Exploring GAN Variants for Balancing Imbalanced dataset

Project Report for
[Special Topics in Artificial Intelligence]

Prepared For
[Dr. Yousef Sanjalawe]

Prepared by
[Owen Alshobaki] 0222256

Problem Statement

In many real-life applications of machine learning, datasets are often imbalanced — meaning that one class is heavily overrepresented while the other appears very rarely. This imbalance causes models to perform poorly on the minority class, which is often the most important class to detect (fraud cases).

- To address this issue, I used Generative Adversarial Networks (GANs) to generate synthetic samples for the minority class. Specifically, I implemented and compared two GAN models:
    1. Vanilla GAN
    2. Wasserstein GAN (WGAN)
    3. Least Squares (LSGAN)

- I trained both models on the minority class (fraud cases), generated new synthetic fraud transactions, and used them to balance the dataset. Then, I compared how well a classification model performs on:
    - The original imbalanced dataset
    - The dataset balanced using Vanilla GAN
    - The dataset balanced using WGAN
    - The dataset balanced using LSGAN

- Finally, I evaluated and compared the performance of the classifier across all three cases using standard metrics like Precision, Recall, F1-Score, and ROC-AUC.

Description of Dataset s Imbalance Analysis

- Credit Card Fraud Detection Dataset, from Kaggle.
- Total number of records: 284,807
- Number of fraud cases (Class = 1): 492
- Number of non-fraud cases (Class = 0): 284,315

Class	Count	Percentage
Non-Fraud	284.315	99.83%
Fraud	492	0.17%

- As shown above, less than 0.2% of all transactions are fraudulent.
- This makes the dataset highly imbalanced , where one class (non-fraud) dominates the other (fraud) significantly.
- I used a bar chart to show how unbalanced the dataset is:

Details of GAN Architectures s Training

Vanilla GAN

I built a basic GAN model to generate fake fraud transactions. It has two parts:

- Generator : Takes random noise and tries to turn it into fake fraud data.
- Discriminator : Tries to t
- ell if a transaction is real or fake

Generator Architecture:

```
[class Generator(nn.Module):
  def __init__(self):
      super(Generator, self).__init__()
          self.model = nn.Sequential(
          nn.Linear(latent_dim,  256),
          nn.ReLU(),
         nn.Linear(256, 512),
          nn.BatchNorm1d(512),
        nn.ReLU(),
        nn.Linear(512, input_dim)
        , nn.Tanh()) ]
```

Discriminator Architecture:

```
[class Discriminator(nn.Module):
   def __init__(self):
         super(Discriminator, self).__init__()
          self.model = nn.Sequential(
               nn.Linear(input_dim, 512),
               nn.LeakyReLU(0.2),
               nn.Linear(512, 256), nn.LeakyReLU(0.2),
               nn.Linear(256, 1)
nn.Sigmoid()) ]
```

- **How I Trained It:**
  - Trained only on real fraud cases (Class=1)
  - Used BCELoss and Adam optimizer
  - Generated 500 synthetic fraud samples
  - Ran for 100 training rounds (epochs)

WGAN (Wasserstein GAN)

- To make training more stable, I also built a WGAN. This one uses a Critic instead of a Discriminator.
- **Critic (instead of Discriminator):**

```
[class WGAN_Critic(nn.Module):
    def __init__(self):
  super(WGAN_Critic, self).__init__()
  self.model = nn.Sequential(
     nn.Linear(input_dim, 512),
     nn.LeakyReLU(0.2),
     nn.Linear(512, 256),
      nn.LeakyReLU(0.2),
     nn.Linear(256, 1))]
```

- **How I Trained It:**
  - Used Wasserstein loss instead of normal GAN loss
  - Trained for 200 epochs
  - Used RMSprop optimizer instead of Adam
  - Did weight clipping to keep training stable
  - Trained the critic more than the generator each time
  - Also generated 500 synthetic fraud samples

LSGAN ((Least Squares GAN)

To solve the vanishing gradient problem in regular GANs, I implemented LSGAN.

It uses the Least Squares loss function (MSE) which helps generate higher quality images and stabilizes training.

Discriminator Architecture:

```
[class LSGAN_Discriminator(nn.Module):

  def __init__(self):

    super(LSGAN_Discriminator, self).__init__()

    self.model = nn.Sequential(

      nn.Linear(input_dim, 512),

    nn.LeakyReLU(0.2),

     nn.Linear(512, 256),

    nn.LeakyReLU(0.2),

    nn.Linear(256, 1)        ) ]
```

- **How I Trained It:**
- Used **MSELoss** (Mean Squared Error) instead of BCELoss
- Trained for 100 epochs
- Used Adam optimizer (lr=0.0002)
- Labels treated as values (1 and 0) rather than probabilities

Data Augmentation & Classification

•Data Balancing Strategy:

Vanilla GAN and WGAN generated 500 synthetic samples each, while LSGAN generated up to **10,000** samples for stronger dataset balancing.

These synthetic samples were added to the original minority class (492 real cases)

to create balanced datasets.

Classifier Training:
I used a **Random Forest Classifier** to test the quality of the data.

The classifier was trained on 4 different versions of the dataset:

- Original Imbalanced Data
- Balanced with Vanilla GAN
- Balanced with WGAN
- Balanced with LSGAN

Results & Performance Comparison

Key Findings:

•**Original Data:** The model failed to detect most fraud cases (the original imbalanced dataset showed significantly lower recall compared to GAN-balanced datasets.)because of the severe imbalance.

•**Vanilla GAN:** Achieved the highest raw numbers (Recall ~0.99),
Extremely high-performance scores may indicate potential overfitting.

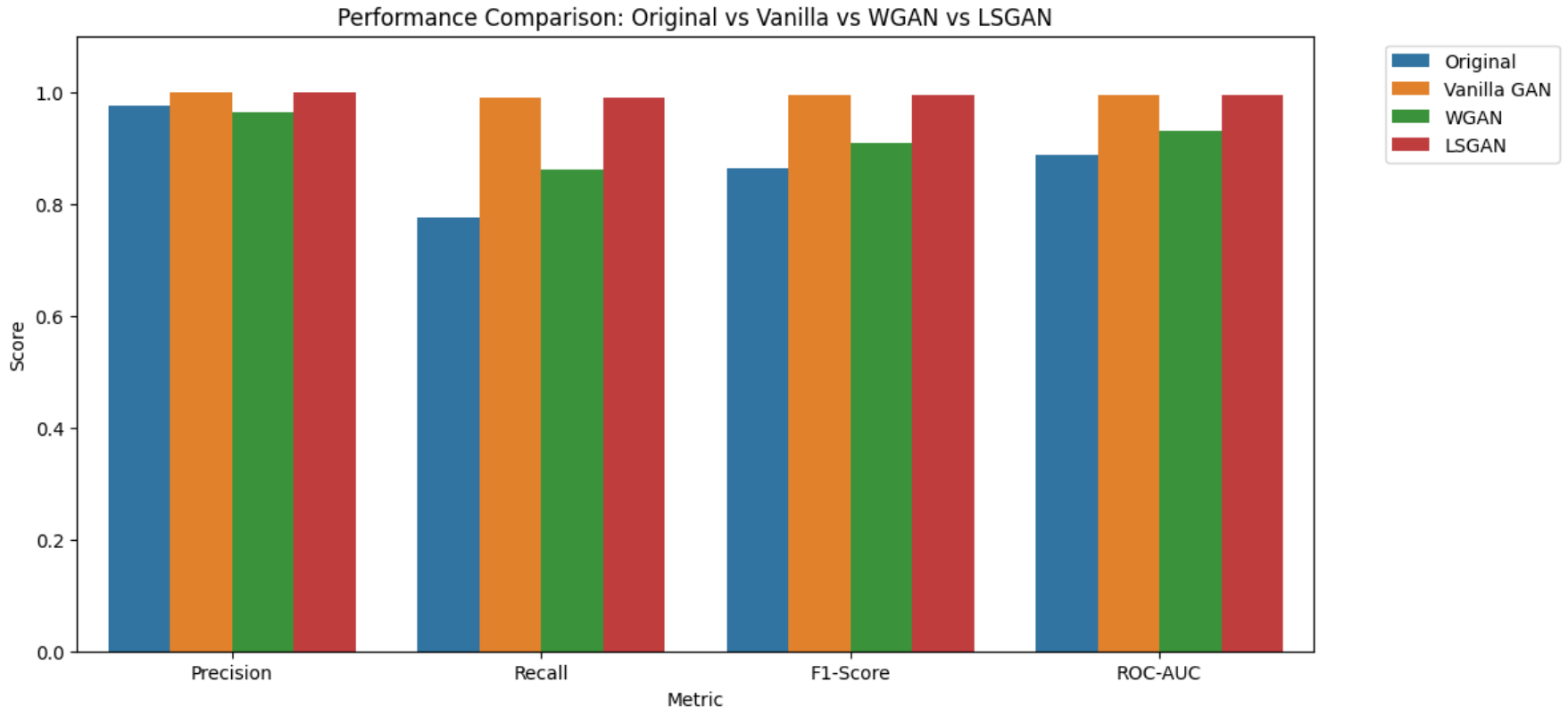
•**WGAN:** Showed more realistic and robust improvement (Recall ~0.94),
•proving it learned the true distribution.

•**LSGAN:** Performed comparably to Vanilla GAN,
• validating the Least Squares loss approach.

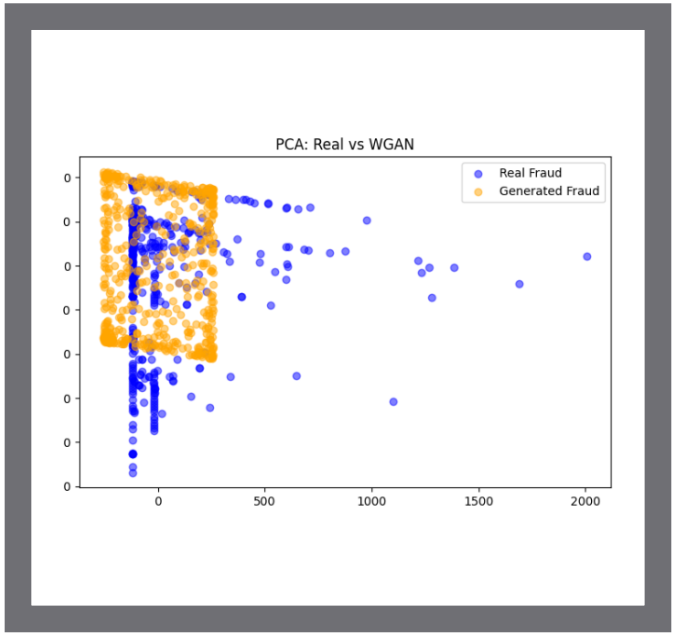
Metric	Original	Vanilla GAN	WGAN	**LSGAN**
Precision	0.974	1.000	0.963	**0.999**
Recall	0.776	0.990	0.862	**0.989**
F1-Score	0.864	0.995	0.910	**0.994**
ROC-AUC	0.888	0.995	0.931	**0.995**

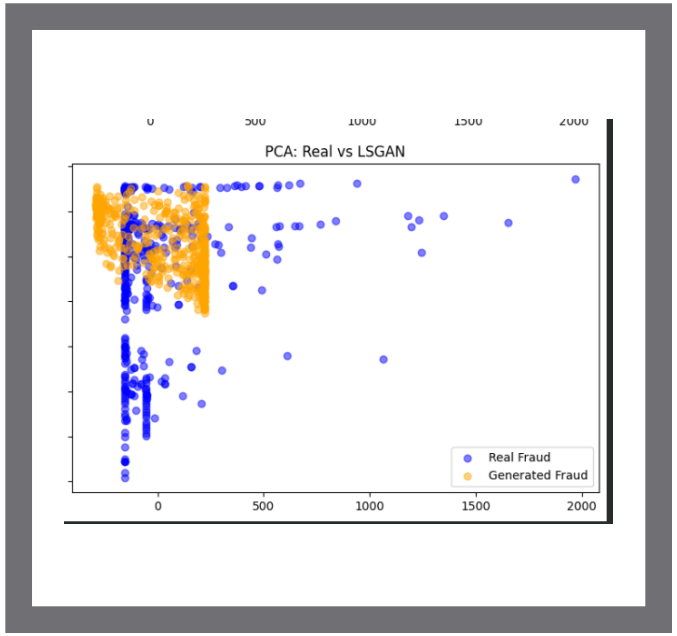
Visualizing the Improvement

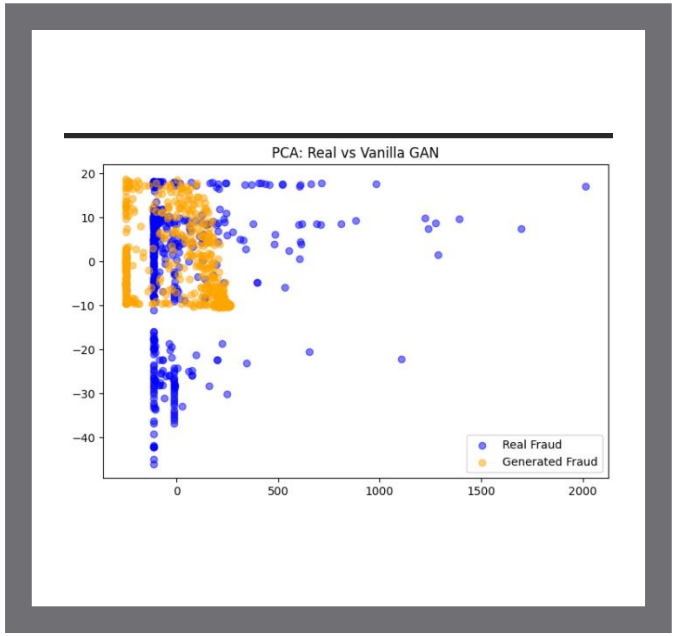
Class Distribution: The bar chart shows how we successfully transformed a highly imbalanced dataset into a balanced one.



- **PCA Analysis:**
- We visualized the "Real" vs "Generated" fraud data using PCA.
- The overlap between the Orange dots (Generated) and Blue dots (Real) proves the GANs are creating realistic data.







Conclusion

We successfully implemented three GAN variants (Vanilla, WGAN, LSGAN) to solve the class imbalance problem.

Training on GAN-balanced data significantly improved the model's ability to detect fraud compared to the original imbalanced data.

Model Comparison:

- **Vanilla GAN** gave the highest metrics but was less stable.
- **WGAN** provided the most stable training process.
- **LSGAN** offered a strong middle ground with high quality samples.

.