that of the training data. To avoid overfitting, the model incorporates weight decay. The model generates samples using Gibbs sampling, iterating between updating the visible and hidden layers while beginning with a randomly selected visible layer. The model uses Gibbs sampling on random initial visible states to produce new samples after training.

To observe if the model was training properly, the reconstruction error per epoch is calculated. If it is found that the reconstruction error steadily decreases per epoch, it is likely that the model is training properly.

For comparison between the other models, the RBM were also set to train for 10, 50, 100, 128, 256, and 512 epochs with a constant batch size of 64. After that, the batch sizes were set to 64, 96, 128, 256, and 512 while keeping the epoch at 128. The learning rate is set to 0.1 which is found to be the most stable during training, but higher learning rates provided more varied outputs albeit more unstable during training.

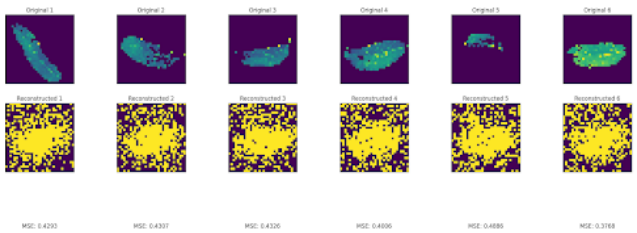

Fig. 26. Generated images from 10 epochs and constant batch size of 64

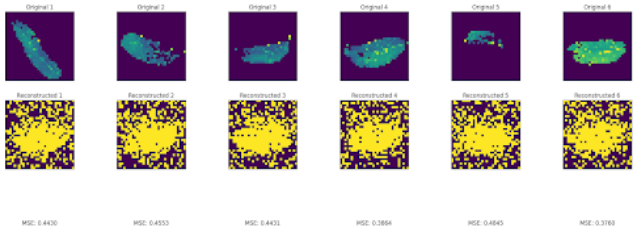

Fig. 27. Generated images from 50 epochs and constant batch size of 64

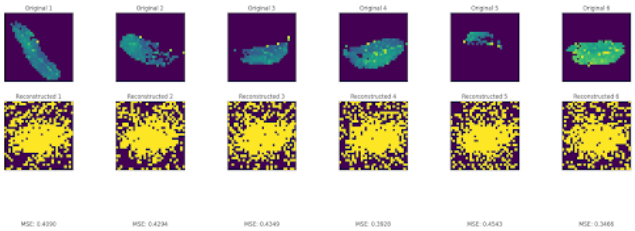

Fig. 28. Generated images from 100 epochs and constant batch size of 64

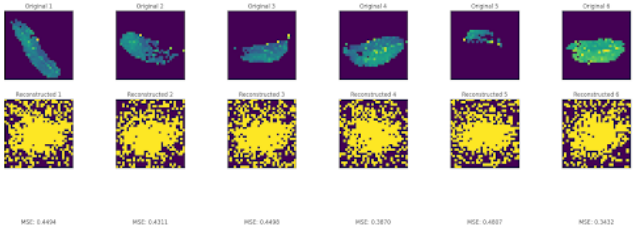

Fig. 29. Generated images from 128 epochs and constant batch size of 64

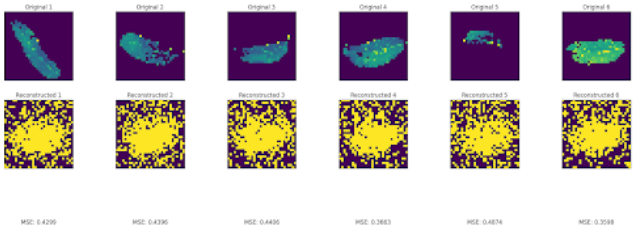

Fig. 30. Generated images from 256 epochs and constant batch size of 64

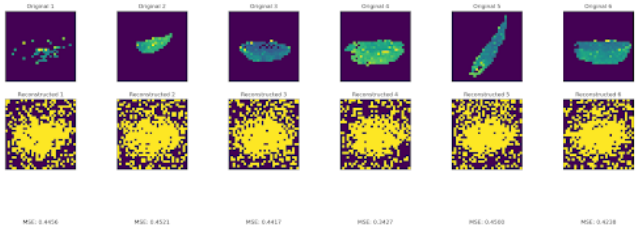

Fig. 31. Generated images from 512 epochs and constant batch size of 64

Epochs	Training time (seconds)
10	1674.66s
50	1777.94s
100	1807.50s
128	2169.40s
256	4700.78s
512	25168.28s

TABLE V
TRAINING TIME FOR EACH EPOCH WITH CONSTANT BATCH SIZE OF 64

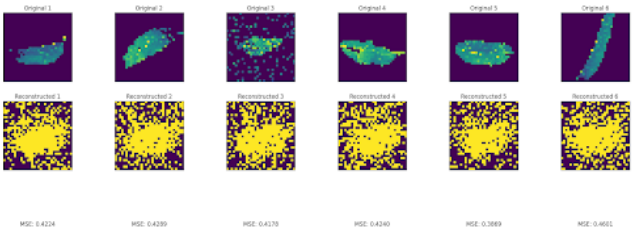

Fig. 32. Generated images from 64 batch size and constant 128 epochs

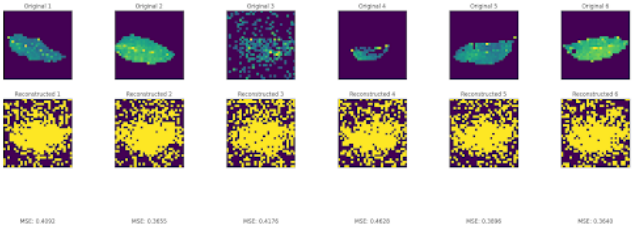

Fig. 33. Generated images from 96 batch size and constant 128 epochs

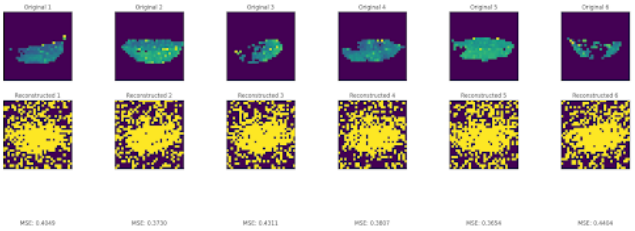

Fig. 34. Generated images from 128 batch size and constant 128 epochs

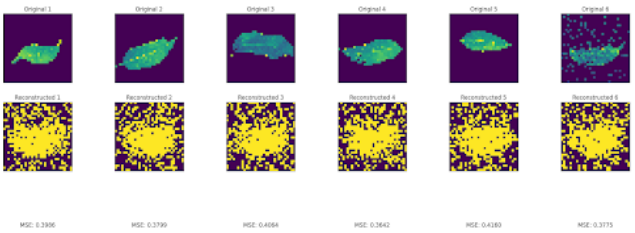

Fig. 35. Generated images from 256 batch size and constant 128 epochs

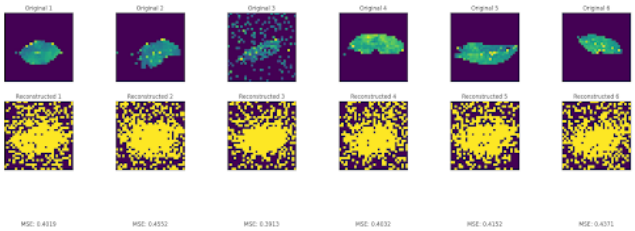

Fig. 36. Generated images from 512 batch size and constant 128 epochs

Batch Size	Training time (seconds)
64	3654.23s
96	3739.72s
128	4100.56s
256	3860.11s
512	3786.83s

TABLE VI
TRAINING TIME FOR EACH BATCH SIZE WITH CONSTANT EPOCH OF 128

The results show that the model has difficulty reconstructing images, with MSE values averaging at around 0.4. These high values mean there is a big difference between the original and reconstructed images, showing that the model is not very accurate. The lowest MSE from a sample is around 0.3642 which somewhat forms a general shape of the leaf, as seen in Figure 35. This suggests that the model is not fully learning the patterns in the dataset. There is a trend where maybe increasing the batch size slightly decreases the MSE. Increasing the number of epochs to be more than 512 may also be able to generate better outputs. Another cause may be even if a learning rate of 0.1 was found to be most stable, it may be too stable that it is not generating varying outputs because it is learning a little faster and missing out details. In other words, other adjustments to training, parameters, or the model itself might be needed to improve its ability to create better reconstructions.

D. Insights

The visuals created by Autoencoders, Restricted Boltzmann Machines (RBMs), and Generative Adversarial Networks (GANs) vary in accuracy. Autoencoders produce the clearest and most accurate outputs, the subject is still very distinguishable and apparent. RBMs are less accurate because data was binarized but still the shape of the subject was recognizable. GANs generated the least accurate visuals among the three where the subject is unrecognizable.

When comparing the time taken by the three models, Autoencoders performed significantly faster than both GANs and RBMs. For example, training Autoencoders with 256 epochs required 702.37 seconds, while GANs took 273.92 seconds for the same number of epochs, but this was only achievable with lower-resolution data due to memory limitations. RBMs were the slowest, taking over 3107 seconds for 100 epochs, emphasizing the computational intensity of their training process. Autoencoders' efficiency makes them suitable for tasks requiring rapid model deployment, whereas the longer runtimes of RBMs and GANs may be justified for more complex datasets or applications.

In terms of Mean Squared Error (MSE), Autoencoders outperformed GANs and RBMs by a wide margin. The MSE values for Autoencoders ranged from 0.0001 to 0.0043, indicating highly accurate image reconstructions. Conversely, RBMs showed average MSEs of 0.2060, with a notable variability between reconstructed images. GANs, on the other hand, struggled significantly, with MSE values between 173.31 and 3392.51, demonstrating poor reconstruction fidelity. These results highlight Autoencoders' effectiveness for image reconstruction tasks, while RBMs and GANs may require extensive fine-tuning or adjustments to achieve comparable accuracy.

IV. CONCLUSION AND RECOMMENDATIONS

In summary, the study was able to meet its objective of comparing the performance of three generative models—Autoencoders Generative Adversarial Networks (GANs) and Restricted Boltzmann Machines (RBMs)—by successfully implementing their basic versions, training them on healthy mango leaf images, and evaluating their outputs using Mean Squared Error (MSE), visual description, and runtime, although with some limitations due to resource restrictions in using Google Colab's free tier.

Longer training times are necessary for generative models to produce higher-quality and more accurate images. The quality and realistic elements of the generated images are improved by a longer training time, which enables the model to acquire larger amounts of patterns and features from the data. In addition, it enables the model to steadily improve performance and reduce errors.

Since this analysis only implements the basic versions of GAN and RBM, it is recommended to explore more advanced variants, such as Wasserstein GANs (WGANs) or Deep Boltzmann Machines (DBMs), to potentially enhance the accuracy and realism of generated images while addressing the observed limitations.

REFERENCES

[1] I. Bharti, K. Chauhan, and P. Aggarwal, "Generative ai," *Advances in linguistics and communication studies*, pp. 1–36, 09 2024. [Online]. Available: https://www.igi-global.com/chapter/generative-ai/357232

[2] G. Rani, J. Singh, and A. Khanna, "Comparative analysis of generative ai models," in *2023 International Conference on Advances in Computation, Communication and Information Technology (ICAICCIT)*, 2023, pp. 760–765.

[3] K. Berahmand, F. Daneshfar, E. S. Salehi, Y. Li, and Y. Xu, "Autoencoders and their applications in machine learning: a survey," *Artificial Intelligence Review*, vol. 57, 02 2024.

[4] R. Wang, "Review of generative models," *Applied and Computational Engineering*, vol. 8, pp. 524–529, 07 2023.

[5] V. Upadhya and P. S. Sastry, "An overview of restricted boltzmann machines," *A Multidisciplinary Reviews Journal*, vol. 99, pp. 225–236, 02 2019.

[6] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[7] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4.