**6501 Progress Report Group 4**

# Improving Credit Card Fraud Detection using Generative Adversarial Networks

Group 4 Team Member: Hao Ning, Jun Ying

## Working Schedule

| Time | Milestone |
| --- | --- |
| 09/21/2020 | Exploratory Data Analysis (EDA): Jun<br>Base Model: Hao |
| 09/28/2020 | Original data + GAN |
| 10/05/2020 | Network & Framework Development<br>WGAN: Hao<br>BAGAN: Jun |
| 10/12/2020 | WGAN & BAGAN Evaluation & Analysis |
| 10/19/2020 | Preliminary Presentation |
| 10/26/2020 | Network & Framework Development<br>WGAN_GP: Hao<br>BEGAN: Jun |
| 11/02/2020 | WGAN_GP, BEGAN Evaluation & Analysis |
| 11/09/2020 & 11/16/2020 | Summary of Results, Github |
| 11/23/2020 | Manuscript |
| 11/30/2020 & 12/07/2020 | Mock Presentation & Presentation and Journal Submission |

**6501 Progress Report Group 4**

**09/21/2020**

EDA

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
```

About the dataset, there are 30 features and 1 class (normal:0, fraud:1)

```
   Time        V1        V2        V3  ...       V27       V28  Amount  Class
0   0.0 -1.359807 -0.072781  2.536347  ...  0.133558 -0.021053  149.62      0
1   0.0  1.191857  0.266151  0.166480  ... -0.008983  0.014724    2.69      0
2   1.0 -1.358354 -1.340163  1.773209  ... -0.055353 -0.059752  378.66      0
3   1.0 -0.966272 -0.185226  1.792993  ...  0.062723  0.061458  123.50      0
4   2.0 -1.158233  0.877737  1.548718  ...  0.219422  0.215153   69.99      0
```

There is no null value in the dataset.

```
 Total null values in the dataset
0
```

As we know, the dataset is extremely imbalanced(0.173%).

```
The amounts of normal transactions (class 0) & fraud transactions (class 1)
0    284315
1       492
```

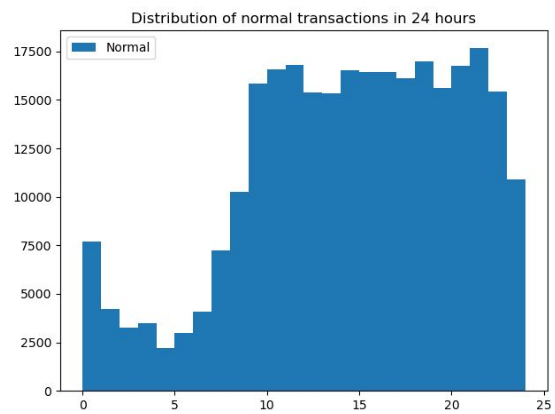
We have observed that there are some transactions which are 0.

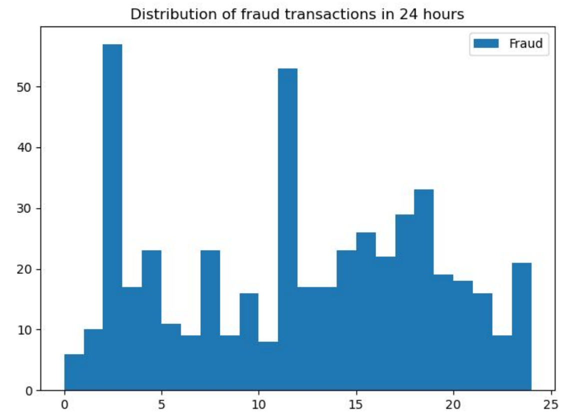
```
                Time            V1  ...         Amount          Class
count  284807.000000  2.848070e+05  ...  284807.000000  284807.000000
mean       14.537951  3.919560e-15  ...      88.349619       0.001727
std         5.847061  1.958696e+00  ...     250.120109       0.041527
min         0.000000 -5.640751e+01  ...       0.000000       0.000000
25%        10.598194 -9.203734e-01  ...       5.600000       0.000000
50%        15.010833  1.810880e-02  ...      22.000000       0.000000
75%        19.329722  1.315642e+00  ...      77.165000       0.000000
max        23.999444  2.454930e+00  ...   25691.160000       1.000000
```

The total number of 0 amount:  1825 (1.479% fraud)

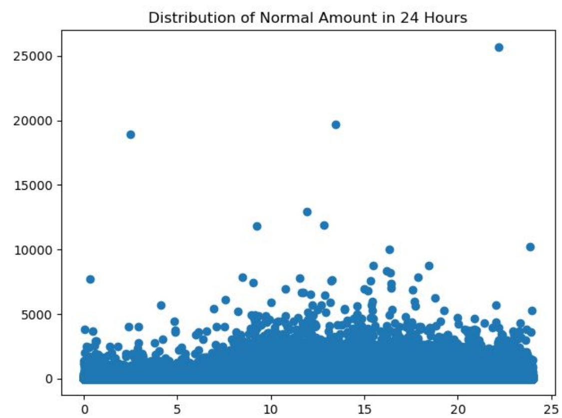
```
The null amounts of normal transactions (class 0) & fraud transactions (class 1)
0    1798
1      27
Name: Class, dtype: int64
```

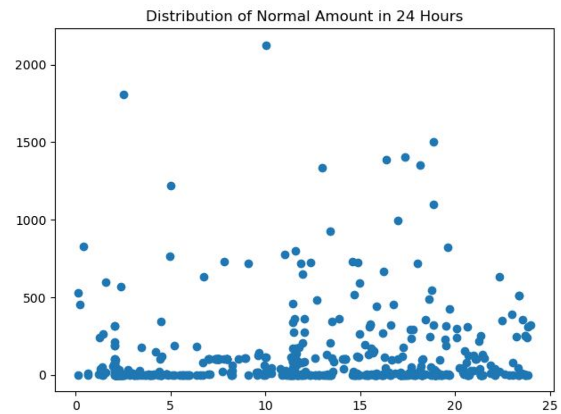
# 6501 Progress Report Group 4



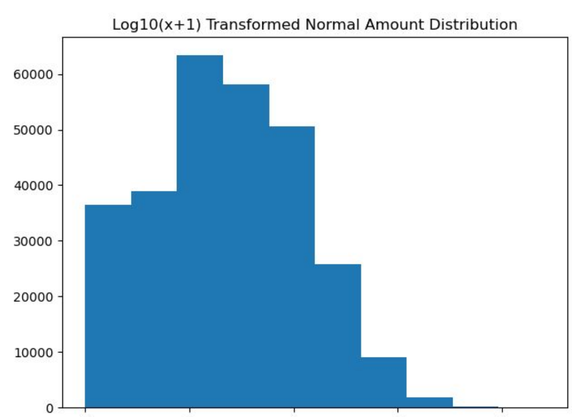



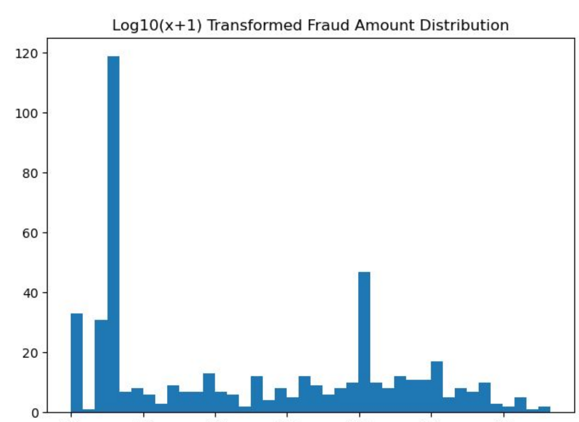

From the histogram, we can observe that normal transactions generally occur from 9 am to 0 am. However, the fraud transactions occur particularly frequently at 2 am and 12 pm.



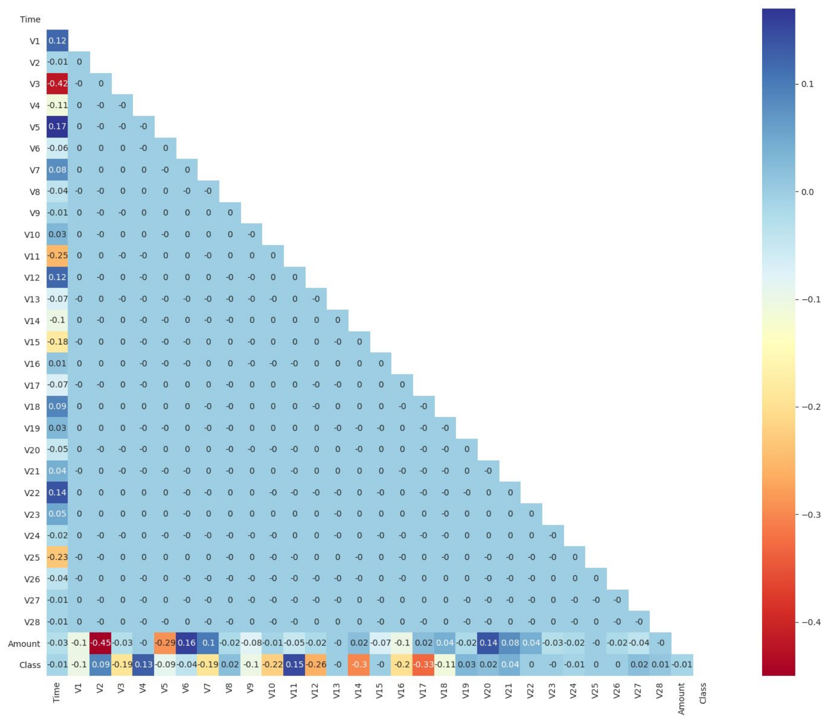

We can find from this scatter plot that the amount of super large transactions is very small. In comparison, the largest amount of nurmal transactions is over €25,000. However, the largest amount of fraud transactions is only €2,000.

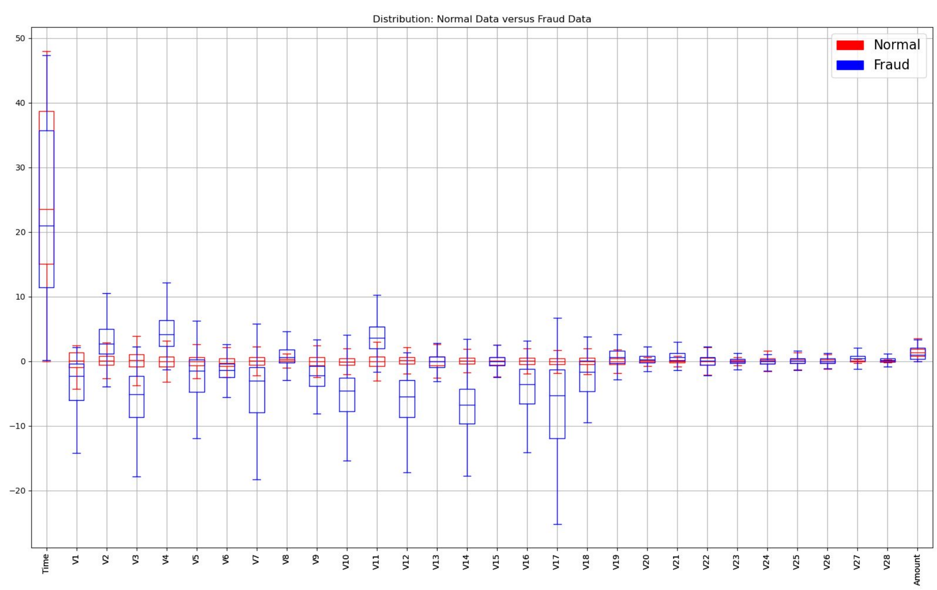



Normal amount was from ten to hundred. Fraud Amount distributed in less than €1.

# 6501 Progress Report Group 4



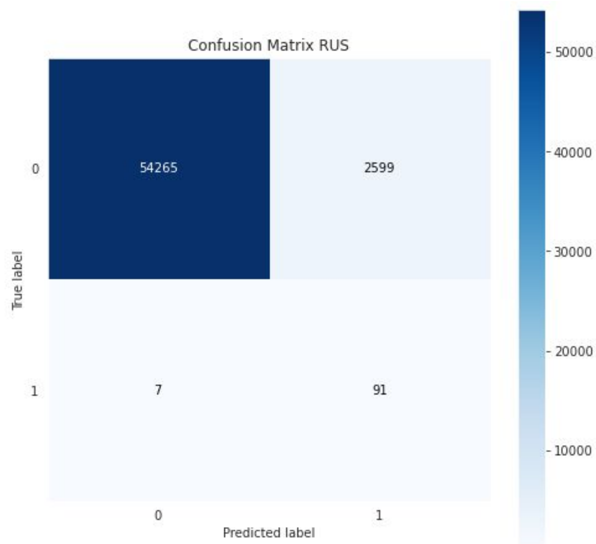


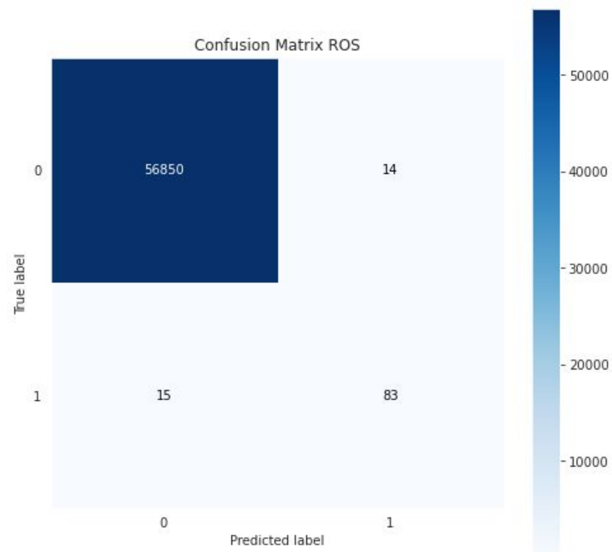

Base Model:

1. Train Test Split & Stratified: 80% 227845 (**394 fraud**), 20%  56962 (**98 fraud**)
2. Random Under Sampling (RUS) and Random Over Sampling (ROS)
3. GridsearchCV for XGBoostClassifier
4. Predict with best_params
5. Test performance evaluation

**6501 Progress Report Group 4**

| | |
|---|---|
| RUS<br>1 394<br>0 394 | ROS<br>1 227451<br>0 227451 |
| Accuracy: 0.9542502018889786<br>Precision: 0.03382899628252788<br>Recall: 0.9285714285714286<br>F1 score: 0.06527977044476328<br>ROC AUC score: 0.941432942760672 | Accuracy: 0.9994908886626171<br>Precision: 0.8556701030927835<br>Recall: 0.8469387755102041<br>F1 score: 0.8512820512820514<br>ROC AUC score: 0.923346287023532 |




**09/28/2020**

Implemented with Keras

A bit change from from original gan:
Vanilla gan deals with images, but we are dealing with tabular data, so tanh is removed.
Also, we removed batch normalization since the training results are bad.

# 6501 Progress Report Group 4

Noise = 32
len(features) = 30

Generator

| Input | Output size |
|---|---|
| (32, 1) | (64,1) |
| (64, 1) | (128, 1) |
| (128, 1) | (256, 1) |
| (256, 1) | (30, 1) |

LeakyReLU(0.2)

Discriminator

| Input | Output size |
|---|---|
| (30, 1) | (256,1) |
| (256, 1) | (128, 1) |
| (128, 1) | (64, 1) |
| (64, 1) | (1, 1) |

LeakyReLU(0.2)

Train
epoch=10, batch_size=128, steps_per_epoch=100

```
EPOCH #  10 -------------------------------------------------
Steps (5 / 50): [Loss_D_real: 0.101874, Loss_D_fake: 0.804704, acc: 55.47%] [Loss_G: 0.577402]
Steps (10 / 50): [Loss_D_real: 0.034651, Loss_D_fake: 0.794337, acc: 58.59%] [Loss_G: 0.583686]
Steps (15 / 50): [Loss_D_real: 0.043936, Loss_D_fake: 0.802042, acc: 64.84%] [Loss_G: 0.606348]
Steps (20 / 50): [Loss_D_real: 0.034166, Loss_D_fake: 0.831742, acc: 57.81%] [Loss_G: 0.573465]
Steps (25 / 50): [Loss_D_real: 0.045282, Loss_D_fake: 0.852394, acc: 56.25%] [Loss_G: 0.597851]
Steps (30 / 50): [Loss_D_real: 0.072814, Loss_D_fake: 0.801378, acc: 60.94%] [Loss_G: 0.593632]
Steps (35 / 50): [Loss_D_real: 0.062091, Loss_D_fake: 0.837749, acc: 55.47%] [Loss_G: 0.597265]
Steps (40 / 50): [Loss_D_real: 0.099103, Loss_D_fake: 0.853835, acc: 54.69%] [Loss_G: 0.578026]
Steps (45 / 50): [Loss_D_real: 0.088608, Loss_D_fake: 0.799784, acc: 60.16%] [Loss_G: 0.573147]
Steps (50 / 50): [Loss_D_real: 0.043118, Loss_D_fake: 0.808880, acc: 61.72%] [Loss_G: 0.589081]
```

epoch=20, batch_size=128, steps_per_epoch=100

**6501 Progress Report Group 4**

```
EPOCH #  20 ------------------------------------------------
Steps (10 / 100): [Loss_D_real: 0.041530, Loss_D_fake: 0.459832, acc: 100.00%] [Loss_G: 1.017299]
Steps (20 / 100): [Loss_D_real: 0.049112, Loss_D_fake: 0.453462, acc: 100.00%] [Loss_G: 1.008852]
Steps (30 / 100): [Loss_D_real: 0.065196, Loss_D_fake: 0.458509, acc: 100.00%] [Loss_G: 1.018385]
Steps (40 / 100): [Loss_D_real: 0.061704, Loss_D_fake: 0.459229, acc: 100.00%] [Loss_G: 1.009706]
Steps (50 / 100): [Loss_D_real: 0.051204, Loss_D_fake: 0.460033, acc: 100.00%] [Loss_G: 1.005355]
Steps (60 / 100): [Loss_D_real: 0.059828, Loss_D_fake: 0.457545, acc: 100.00%] [Loss_G: 1.008903]
Steps (70 / 100): [Loss_D_real: 0.056420, Loss_D_fake: 0.468705, acc: 100.00%] [Loss_G: 1.013539]
Steps (80 / 100): [Loss_D_real: 0.052308, Loss_D_fake: 0.452640, acc: 100.00%] [Loss_G: 1.017379]
Steps (90 / 100): [Loss_D_real: 0.053092, Loss_D_fake: 0.457065, acc: 100.00%] [Loss_G: 0.990622]
Steps (100 / 100): [Loss_D_real: 0.052301, Loss_D_fake: 0.467294, acc: 100.00%] [Loss_G: 1.017609]
```

Original data + GAN

Original x_train has total of 227451 transactions, 227451 normal & 394 Fraud

1000

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 |
|---|---|---|---|---|---|---|---|---|
| **count** | 1000.000000 | 1000.000000 | 1000.000000 | 1000.000000 | 1000.000000 | 1000.000000 | 1000.000000 | 1000.000000 |
| **mean** | 3.198658 | -0.762727 | 0.550367 | -0.873793 | 1.195856 | -0.272612 | -0.130996 | -0.884949 |
| **std** | 0.809518 | 0.242217 | 0.253704 | 0.331019 | 0.368733 | 0.255425 | 0.269881 | 0.360904 |
| **min** | 1.008744 | -1.553868 | -0.155640 | -2.179475 | 0.302933 | -1.204325 | -1.027105 | -2.258104 |
| **25%** | 2.607475 | -0.912981 | 0.371793 | -1.077172 | 0.923517 | -0.423956 | -0.310939 | -1.102170 |
| **50%** | 3.141616 | -0.749497 | 0.537789 | -0.843910 | 1.163639 | -0.256563 | -0.115827 | -0.863040 |
| **75%** | 3.701715 | -0.591140 | 0.721689 | -0.630434 | 1.437751 | -0.110544 | 0.049115 | -0.619941 |
| **max** | 6.568467 | -0.130614 | 1.558176 | -0.094866 | 2.794897 | 0.407353 | 0.780389 | -0.010803 |

227451

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 |
|---|---|---|---|---|---|---|---|---|
| **count** | 227057.000000 | 227057.000000 | 227057.000000 | 227057.000000 | 227057.000000 | 227057.000000 | 227057.000000 | 227057.000000 |
| **mean** | 3.218182 | -0.756782 | 0.545352 | -0.866097 | 1.202468 | -0.291984 | -0.135337 | -0.896145 |
| **std** | 0.811016 | 0.237066 | 0.248467 | 0.324762 | 0.375655 | 0.256962 | 0.262601 | 0.354981 |
| **min** | 0.864484 | -2.054668 | -0.467559 | -3.083328 | -0.161810 | -1.679709 | -1.472434 | -2.770730 |
| **25%** | 2.637128 | -0.907640 | 0.373894 | -1.067587 | 0.936952 | -0.454703 | -0.305321 | -1.119700 |
| **50%** | 3.155144 | -0.741693 | 0.535582 | -0.835388 | 1.177741 | -0.278191 | -0.129604 | -0.866557 |
| **75%** | 3.728471 | -0.590163 | 0.706314 | -0.633782 | 1.443069 | -0.115094 | 0.040692 | -0.641359 |
| **max** | 8.098510 | 0.093333 | 1.781384 | 0.096544 | 3.173466 | 0.693545 | 1.145589 | 0.370842 |

The fraud in x_train

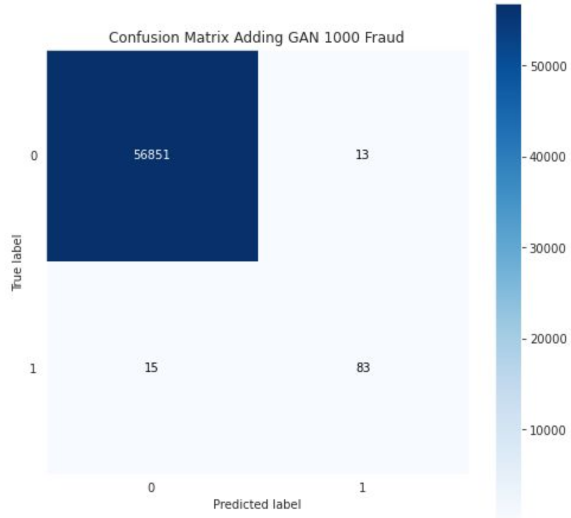
**6501 Progress Report Group 4**

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 |
|---|---|---|---|---|---|---|---|---|
| **count** | 394.000000 | 394.000000 | 394.000000 | 394.000000 | 394.000000 | 394.000000 | 394.000000 | 394.000000 |
| **mean** | 23.008938 | -4.707808 | 3.588729 | -7.068378 | 4.592975 | -3.101629 | -1.387192 | -5.539909 |
| **std** | 13.347935 | 6.841390 | 4.309436 | 7.166449 | 2.883467 | 5.406586 | 1.864770 | 7.316745 |
| **min** | 1.239444 | -30.552380 | -8.402154 | -31.103685 | -1.313275 | -22.105532 | -5.773192 | -43.557242 |
| **25%** | 11.500278 | -5.996596 | 1.229209 | -8.436924 | 2.419178 | -4.741036 | -2.504633 | -7.765017 |
| **50%** | 21.393056 | -2.272114 | 2.662472 | -5.133485 | 4.258196 | -1.522962 | -1.421577 | -2.926216 |
| **75%** | 35.912917 | -0.410418 | 4.737900 | -2.302626 | 6.390866 | 0.240184 | -0.361122 | -0.900824 |
| **max** | 47.318889 | 2.132386 | 22.057729 | 2.250210 | 12.114672 | 11.095089 | 6.474115 | 5.802537 |

The little std observed in the GAN generated data indicates **mode collapse** in vanilla gan

| Base ROS from Original data<br>Normal: 227451<br>Fraud: 227451 | Add GAN 1000 then ros.fit<br>Normal: 227451<br>Fraud: 227451 | GAN 227057<br>Normal: 227451<br>Fraud: 227451 |
|---|---|---|
| Accuracy:<br>0.9994908886626171<br>Precision:<br>0.8556701030927835<br>Recall:<br>0.8469387755102041<br>F1 score:<br>0.8512820512820514<br>ROC AUC score:<br>0.923346287023532 | Accuracy:<br>0.9995084442259752<br>Precision:<br>**0.8645833333333334**<br>Recall:<br>0.8469387755102041<br>F1 score:<br>0.8556701030927835<br>ROC AUC score:<br>0.9233550799329299 | Accuracy:<br>0.9995435553526912<br>Precision:<br>0.9<br>Recall:<br>0.826530612244898<br>F1 score:<br>0.8617021276595744<br>ROC AUC score:<br>0.9131861699378682 |

# 6501 Progress Report Group 4





GAN didn't really improve much of the performance of the classification model for now, we think this is because the generator is only producing low spectrum data. We will work on a few improved GAN algorithms to see if the problems are resolved.

**6501 Progress Report Group 4**

**10/05/2020**

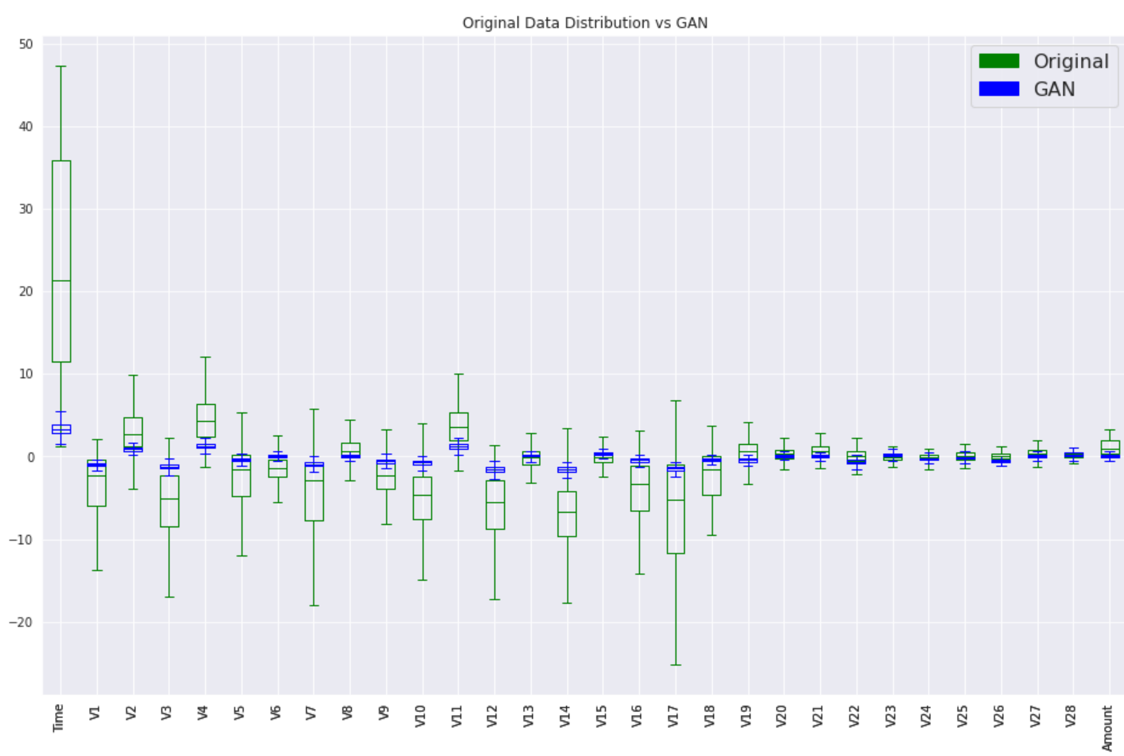
*WGAN development: Hao*

Why WGAN:
- Prevent mode collapse and gradient vanishing in vanilla GAN.
- Evaluate the difference between real and generated samples with wasserstein distance, using a score rather than label.

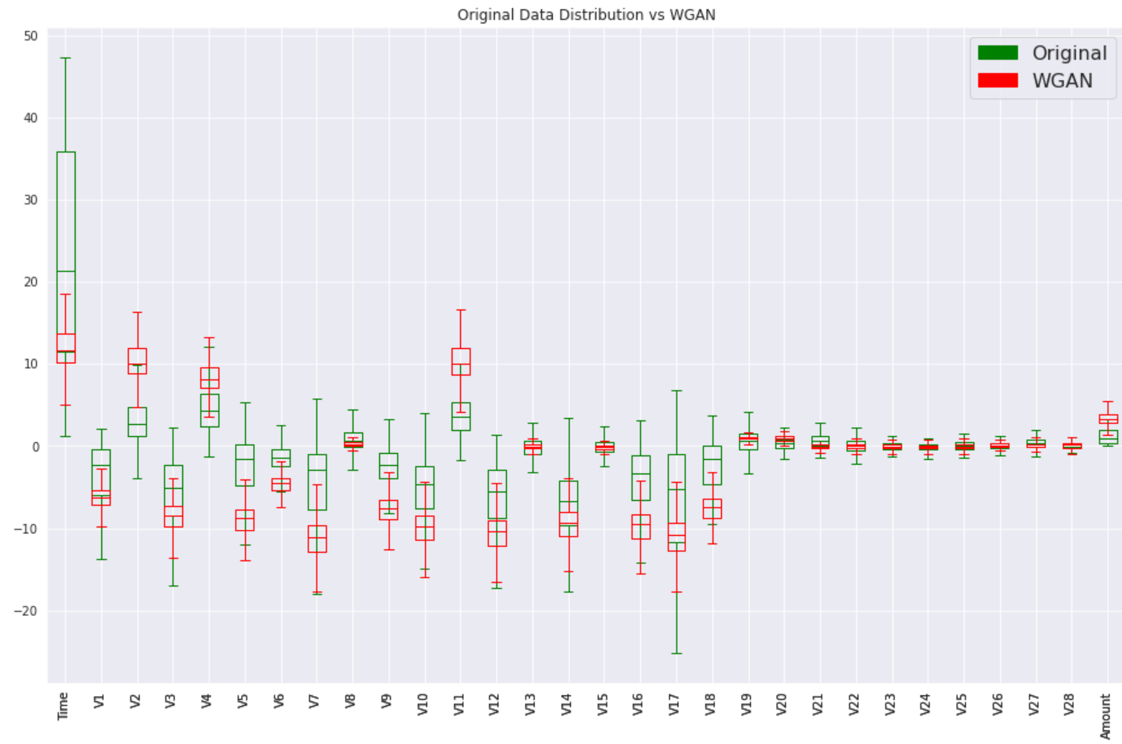
Implementation in Keras, compared to vanilla GAN:
- Loss function, no log
- D: no sigmoid
- Clip the weight of D (-c,c), if w>c, w=c, if w<-c, w=-c
- Use RMSProp
- Train D more than G

Training Parameters:  RMSprop(lr=0.00001), batch_size=128, train D twice

Generate 1000 ‘Fraud’, visually compare the data distribution using boxplot:
- GAN has very narrow spectrum of data distribution
- WGAN shows a wider distribution and close to the original data distribution
    - Good overlapping with the original distribution



# 6501 Progress Report Group 4



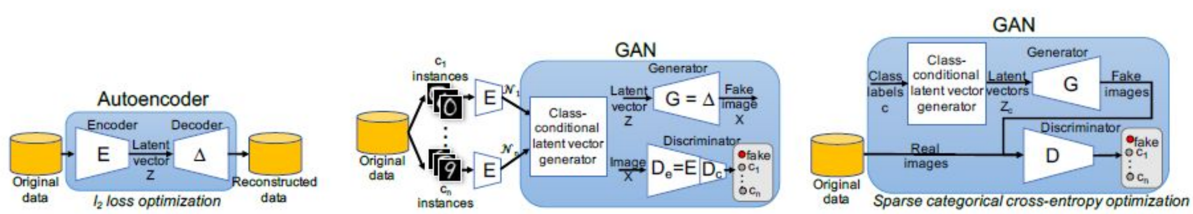

*BAGAN development: JUN*

Why BAGAN:

- It is an augmentation tool to restore the dataset balance by generating new minority-class data.
- It can learn the underlying features of the specific classification problem starting from all data and then to apply these features for the generation of new minority-class data.

Compared to original GAN:

- Discriminator has a single output that returns either a problem-specific class label c or the label fake.
- Coupling GAN and autoencoding techniques to provide a precise selection of the class conditioning and to better avoid mode collapse.




(a) Autoencoder training. (b) GAN initialization. (c) GAN training.

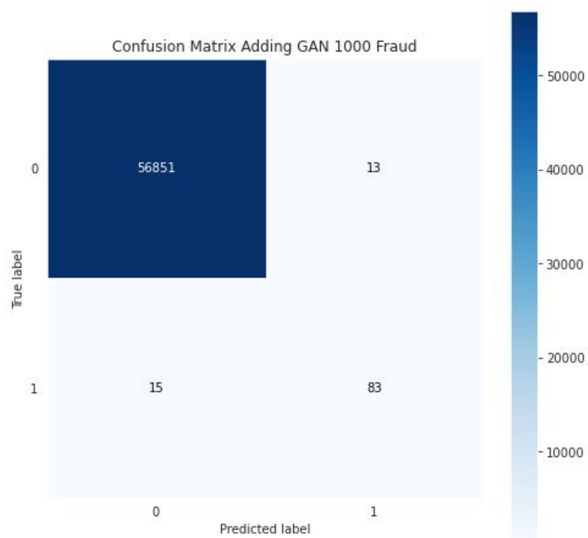
**6501 Progress Report Group 4**

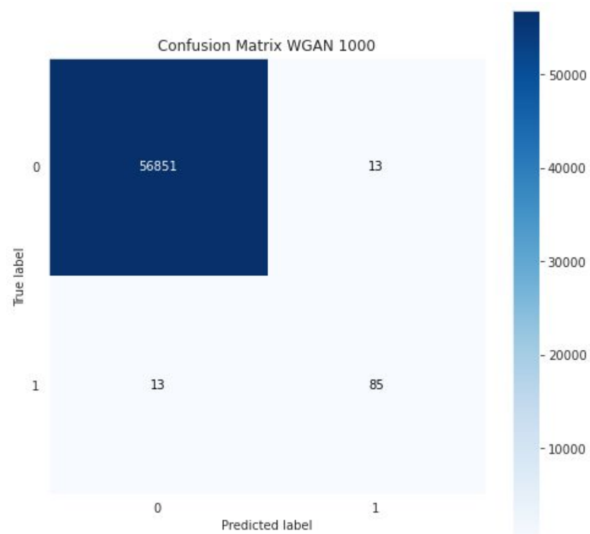
**10/12/2020**

*WGAN performance evaluation: Hao*

Obvious improvement on model performance

| Base ROS from Original data<br>Normal: 227451<br>Fraud: 227451 | Add GAN 1000 then ros.fit<br>Normal: 227451<br>Fraud: 227451 | Add WGAN 1000 then ros.fit<br>Normal: 227451<br>Fraud: 227451 |
|---|---|---|
| Accuracy:<br>0.9994908886626171<br>Precision:<br>0.8556701030927835<br>Recall:<br>0.8469387755102041<br>F1 score:<br>0.8512820512820514<br>ROC AUC score:<br>0.923346287023532 | Accuracy:<br>0.9995084442259752<br>Precision:<br>0.8645833333333334<br>Recall:<br>0.8469387755102041<br>F1 score:<br>0.8556701030927835<br>ROC AUC score:<br>0.9233550799329299 | **Accuracy:<br>0.9995435553526912<br>Precision:<br>0.8673469387755102<br>Recall:<br>0.8673469387755102<br>F1 score:<br>0.8673469387755102<br>ROC AUC score:<br>0.9335591615655828** |

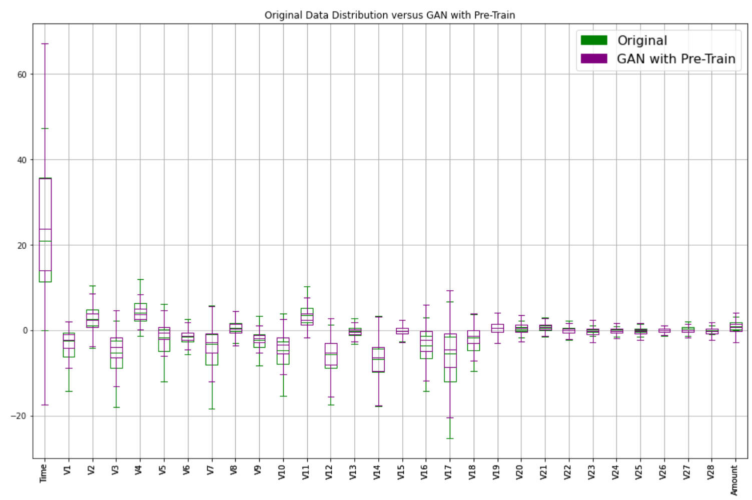


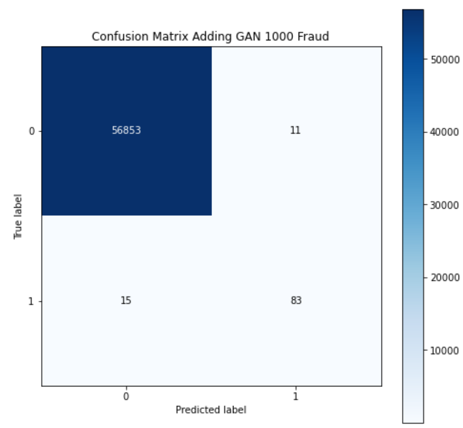



**6501 Progress Report Group 4**

*GAN with Pre-Train Performance : JUN*

Better than Original GAN, slightly worse than WGAN, adding 1000 generated fraud
Wider spectrum of data & Overlap well with the original data




| | GAN | GAN + AE |
|---|---|---|
| **Accuracy** | 0.999508 | 0.999544 |
| **Precision** | 0.864583 | 0.882979 |
| **Recall** | 0.846939 | 0.846939 |
| **F1 score** | 0.855670 | 0.864583 |
| **ROC AUC score** | 0.923355 | 0.923373 |

*Preliminary Presentation*

**6501 Progress Report Group 4**

**10/19/2020**

*WGAN_GP development: Hao*

Why WGAN_GP:

- WGAN weight clipping does not strictly fullil the 1-Lipschitz function requirement

  $$|f(x_1) - f(x_2)| \leq K|x_1 - x_2|.$$

  K =1, is 1-Lipschitz function, it has strong uniform continuity
- A differentiable function is 1-Lipschitz if and only if it has gradients with **norm at most 1 everywhere**
- WGAN_GP constrain the weight more effectively

Note: batch normalization is already removed, so we don't need to do anything here

Changes compared to WGAN

- Gradient penalty

  $$\underbrace{\lambda \mathop{\mathbb{E}}_{\hat{x} \sim \mathbb{P}_{\hat{x}}} \left[ (\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2 \right].}_{\text{Our gradient penalty}}$$

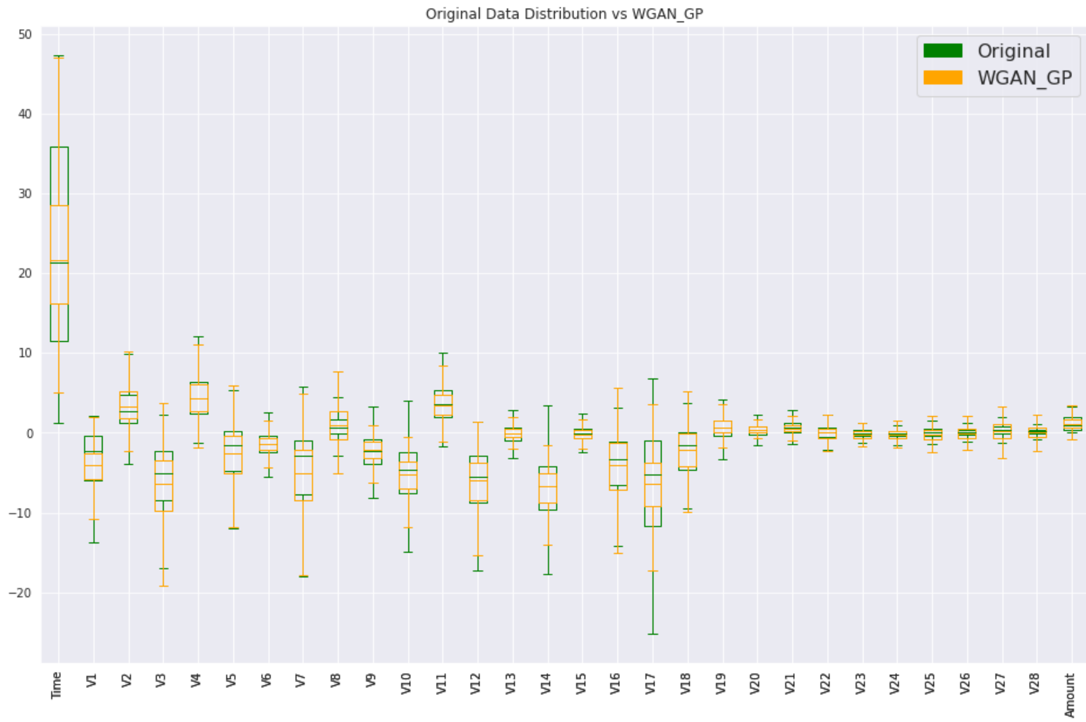
- Adam optimizer

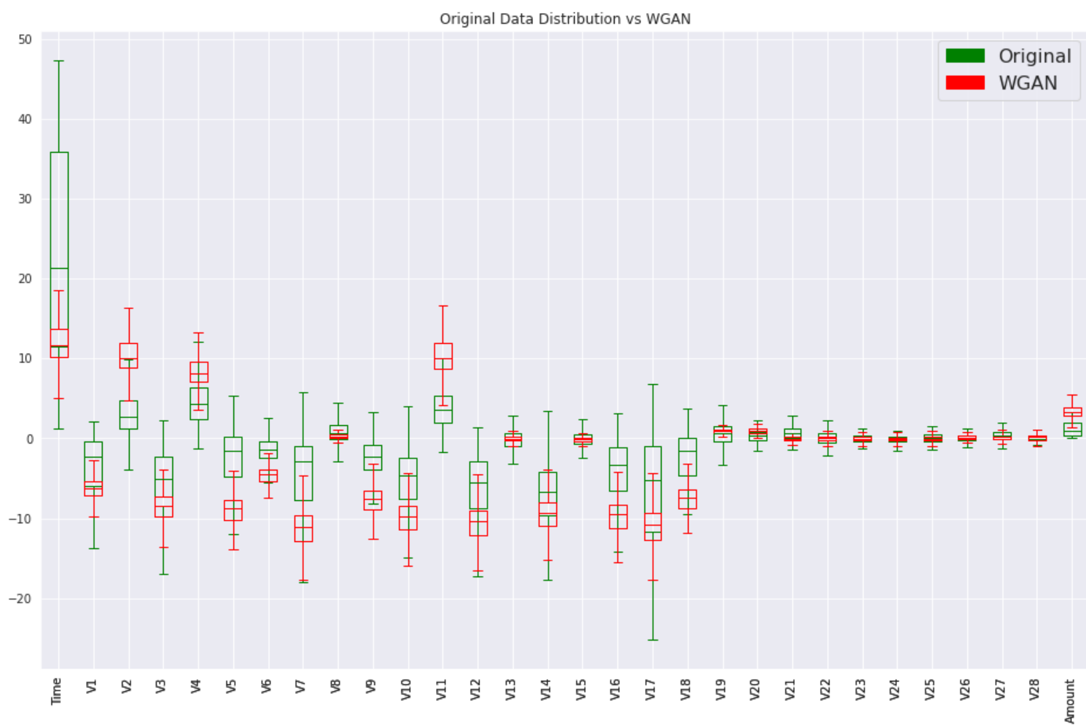
Training Parameters: ADAM(lr=0.00001), batch_size=128, gp_lambda=5

Generate 1000 'Fraud', visually compare the data distribution using boxplot:

- Wider range and better overlaps with original data compared to WGAN

# 6501 Progress Report Group 4




*BEGAN development: Jun*

Why BEGAN:

- a novel equilibrium method for balancing adversarial networks
- using an autoencoder as the discriminator
- using a variable kt∈ [0, 1] to control equilibrium(kt is adjusted at each step)

# 6501 Progress Report Group 4

$$\begin{cases} \mathcal{L}_D = \mathcal{L}(x) - k_t.\mathcal{L}(G(z_D)) & \text{for } \theta_D \\ \mathcal{L}_G = \mathcal{L}(G(z_G)) & \text{for } \theta_G \\ k_{t+1} = k_t + \lambda_k(\gamma\mathcal{L}(x) - \mathcal{L}(G(z_G))) & \text{for each training step } t \end{cases}$$

Derive a global measure of convergence by using the equilibrium concept:

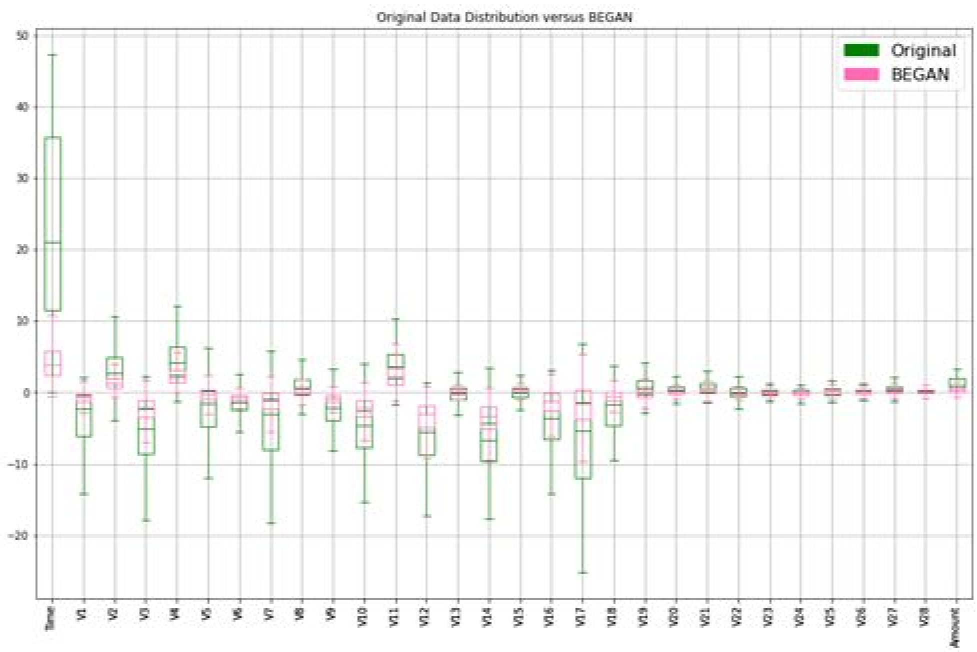
$$\mathcal{M}_{global} = \mathcal{L}(x) + |\gamma\mathcal{L}(x) - \mathcal{L}(G(z_G))|$$

This measure can be used to determine when the network has reached its final state or if the model has collapsed.

Training Parameters:  ADAM(lr=0.0001), batch_size=128, gp_lambda=5

Generate 1000 'Fraud', visually compare the data distribution using boxplot:

- Although better than the basic GAN, it does not overlap with the dataset well.
- There are cases where Time or Amount is less than 0.




Future work

- Fix and improve the loss function
- Adjustment parameters

# 6501 Progress Report Group 4

**10/26/2020**

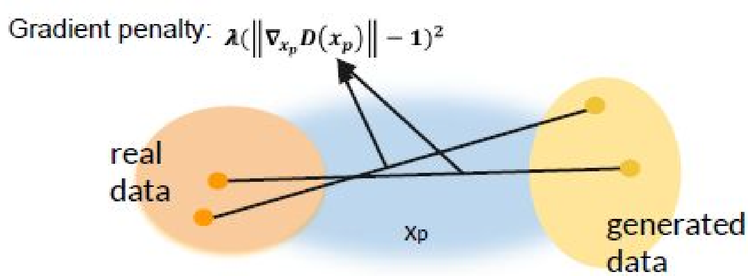
*WGAN_GP performance evaluation: Hao*

Interpolated part explanation

```
def gradient_penalty(self, batch_size, real_data, fake_data):
    """ Calculates the gradient penalty.
    This loss is calculated on an interpolated data
    and added to the discriminator loss.
    """
    # get the interplated image
    alpha = tf.random.normal([batch_size, 1, 1, 1], 0.0, 1.0)
    diff = fake_data - real_data
    interpolated = real_data + alpha * diff
```

The points interpolated between real and generated data should have a gradient norm of 1.

It's hard to get gradients with norm at most 1 everywhere, so we interpolate between real and generated samples. Instead of applying clipping, WGAN-GP applies a penalty if the gradient norm moves away from its target norm value 1.




Obvious further improvement on model performance with WGAN_GP, best model

| | Base | GAN | WGAN | WGAN_GP |
|---|---|---|---|---|
| **Accuracy** | 0.999491 | 0.999491 | 0.999544 | **0.999596** |
| **Precision** | 0.855670 | 0.864583 | 0.867347 | **0.894737** |
| **Recall** | 0.846939 | 0.846939 | 0.867347 | **0.867347** |
| **F1 score** | 0.851282 | 0.855670 | 0.867347 | **0.880829** |
| **ROC AUC score** | 0.923346 | 0.923355 | 0.933559 | **0.933586** |

**6501 Progress Report Group 4**


*BEGAN performance evaluation: Jun*

Discriminator: Autoencoder (Encoder + Decoder)
Generator: Decoder
Loss Function:

$$\begin{cases} \mathcal{L}_D = \mathcal{L}(x) - k_t.\mathcal{L}(G(z_D)) & \text{for } \theta_D \\ \mathcal{L}_G = \mathcal{L}(G(z_G)) & \text{for } \theta_G \\ k_{t+1} = k_t + \lambda_k(\gamma\mathcal{L}(x) - \mathcal{L}(G(z_G))) & \text{for each training step } t \end{cases}$$

$$\mathcal{M}_{global} = \mathcal{L}(x) + |\gamma\mathcal{L}(x) - \mathcal{L}(G(z_G))|$$

This measure can be used to determine when the network has reached its final state or if the model has collapsed.

# 6501 Progress Report Group 4

K∈[0,1]

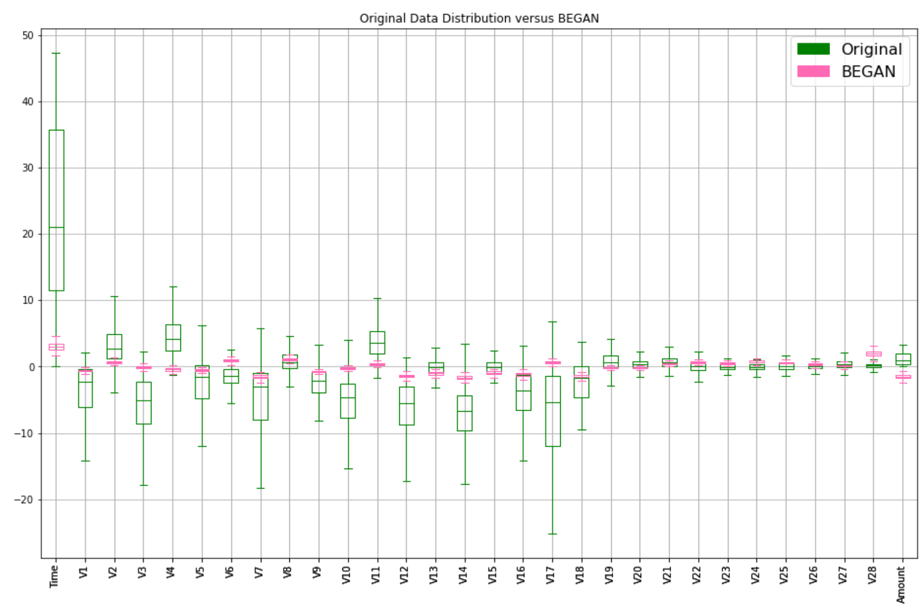
k=1e-05

```
EPOCH #  20 ---------------------------------------------------
Steps (10 / 100): [loss_D: 23.763582] [Loss_G: 5191.047363] [M_global: 5191.140451]
Steps (20 / 100): [loss_D: 23.283887] [Loss_G: 5037.947266] [M_global: 5038.038371]
Steps (30 / 100): [loss_D: 20.021150] [Loss_G: 5360.807129] [M_global: 5360.880726]
Steps (40 / 100): [loss_D: 19.312572] [Loss_G: 5416.488770] [M_global: 5416.558218]
Steps (50 / 100): [loss_D: 21.133590] [Loss_G: 5494.187012] [M_global: 5494.264138]
Steps (60 / 100): [loss_D: 21.643043] [Loss_G: 5638.911621] [M_global: 5638.991511]
Steps (70 / 100): [loss_D: 20.518601] [Loss_G: 5681.296387] [M_global: 5681.370212]
Steps (80 / 100): [loss_D: 19.534069] [Loss_G: 5762.511230] [M_global: 5762.579990]
Steps (90 / 100): [loss_D: 21.824670] [Loss_G: 5780.583496] [M_global: 5780.663860]
Steps (100 / 100): [loss_D: 19.115582] [Loss_G: 6058.291992] [M_global: 6058.358950]
```

| | Time |
|---|---|
| count | 1000.000000 |
| mean | 45316.074219 |
| std | 9665.684570 |
| min | 18976.361328 |
| 25% | 38610.062500 |
| 50% | 45014.445312 |
| 75% | 51699.350586 |
| max | 78984.054688 |

k=1

| | Time |
|---|---|
| count | 1000.000000 |
| mean | 2.994987 |
| std | 0.581427 |
| min | 1.631465 |
| 25% | 2.582442 |
| 50% | 2.933775 |
| 75% | 3.378483 |
| max | 5.222883 |

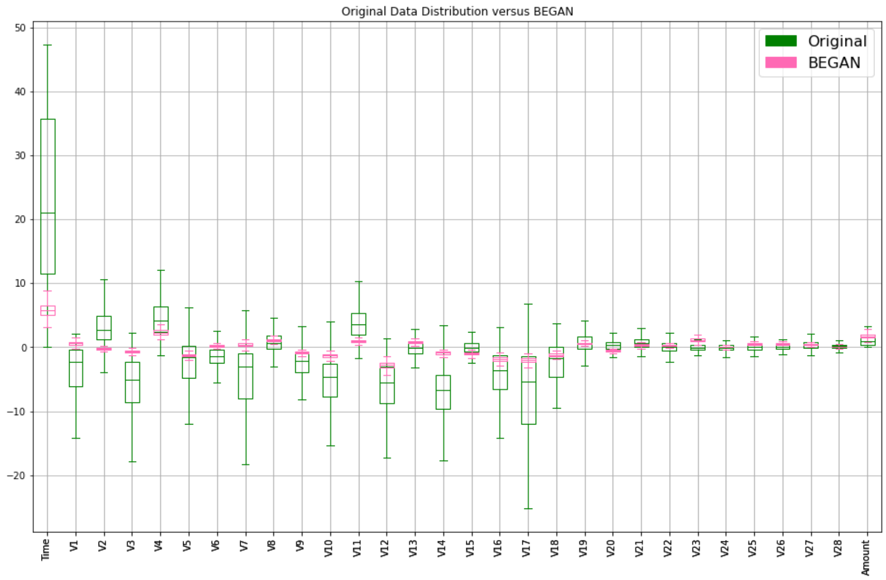



k=0.08

```
EPOCH #  20 ---------------------------------------------------
Steps (10 / 100): [loss_D: 23.402295] [Loss_G: 0.590853] [M_global: 0.687436]
Steps (20 / 100): [loss_D: 22.523965] [Loss_G: 0.568527] [M_global: 0.661421]
Steps (30 / 100): [loss_D: 22.351444] [Loss_G: 0.538081] [M_global: 0.631413]
Steps (40 / 100): [loss_D: 21.146445] [Loss_G: 0.510166] [M_global: 0.598388]
Steps (50 / 100): [loss_D: 22.483769] [Loss_G: 0.456806] [M_global: 0.552923]

Steps (60 / 100): [loss_D: 22.652090] [Loss_G: 0.423857] [M_global: 0.522124]
Steps (70 / 100): [loss_D: 24.288411] [Loss_G: 0.413211] [M_global: 0.520235]
Steps (80 / 100): [loss_D: 24.751768] [Loss_G: 0.415951] [M_global: 0.524934]
Steps (90 / 100): [loss_D: 22.647221] [Loss_G: 0.456992] [M_global: 0.554145]
Steps (100 / 100): [loss_D: 23.766275] [Loss_G: 0.482337] [M_global: 0.584548]
```

| | Time |
|---|---|
| count | 1000.000000 |
| mean | 5.871967 |
| std | 1.094325 |
| min | 3.122333 |
| 25% | 5.059084 |
| 50% | 5.770874 |
| 75% | 6.568066 |
| max | 9.851136 |

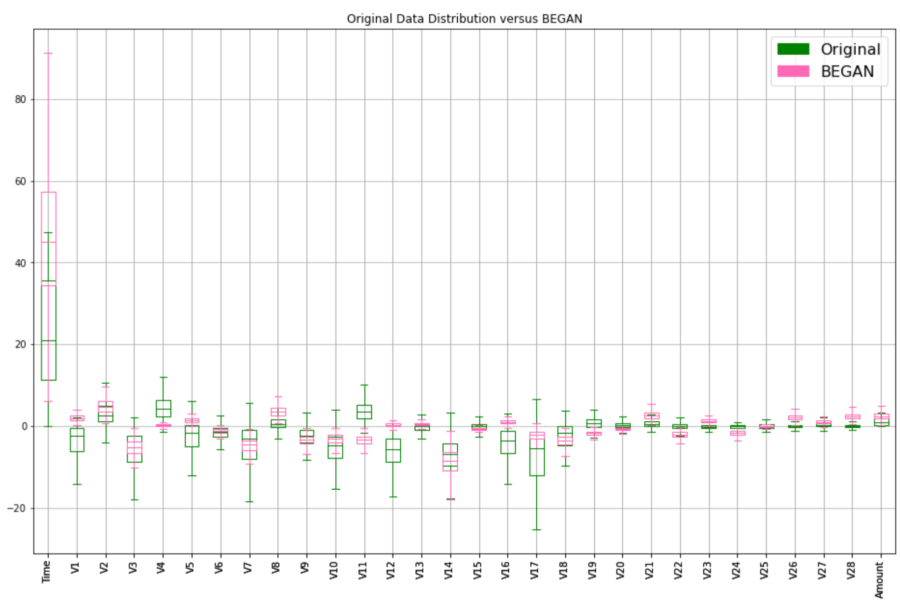
# 6501 Progress Report Group 4




k=0.06, batch_size=64 (previous batch_size=128)

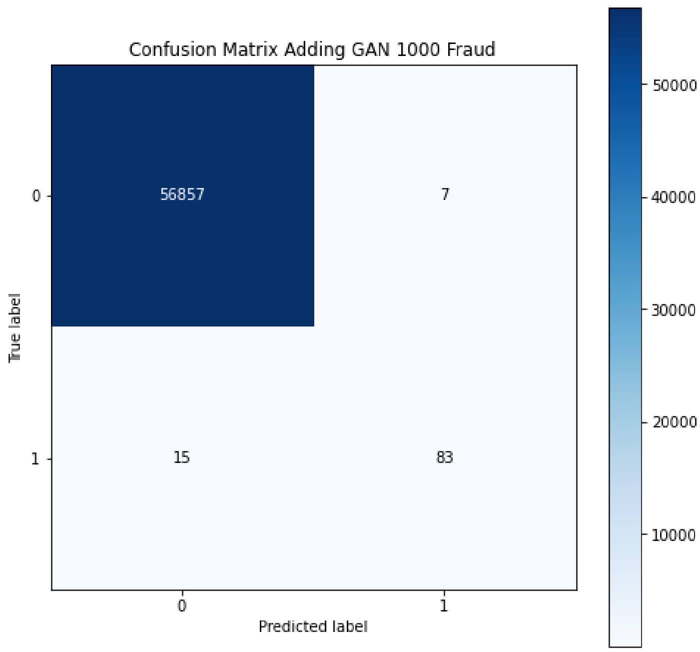
```
EPOCH #  20 ------------------------------------------------
Steps (10 / 100): [loss_D: -10.341525] [Loss_G: 12.585917] [M_global: 12.199375]
Steps (20 / 100): [loss_D: -3.326088] [Loss_G: 11.233190] [M_global: 10.936512]
Steps (30 / 100): [loss_D: 20.852566] [Loss_G: 7.410180] [M_global: 7.314507]
Steps (40 / 100): [loss_D: 30.622616] [Loss_G: 4.393467] [M_global: 4.416713]
Steps (50 / 100): [loss_D: 29.780138] [Loss_G: 3.984214] [M_global: 4.020640]

Steps (60 / 100): [loss_D: 28.448325] [Loss_G: 4.266831] [M_global: 4.289491]
Steps (70 / 100): [loss_D: 17.527455] [Loss_G: 4.801738] [M_global: 4.757676]
Steps (80 / 100): [loss_D: 16.920543] [Loss_G: 5.242315] [M_global: 5.196826]
Steps (90 / 100): [loss_D: 15.595847] [Loss_G: 4.730126] [M_global: 4.674268]
Steps (100 / 100): [loss_D: 18.740515] [Loss_G: 4.700145] [M_global: 4.672210]
```

| | Time |
|---|---|
| **count** | 1000.000000 |
| **mean** | 46.818562 |
| **std** | 16.844614 |
| **min** | 6.206626 |
| **25%** | 34.534291 |
| **50%** | 45.042454 |
| **75%** | 57.417752 |
| **max** | 114.041725 |



# 6501 Progress Report Group 4

| | Base | GAN | BEGAN |
|---|---|---|---|
| **Accuracy** | 0.999491 | 0.999491 | **0.999613** |
| **Precision** | 0.855670 | 0.864583 | **0.922222** |
| **Recall** | 0.846939 | 0.846939 | 0.846939 |
| **F1 score** | 0.851282 | 0.855670 | **0.882979** |
| **ROC AUC score** | 0.923346 | 0.923355 | **0.923408** |



# 6501 Progress Report Group 4