Liat Cohen- 205595283
Amit Shakarchy- 313278889

DEEP LEARNING
ASSIGNMENT4

DATA PREPROCESSING

Diabetes dataset preprocessing:

- We used a MinMaxScaler for scaling the data.
- The target feature (“class”) was converted to ordinal values: 0 denotes “tested_negative” and 1 denotes “tested_positive”.

German credit dataset preprocessing:

- We used MinMaxScaler to scale all numeric features.
- We used ordinal encoding for the categorical features with Label Encoder.
- We converted the target feature (the 21st feature) to ordinal values of 0/1.

PART 1- EXPERIMENTAL SETUP

Data:

We used the entire dataset to run each experiment, as written in the instructions

Network architecture:

We implemented a GAN, consisting of a generator and a discriminator.

- The generator is built of dense and dropout layers, for regularization. It generates a new sample with n features. In this case- 8 features, for the diabetes dataset.

```
Model: "Generator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_13 (InputLayer)       [(None, 9)]               0

 dense_54 (Dense)            (None, 256)               2560

 batch_normalization_18 (Bat  (None, 256)              1024
 chNormalization)

 dense_55 (Dense)            (None, 128)               32896

 batch_normalization_19 (Bat  (None, 128)              512
 chNormalization)

 dense_56 (Dense)            (None, 64)                8256

 batch_normalization_20 (Bat  (None, 64)               256
 chNormalization)

 dense_57 (Dense)            (None, 32)                2080

 dense_58 (Dense)            (None, 9)                 297

=================================================================
Total params: 47,881
Trainable params: 46,985
Non-trainable params: 896
_________________________________________________________________
```

- The discriminator- is built of dense and dropout layers, for regularization. It receives a sample and determines whether it is real or fake.

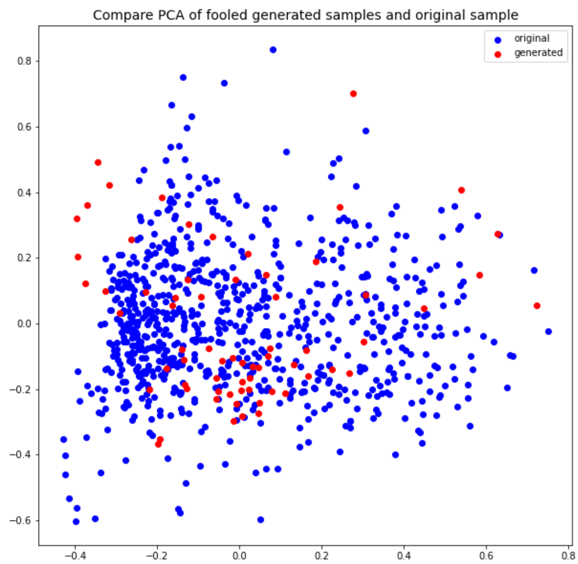
```
Model: "Discriminator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_14 (InputLayer)       [(None, 9)]               0

 dense_59 (Dense)            (None, 64)                640

 dropout_12 (Dropout)        (None, 64)                0

 dense_60 (Dense)            (None, 128)               8320

 dropout_13 (Dropout)        (None, 128)               0

 dense_61 (Dense)            (None, 512)               66048

 dense_62 (Dense)            (None, 1)                 513

=================================================================
Total params: 75,521
Trainable params: 75,521
Non-trainable params: 0
_________________________________________________________________
```

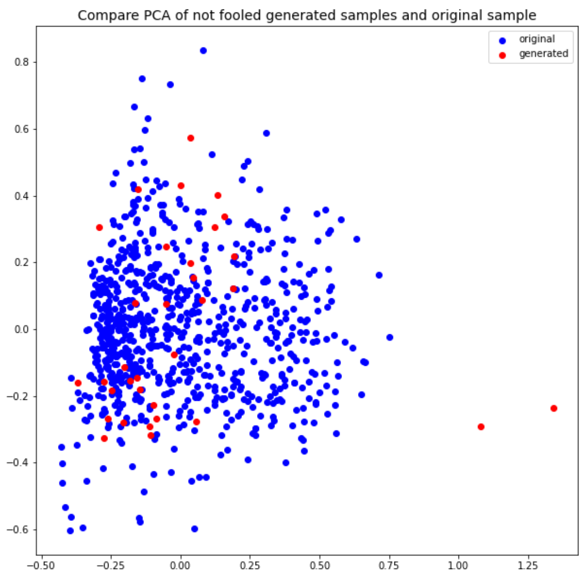
Additional parameters:

- Batch size - 64.
- Epochs- 250.
- Dropout rate- 0.2 – for regularization.
- Loss- we used Binary cross-entropy loss for both the generator and discriminator.
- Optimizer- We used Adam optimizer with a learning rate of 0.0001.

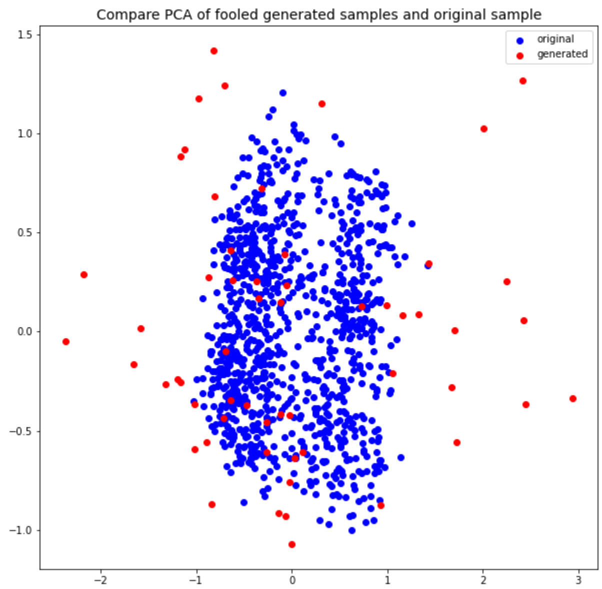
a. We will now provide several samples that managed to fool the discriminator, and several samples that didn't fool it. We used PCA visualization to determine whether the samples that fooled the discriminator are similar to samples from the original data.

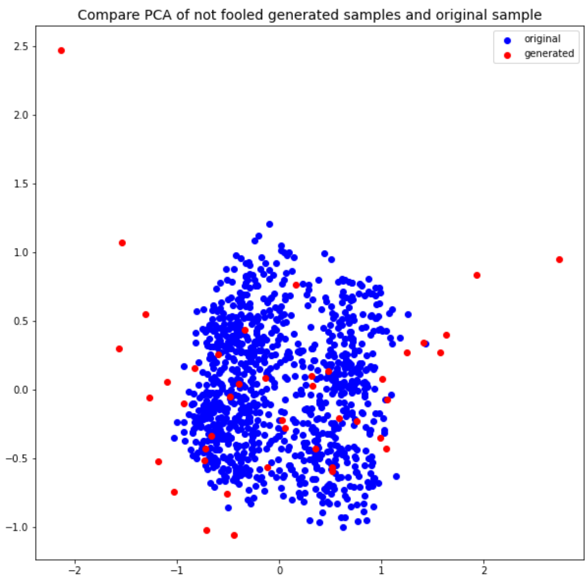
Diabetes dataset:



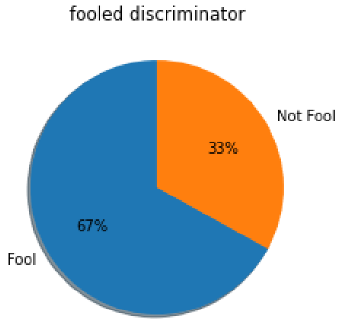


German credit dataset:





b. We generated 100 random samples (for each dataset). We added the discriminator's probabilities to the generated data frame. We will show now how many samples passed as 'real' ones:
<u>Diabetes dataset:</u>
Out of 100 samples, 67 fooled the discriminator, and 33 didn't.



an example of simples that fooled the discriminator:

	preg	plas	pres	skin	insu	mass	pedi	age	class	disc_pred
0	10.681387	127.236130	81.358795	20.090235	-22.716509	31.234249	0.280346	52.269047	0	0.568550
1	6.730704	77.755920	68.579910	8.147600	-15.646030	26.966961	0.242242	34.571884	0	0.565561
4	9.480645	114.728638	78.150055	22.443827	26.162586	36.945728	0.425324	25.755146	1	0.586105
5	1.161134	120.284500	88.714493	40.766239	107.965218	58.005264	1.007045	16.115887	0	0.584314
6	2.117890	93.850769	51.969650	21.101570	131.929199	25.386208	0.252070	22.729452	0	0.527349
...	...	...	...	...	...	...	...	...	...	...
93	11.924850	128.104996	114.089523	17.591732	-7.274595	33.164734	0.235017	59.843853	0	0.542390
94	4.015940	93.064438	101.585533	7.994150	-6.783516	29.153801	0.421438	36.727371	0	0.575249
95	5.638880	91.115883	59.275555	28.872055	134.783249	25.175053	0.329968	31.972000	0	0.525495
96	5.889933	137.680588	75.530968	22.862185	26.950876	30.214382	0.196639	42.218815	1	0.545636
99	3.689390	133.911774	78.993172	20.589691	-12.113835	40.269661	0.454405	22.934742	0	0.549368

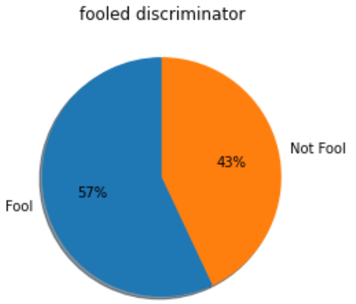
81 rows × 10 columns

an example of simples that didn't fool the discriminator:

	preg	plas	pres	skin	insu	mass	pedi	age	class	disc_pred
2	2.973588	97.836624	60.830891	32.453697	70.012100	31.131964	0.291330	22.608315	1	0.341797
3	6.982035	68.650833	19.863064	-7.718239	-123.823288	27.223410	0.453342	43.532585	0	0.317215
8	3.227352	72.171837	47.052799	40.001289	93.690865	26.492846	0.434963	17.117107	0	0.493713
11	3.330611	84.821152	61.337147	26.213797	-15.373146	22.097445	0.458688	25.822393	1	0.368579
17	4.418523	74.839157	45.686047	3.972663	-15.192578	12.930507	1.035251	29.288536	0	0.376376
23	12.299655	77.631409	82.305824	18.612576	44.949406	32.914261	0.259932	55.972496	0	0.462448
27	5.851851	87.565987	49.672131	17.827156	-24.016897	24.584028	0.404530	29.257362	0	0.247750
33	2.109506	127.800529	50.033981	-5.148126	42.564774	23.394566	0.864445	24.337769	0	0.495327
41	2.280150	93.066788	38.724895	7.671420	-30.825083	19.004581	1.183693	16.459177	1	0.379245
56	4.599531	128.913788	56.246029	20.388792	219.818909	43.030754	0.135866	28.586727	0	0.329494
57	3.386130	163.340622	111.377869	22.755783	41.339878	43.438011	0.903184	32.675610	0	0.495721
59	4.279352	220.075821	79.791946	18.178381	33.234810	36.997234	0.763294	61.724060	1	0.494882

German credit dataset:

Out of 100 samples, 34 fooled the discriminator, and 66 didn't.



an example of simples that fooled the discriminator:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	disc_pred
2	A13	14.0	A33	A46	3079.0	A61	A72	1.0	A94	A101	3.0	A122	23.0	A143	A151	2.0	A172	1.0	A191	A201	0	0.587319
4	A12	11.0	A31	A48	590.0	A63	A75	4.0	A93	A101	4.0	A121	28.0	A143	A153	1.0	A172	1.0	A191	A201	1	0.516651
5	A12	20.0	A32	A41	1934.0	A64	A74	4.0	A93	A101	4.0	A124	42.0	A142	A152	1.0	A173	2.0	A192	A201	0	0.542075
6	A13	40.0	A32	A43	3660.0	A63	A74	4.0	A94	A101	4.0	A124	34.0	A142	A153	1.0	A173	1.0	A192	A201	1	0.543413
8	A13	72.0	A34	A49	1169.0	A63	A72	2.0	A94	A102	3.0	A123	24.0	A143	A153	3.0	A172	1.0	A192	A201	1	0.686263
9	A12	22.0	A33	A410	1893.0	A63	A71	2.0	A93	A102	2.0	A123	55.0	A142	A153	2.0	A172	1.0	A191	A201	1	0.577049
10	A12	20.0	A32	A40	2828.0	A63	A74	3.0	A93	A101	4.0	A124	32.0	A142	A152	2.0	A173	1.0	A192	A201	0	0.559446
11	A12	12.0	A32	A41	1922.0	A63	A74	3.0	A92	A101	3.0	A122	30.0	A142	A151	1.0	A172	1.0	A191	A201	0	0.504686
13	A12	9.0	A33	A48	2133.0	A62	A74	1.0	A93	A101	2.0	A124	32.0	A143	A151	1.0	A172	1.0	A191	A201	0	0.520595
14	A11	12.0	A34	A410	2647.0	A62	A75	2.0	A92	A102	4.0	A124	27.0	A143	A152	3.0	A173	1.0	A191	A201	0	0.510752
15	A12	16.0	A34	A43	2924.0	A61	A75	2.0	A92	A101	3.0	A124	22.0	A143	A152	2.0	A173	1.0	A191	A201	0	0.524482
17	A11	13.0	A32	A42	2315.0	A62	A72	2.0	A92	A102	2.0	A121	41.0	A142	A151	1.0	A172	1.0	A191	A201	1	0.508350

an example of simples that didn't fool the discriminator:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	disc_pred
0	A14	10.0	A32	A40	2238.0	A63	A73	1.0	A92	A101	2.0	A122	30.0	A143	A152	1.0	A173	1.0	A191	A201	0	0.498389
1	A14	26.0	A33	A41	1882.0	A65	A75	4.0	A94	A101	4.0	A124	32.0	A142	A152	1.0	A174	2.0	A192	A201	1	0.492212
3	A12	6.0	A33	A42	2279.0	A63	A74	3.0	A92	A101	2.0	A122	22.0	A143	A152	2.0	A173	1.0	A191	A201	0	0.475797
7	A13	7.0	A34	A43	1913.0	A62	A75	2.0	A92	A101	2.0	A124	29.0	A143	A151	1.0	A173	1.0	A191	A201	0	0.489265
12	A14	45.0	A33	A410	3622.0	A64	A75	4.0	A93	A102	4.0	A124	48.0	A142	A153	1.0	A174	1.0	A192	A201	1	0.477241
16	A14	15.0	A33	A40	1908.0	A61	A74	2.0	A92	A101	3.0	A124	27.0	A143	A153	2.0	A173	1.0	A191	A201	0	0.477569
19	A11	10.0	A34	A45	2359.0	A62	A75	2.0	A93	A101	3.0	A124	20.0	A143	A152	2.0	A173	1.0	A191	A201	0	0.493915
21	A14	14.0	A33	A43	9857.0	A63	A75	3.0	A93	A102	2.0	A124	29.0	A143	A152	1.0	A173	1.0	A191	A201	0	0.494358
22	A12	7.0	A32	A43	2273.0	A62	A74	1.0	A92	A101	3.0	A124	29.0	A143	A151	1.0	A172	1.0	A191	A201	0	0.487322
23	A14	26.0	A33	A410	1961.0	A65	A74	4.0	A92	A101	3.0	A124	35.0	A142	A152	1.0	A174	2.0	A192	A201	1	0.481096
25	A14	13.0	A32	A410	1572.0	A64	A75	4.0	A93	A101	4.0	A123	50.0	A143	A152	2.0	A173	1.0	A192	A201	0	0.419332
26	A12	12.0	A32	A43	3331.0	A62	A74	1.0	A92	A101	3.0	A123	31.0	A143	A152	2.0	A172	1.0	A191	A201	0	0.494250
28	A14	15.0	A32	A41	1922.0	A63	A74	3.0	A93	A101	4.0	A122	27.0	A142	A151	1.0	A173	1.0	A191	A201	0	0.446267

c. Graphs describing the loss of the generator and the discriminator:
Diabetes dataset:

German credit dataset:

As we can see in the graphs, both models converged in around 50 epochs. In both, the discriminator's loss is consistently smaller.

Network architecture:

We implemented a ‘twisted GAN’, consists of a generator and a discriminator, as written in the instructions.

- The generator- receives two inputs- a noise vector, and a confidence score c. The generator concatenates the c vector with each layer- for a better “knowledge” of the confidence score.

```
Model: "model_14"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_15 (InputLayer)          [(None, 5)]          0           []

 input_16 (InputLayer)          [(None, 1)]          0           []

 concatenate (Concatenate)      (None, 6)            0           ['input_15[0][0]',
                                                                  'input_16[0][0]']

 dense_63 (Dense)               (None, 512)          3584        ['concatenate[0][0]']

 concatenate_1 (Concatenate)    (None, 513)          0           ['dense_63[0][0]',
                                                                  'input_16[0][0]']

 batch_normalization_21 (BatchN  (None, 513)         2052        ['concatenate_1[0][0]']
 ormalization)

 dense_64 (Dense)               (None, 256)          131584      ['batch_normalization_21[0][0]']

 concatenate_2 (Concatenate)    (None, 257)          0           ['dense_64[0][0]',
                                                                  'input_16[0][0]']

 batch_normalization_22 (BatchN  (None, 257)         1028        ['concatenate_2[0][0]']
 ormalization)

 dense_65 (Dense)               (None, 128)          33024       ['batch_normalization_22[0][0]']

 concatenate_3 (Concatenate)    (None, 129)          0           ['dense_65[0][0]',
                                                                  'input_16[0][0]']

 batch_normalization_23 (BatchN  (None, 129)         516         ['concatenate_3[0][0]']
 ormalization)

 dense_66 (Dense)               (None, 64)           8320        ['batch_normalization_23[0][0]']

 concatenate_4 (Concatenate)    (None, 65)           0           ['dense_66[0][0]',
                                                                  'input_16[0][0]']

 batch_normalization_24 (BatchN  (None, 65)          260         ['concatenate_4[0][0]']
 ormalization)

 dense_67 (Dense)               (None, 32)           2112        ['batch_normalization_24[0][0]']

 concatenate_5 (Concatenate)    (None, 33)           0           ['dense_67[0][0]',
                                                                  'input_16[0][0]']

 batch_normalization_25 (BatchN  (None, 33)          132         ['concatenate_5[0][0]']
 ormalization)

 dense_68 (Dense)               (None, 16)           544         ['batch_normalization_25[0][0]']

 dense_69 (Dense)               (None, 8)            136         ['dense_68[0][0]']

==================================================================================================
Total params: 183,292
Trainable params: 181,298
          Non-trainable params: 1,994
```

- The discriminator- receives a sample generated by the generator and the confidence score c. It concatenates the inputs and determines whether the sample is real/fake.

```
________________________________________________________________________________________________
Model: "model_15"
________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
================================================================================================
 input_17 (InputLayer)          [(None, 8)]          0           []

 input_18 (InputLayer)          [(None, 1)]          0           []

 concatenate_6 (Concatenate)    (None, 9)            0           ['input_17[0][0]',
                                                                  'input_18[0][0]']

 batch_normalization_26 (BatchN  (None, 9)           36          ['concatenate_6[0][0]']
 ormalization)

 dense_70 (Dense)               (None, 32)           320         ['batch_normalization_26[0][0]']

 dense_71 (Dense)               (None, 16)           528         ['dense_70[0][0]']

 concatenate_7 (Concatenate)    (None, 17)           0           ['dense_71[0][0]',
                                                                  'input_18[0][0]']

 dense_72 (Dense)               (None, 1)            18          ['concatenate_7[0][0]']

================================================================================================
Total params: 902
Trainable params: 884
Non-trainable params: 18
________________________________________________________________________________________________
```

Additional parameters:

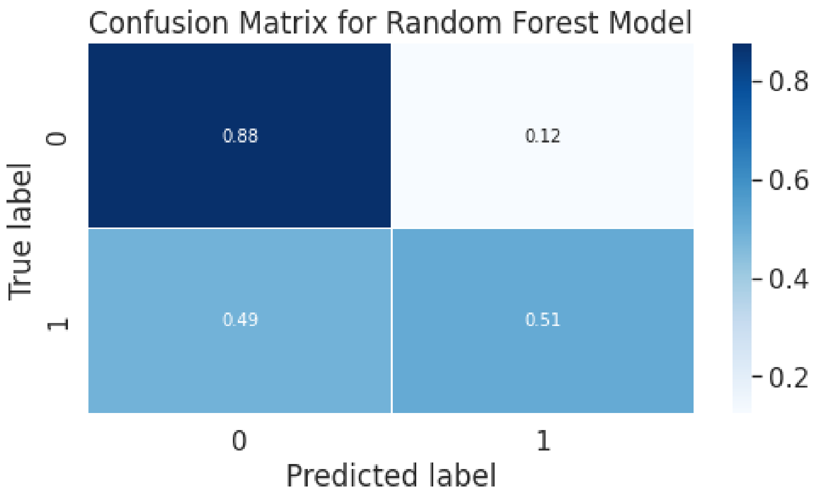
Same as the original architecture.

Random Forest classifier:

Was trained using a random split of 70%/30%.

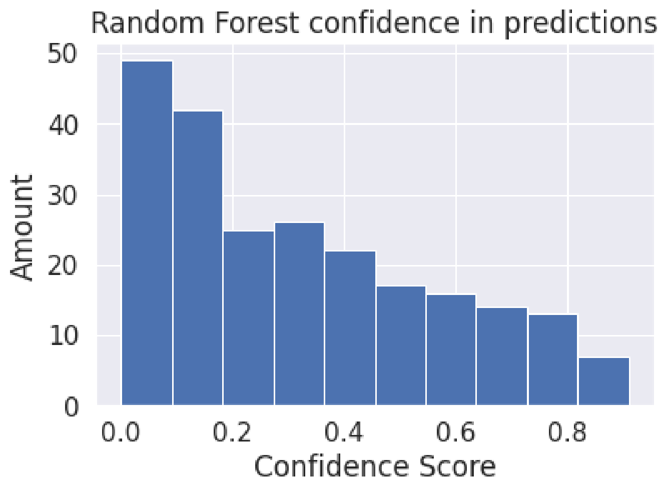
- Classifier's performance- Diabetes dataset:

accuracy	0.753
precision	0.73
recall	0.69
F1-score	0.7



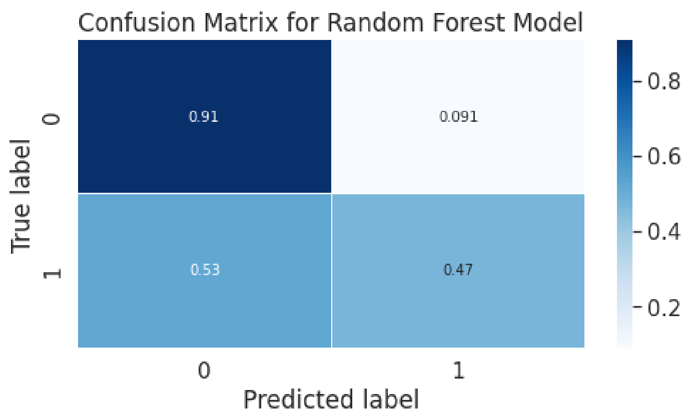
- Confidence in predictions:

Maximum Confidence	0.91
Minimum Confidence	0
Average Confidence	0.32
Median Confidence	0.27



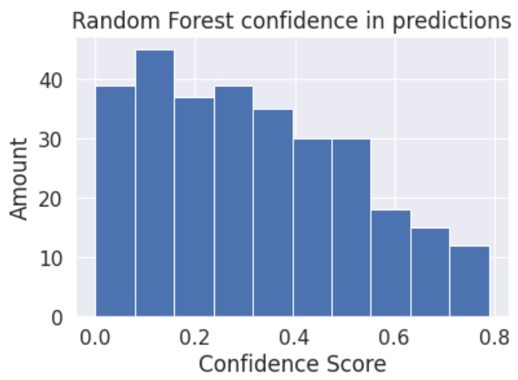
- Classifier's performance- German Credit dataset:

accuracy	0.776
precision	0.75
recall	0.69
F1-score	0.71



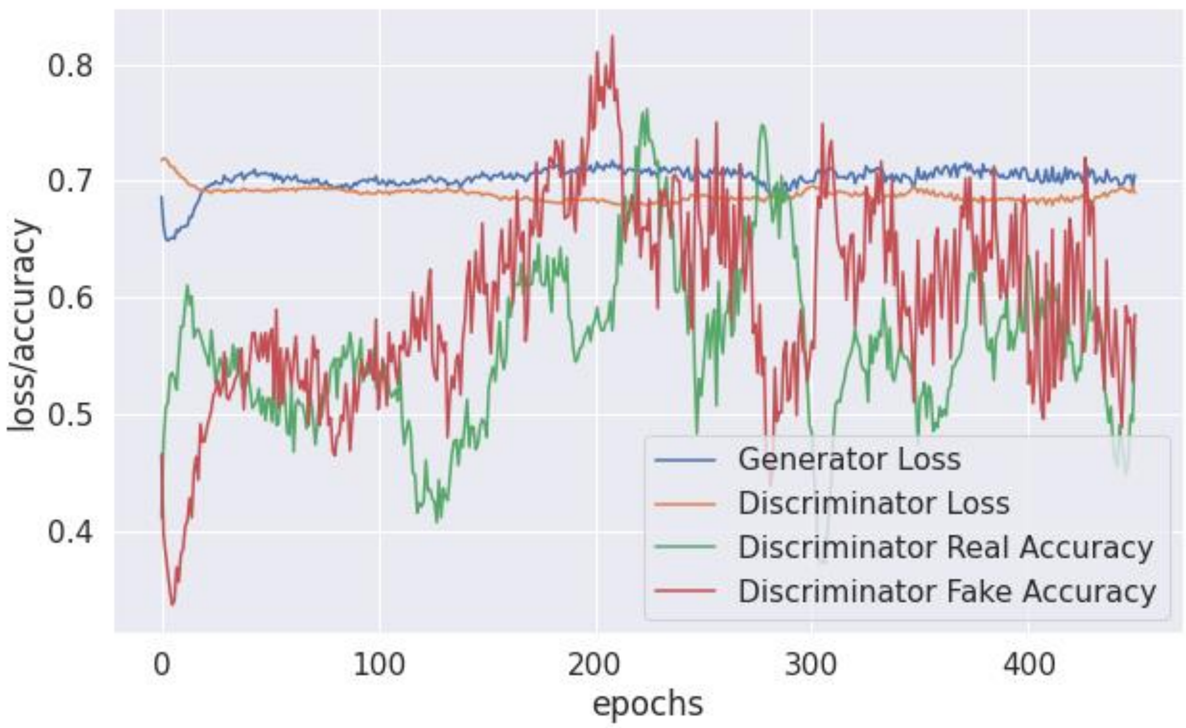
- Confidence in predictions:

Maximum Confidence	0.79
Minimum Confidence	0
Average Confidence	0.32
Median Confidence	0.29



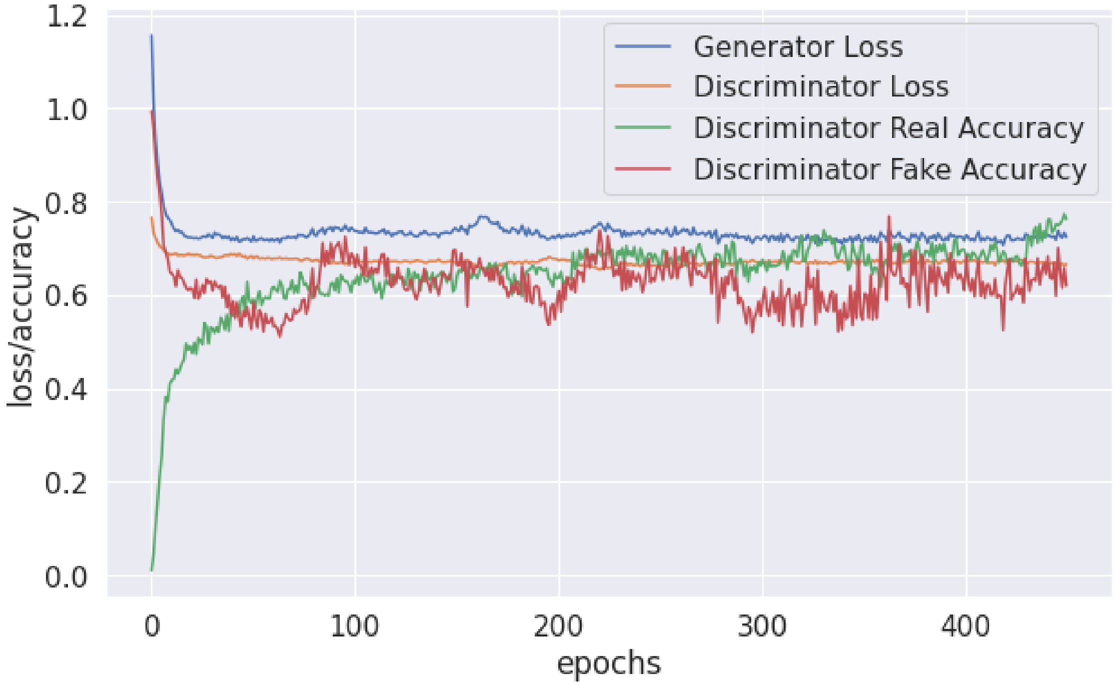
Twisted GAN performances:

- Diabetes dataset:



It seems that the discriminator is leading- its loss is consistently better than the generator's.

- German Credit dataset:



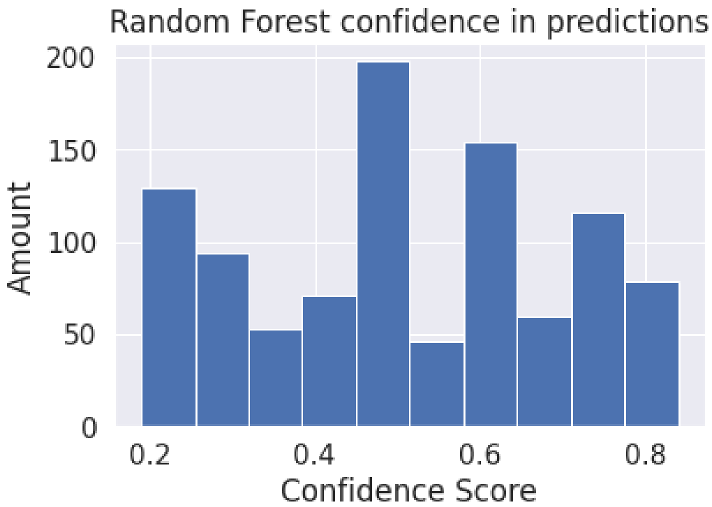
It seems that the discriminator is leading- its loss is consistently better than the generator's.

In this section, we generated 1,000 samples from our trained GAN, with uniformly sampled confidence scores. We will now show some statistics on the score distributions:

- Diabetes dataset:

    - Confidence in predictions:

Maximum Confidence	0.84
Minimum Confidence	0.19
Average Confidence	0.51
Median Confidence	0.5



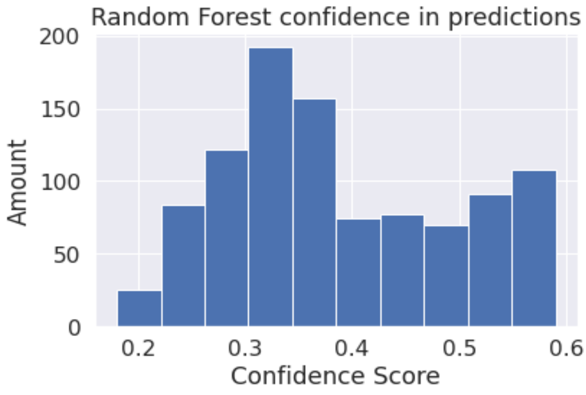
As we can see, results are "spreader" more equally, comparing to the original architecture.

- Pearson correlation of the prediction of the black-box model and the uniformly sampled confidence values the twisted GAN was fed with: **0.931**.
- We calculated the accuracy score, of the confidence value that was fed to the twisted GAN in compared to the black-box model's predictions: **81.50%**

- German Credit dataset:

    - Confidence in predictions:

Maximum Confidence	0.59
Minimum Confidence	0.18
Average Confidence	0.39
Median Confidence	0.36

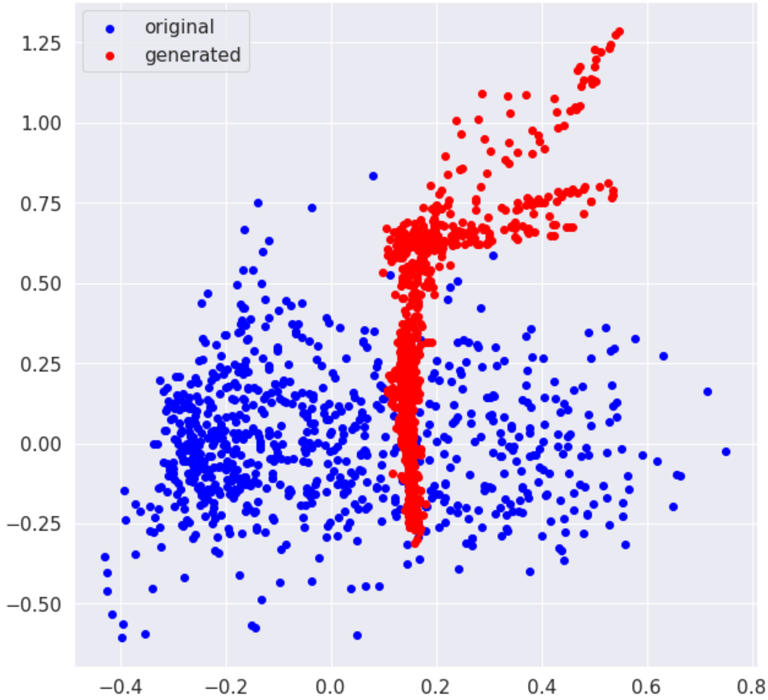


- Pearson correlation of the prediction of the black-box model and the uniformly sampled confidence values the twisted GAN was fed with: **0.798**.
- We calculated the accuracy score, of the confidence value that was fed to the twisted GAN in compared to the black-box model's predictions: **70.60%**

MODE COLLAPSE OF THE MODEL

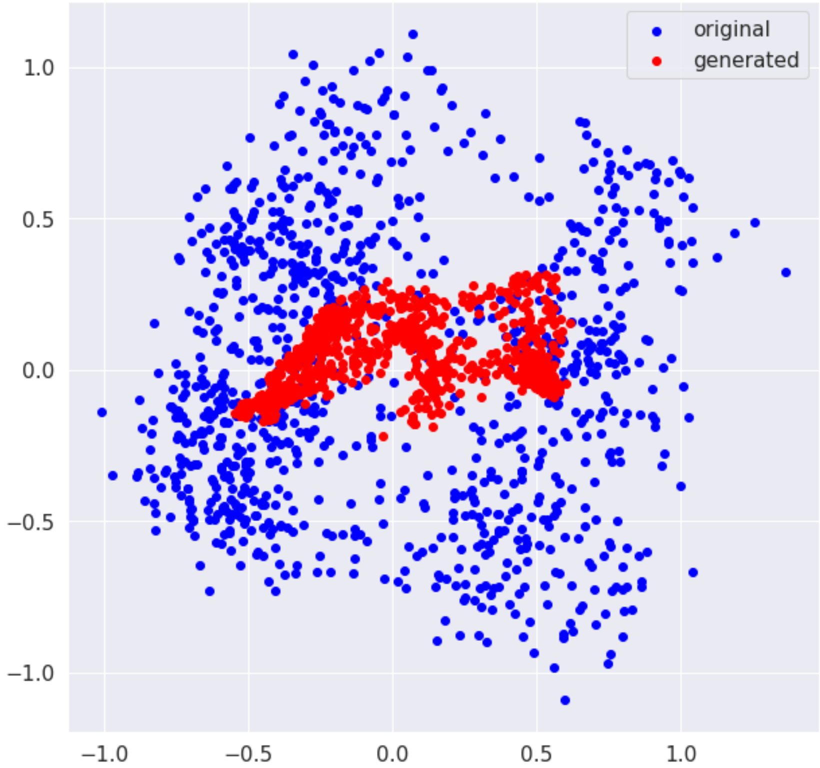
We used a PCA visualization of the real data and the sampled data, on order to see the distribution of the generated 1,000 samples. The twisted GAN model suffers from mode collapse for both datasets:

- Diabetes dataset:



The generated samples are centered on the middle area of the PCA, and not on the mass of the real data.

- German Credit dataset:



The generated samples are centered on the mass of the real data. But still, they don't represent the full distribution of the original data.

SUMMARY & IMPROVING THE MODEL

In this assignment, we experimented with creating two generative networks- a standard GAN, and a modified architecture. In the second part of the assignment, we stumbled upon mode collapse, as the generator produces a small set of different output types of the original data. It might happen because the generator found a type of data that is easily able to fool the discriminator with- and kept generating similar samples. To overcome this issue, we could try and perform several things:

- reward the generator for sample diversity, by rewarding the generator for generating data with distribution that is similar to the original data's distribution.
- use a better architecture to avoid mode collapse.