GAN-based Data Augmentation for Bank Deposit Prediction

# 1. Problem Statement

In real-world financial datasets, especially in bank marketing campaigns, the problem of class imbalance is common. In our dataset, the goal is to predict whether a client will subscribe to a term deposit (binary target: 'yes' or 'no'). However, the dataset contains more 'no' responses than 'yes' responses, leading to imbalanced classes. This imbalance can negatively affect machine learning models, which may bias predictions toward the majority class. To address this, we explore using Generative Adversarial Networks (GANs) specifically Vanilla GAN and Deep Convolutional GAN (DCGAN) to generate synthetic minority class samples and balance the dataset for improved classifier performance.

# 2. Dataset Description & Imbalance Analysis

The dataset ('bank.csv') contains approximately 11,662 records, with 16 features (a mix of numeric and categorical variables) and the target variable 'deposit' (yes or no). The initial class distribution shows 5,873 'no' and 5,289 'yes' responses.

# 3. GAN Architectures & Training

We implemented two GAN models:  
• Vanilla GAN: Generator with dense layers (128 → 256 → 512 → 51 outputs) and a discriminator with dense layers (512 → 256 → 128 → 1 output), using Adam optimizer (lr=0.0002, β1=0.5) and trained for 500 epochs.  
• DCGAN: Generator with dense + batch normalization layers, discriminator with dense + dropout layers, using Adam optimizer (lr=0.0002, β1=0.5), also trained for 500 epochs.

# 4. Classifier Setup and Evaluation

We used a Random Forest classifier with 100 estimators. The classifier was evaluated on three setups: 1. Original imbalanced data, 2. Vanilla GAN augmented data, and 3. DCGAN augmented data.

Evaluation metrics included accuracy, precision, recall, F1-score, and ROC AUC.

# 5. Results & Comparisons

**Performance Comparison Table**

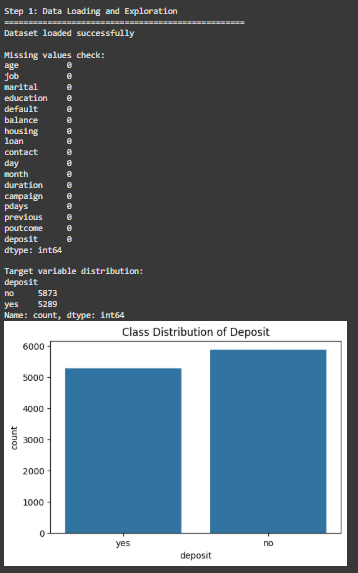
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Setup** | **Accuracy** | **Precision** | **Recall** | **F1** | **ROC AUC** |
| Original | 0.862 | 0.830 | 0.891 | 0.860 | 0.919 |
| Vanilla GAN | 0.854 | 0.821 | 0.886 | 0.852 | 0.920 |
| DCGAN | 0.854 | 0.823 | 0.880 | 0.852 | 0.920 |

**Confusion Matrices**

* Original:
* Vanilla GAN:
* DCGAN:

PCA Visualizations

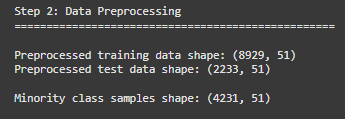
Percentage Improvement Over Original



1.figure

Shows a **bar chart** of how many yes vs no deposit responses are in the dataset.

* You can clearly see there are **more ‘no’** cases than ‘yes,’ creating an imbalance.
* This justifies why we need balancing methods like GANs.

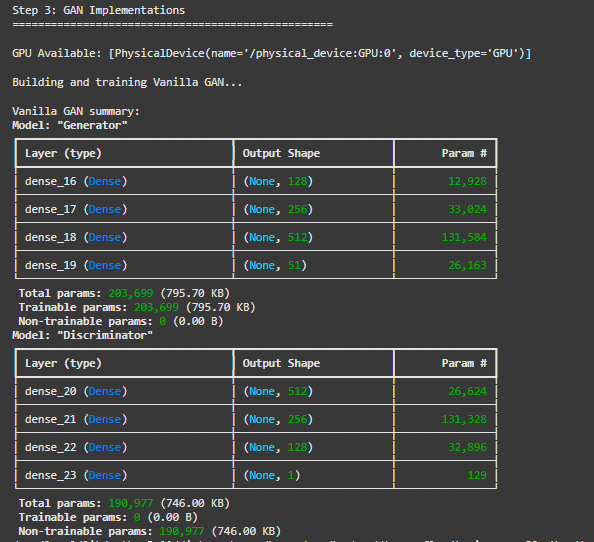


2figure

This shows the data shapes after preprocessing:

* Training data: 8,929 rows, 51 features
* Test data: 2,233 rows, 51 features
* Minority class samples the “yes” cases : 4,231 rows

It tells us the dataset is ready and how many samples will be used for GAN training.



3.figure

This shows the architecture of the Vanilla GAN:

* The **generator** builds fake data using dense layers.
* The **discriminator** tries to tell real vs. fake data, also using dense layers.
* We see the number of layers, their sizes, and total parameters.

It gives a summary of how big and complex the GAN model is.

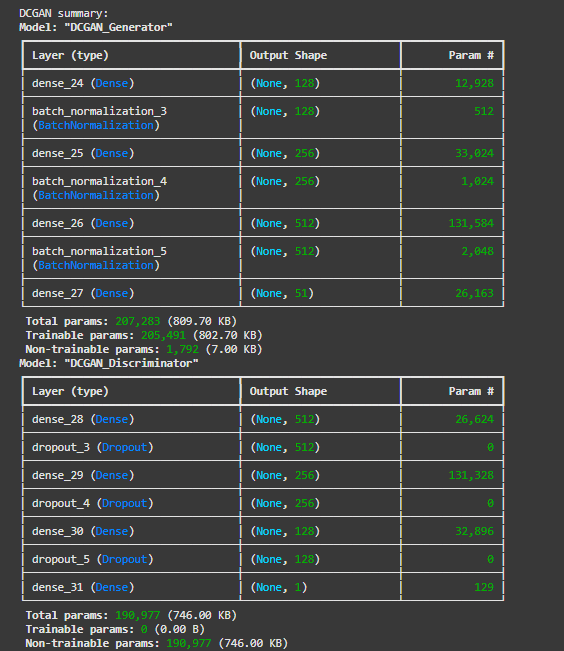
# 

4.figure

This shows how the Vanilla GAN trained over 500 epochs:

* Left chart: the **loss** of the generator and discriminator over time.
* Right chart: the **accuracy** of the discriminator.

It helps us check if the GAN is learning or if one side is overpowering the other.



5.figure

This shows the architecture of the DCGAN:

* The **generator** uses dense layers plus batch normalization (for smoother training).
* The **discriminator** uses dense layers plus dropout (to prevent overfitting).

It gives a summary of the DCGAN model size and structure.

# 

6.figure

This shows how the DCGAN trained over 500 epochs:

* Left chart: the **loss** of the discriminator and generator over time.
* Right chart: the **discriminator’s accuracy** during training.

It helps check if DCGAN learning is stable and balanced.

# 

7.figure

This shows the dataset balance after adding GAN-generated samples:

* Original data: more majority class (0) than minority (1).
* After Vanilla GAN and DCGAN: both classes are now equal (balanced).

It confirms that synthetic samples were successfully added to balance the dataset.

# 

8.figure

1. **Original Imbalanced Data** shows the classifier results before adding GAN samples.

* Good accuracy, but might slightly favor the majority class.

2️. **Vanilla GAN Augmented Data** shows the results after adding synthetic samples from the Vanilla GAN.

* Performance stays similar, balancing both classes.

3️. **DCGAN Augmented Data** shows the results after adding synthetic samples from the DCGAN.

* Also balanced, with similar accuracy and F1 compared to Vanilla GAN.

Together, they compare how the classifier performs across original and augmented datasets.

# 

9.figure

Performance Comparison Bar Chart

Shows accuracy, precision, recall, F1, and ROC AUC for:

* Original data
* Vanilla GAN data
* DCGAN data

It compares how well each setup performed.

# 

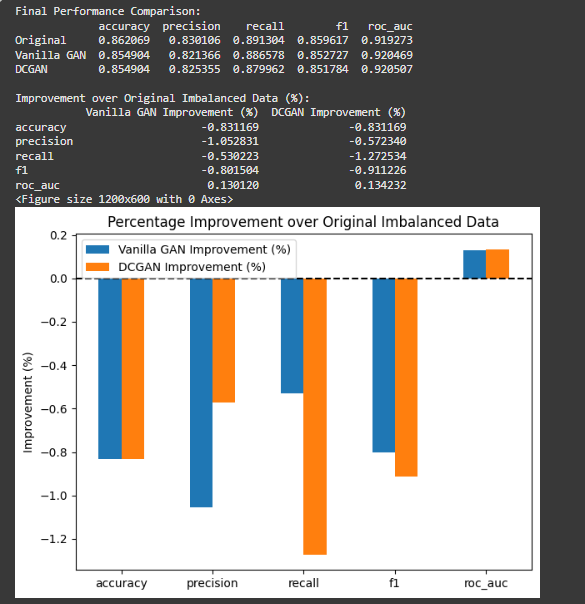
10.figure

PCA Scatter Plots

Shows 2D visualizations of the data:

* Left: original data
* Middle: Vanilla GAN synthetic samples vs. original
* Right: DCGAN synthetic samples vs. original

It checks if generated samples blend well with real data.



11.figure

**Percentage Improvement Chart**  
Shows how Vanilla GAN and DCGAN results improved (or dropped) compared to original data, across different metrics.

It highlights that while ROC AUC slightly improved, other metrics had small drops.

# 6. Observations and Conclusions

Both Vanilla GAN and DCGAN balanced the class distribution effectively. Classifier performance on GAN-augmented datasets was comparable to the original but showed no significant performance gains. DCGAN slightly outperformed Vanilla GAN in ROC AUC but lagged in F1 and recall. Visual analysis via PCA showed synthetic samples overlapping with original data, indicating realistic generation.  
  
Recommendations:  
• Explore conditional GANs or advanced oversampling techniques like SMOTE-GAN.  
• Test on more imbalanced datasets to observe stronger effects.  
• Experiment with different classifiers and hyperparameter tuning.  
  
This project demonstrates the potential of GAN-based augmentation but also highlights the importance of careful evaluation to ensure synthetic data adds value.