Anomalous Rhythm Detection

Nguyen Anh Tuan
AI202216701

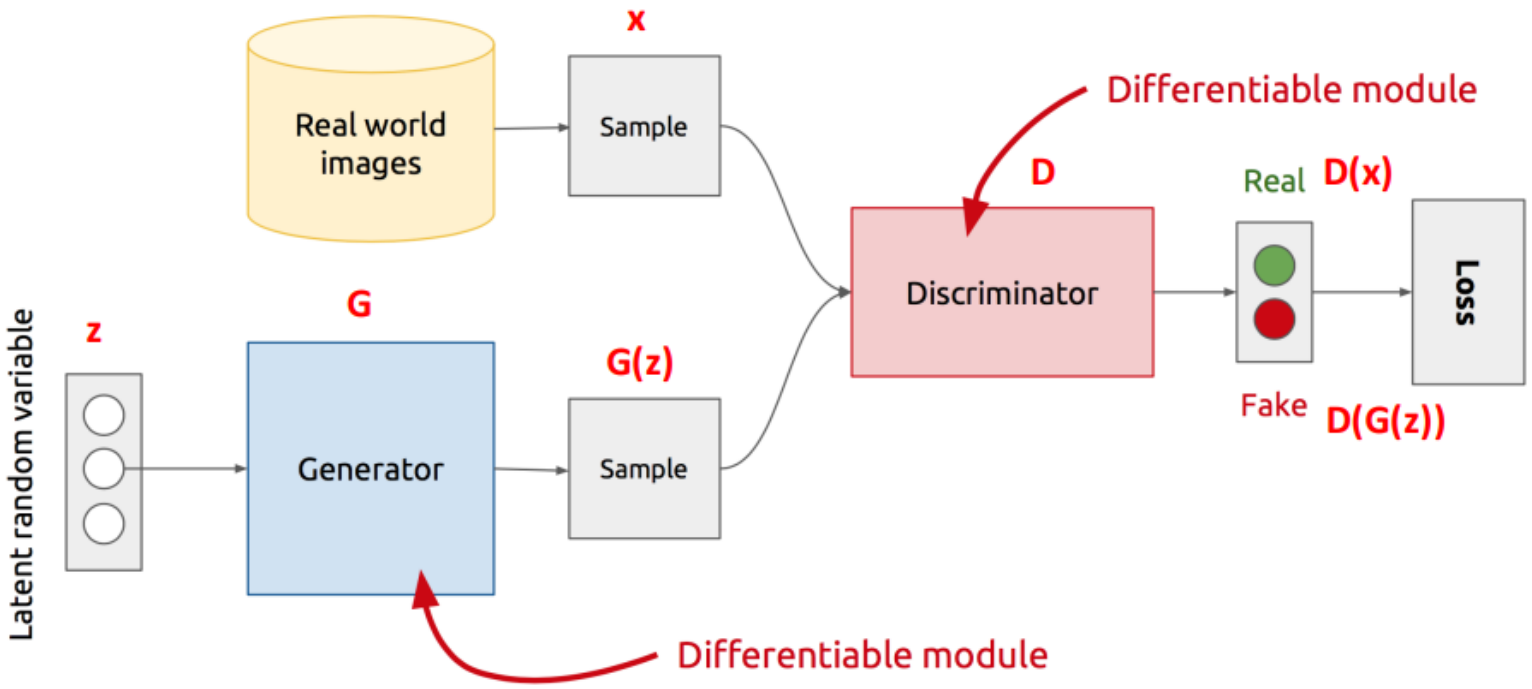
Introduction

Anomaly Detection

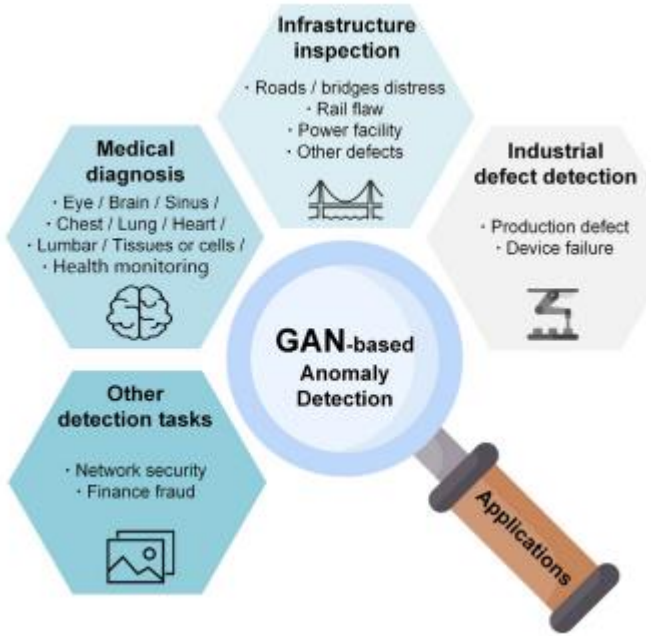
- Anomaly Detection
  - Detect outliers different from normal
- Various application field
  - Manufacturing industry, medical, CCTV,…
- Different ways to solve the problem
  - Supervised, unsupervised, semi-supervised,…

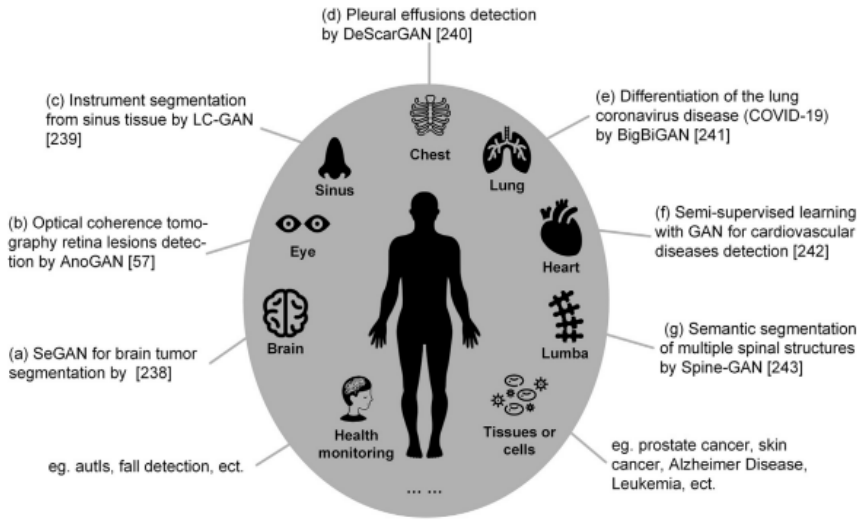
Generative adversarial network

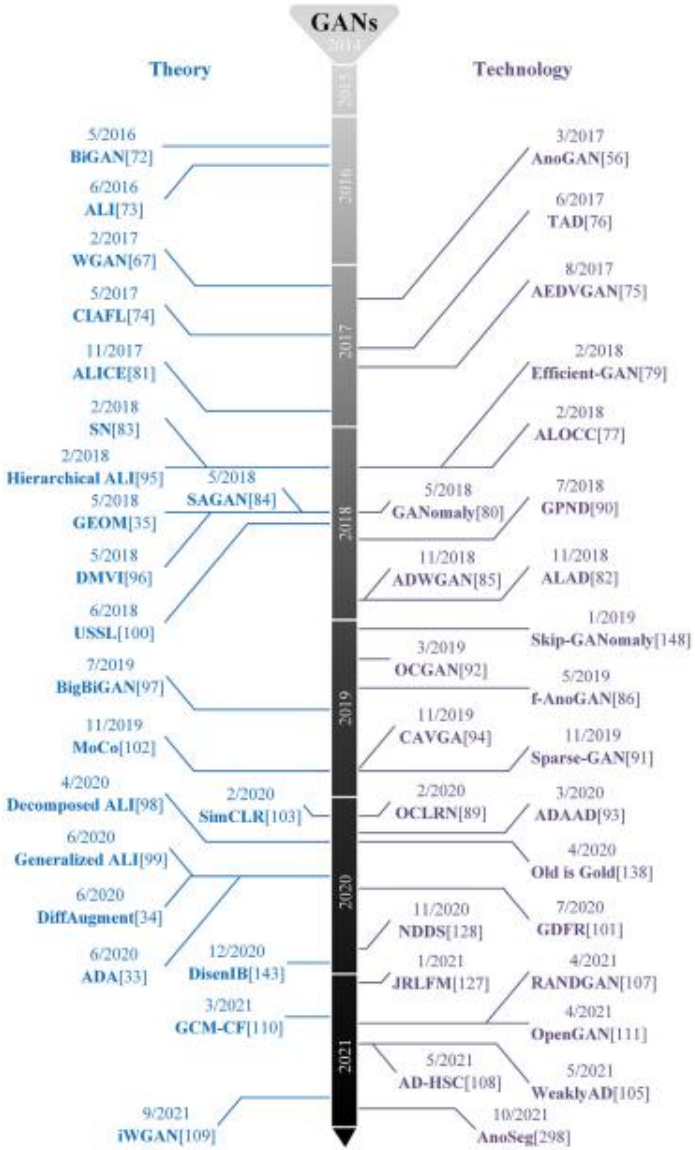
- A generative adversarial network (GAN) is a class of machine learning frameworks designed by Ian Goodfellow and his colleagues in 2014.
- The idea behind GAN: train two different networks
  - The generator tries to produce realistic-looking samples
  - The discriminator tries to figure out whether a sample is real or from generator



GAN for anomaly detection







BeatGAN: Anomalous Rhythm Detection using Adversarially Generated Time Series

Bin Zhou, Shenghua Liu, Bryan Hooi, Xueqi Cheng, Jing Ye

Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)

Abstract

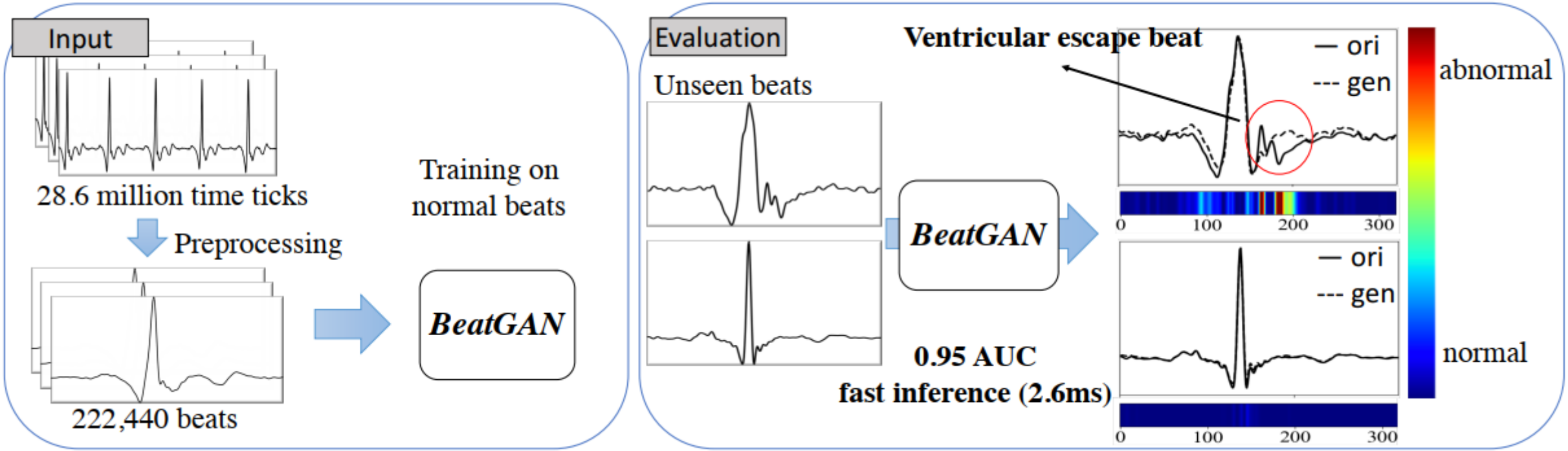
- With the rapid growth in availability of medical sensor data such as ECG (electrocardiogram), blood pressure, …, anomaly detection in medical time series has become an increasingly important topic of research.

- The authors propose BeatGAN, an unsupervised anomaly detection algorithm for time series data. It can detect anomalous heartbeats from ECG efficient way.

- Besides, BeatGAN outputs explainable results to pinpoint the anomalous time ticks of an input beat, by comparing them to adversarially generated beats.

- Experiments show that BeatGAN accurately and efficiently detects anomalous beats in ECG time series, and routes doctors' attention to anomalous time ticks

BeatGAN: General Framework



BeatGAN:Network structure

- An encoder $G_E(x)$, a decoder $G_D(z)$ and discriminator D
- Reconstruction loss with adversarial regularization:

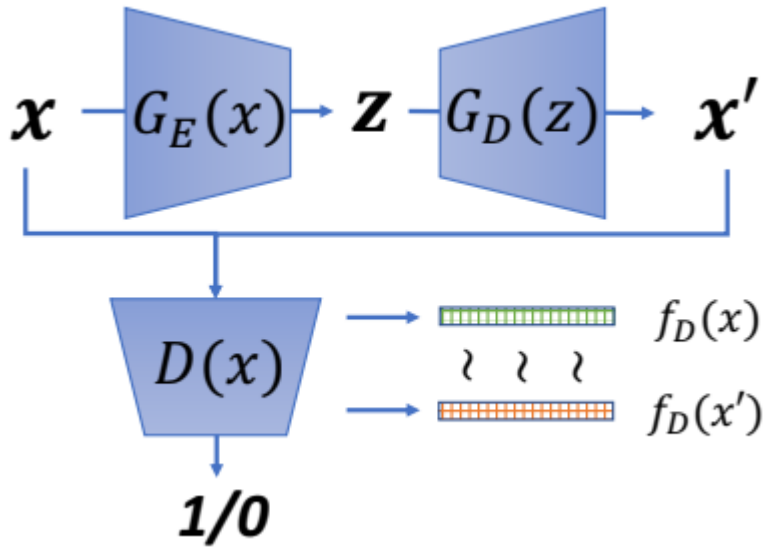
$$L_G = ||x - x'||_2 + \lambda ||f_D(x) - f_D(x')||_2$$

- Discrimination loss:

$$L_D = \frac{1}{N}\sum_{i}^{N}[\log D(x_i) + \log(1 - D(x_i'))]$$

- Anomalousness Score:

$$A(x) = \left|\left|x - x'\right|\right|_2$$



Algorithm 1 Training algorithm

1: $\theta_G, \theta_D \leftarrow$ initialize network parameters
2: **for** number of training iterations **do**
3: Sample $\{x_1, ..., x_m\} \sim$ a batch from the normal data
4: Generate $\{x_1', ..., x_m'\}$ by G_E and G_D
5: Compute L_D by Eq (7)
6: $\theta_D \longleftarrow \theta_D + \alpha \nabla_{\theta_D}(L_D)$ // ∇ is the gradient
7: Compute L_G by Eq (6)
8: $\theta_G \longleftarrow \theta_G + \alpha \nabla_{\theta_G}(L_G)$
9: **end for**

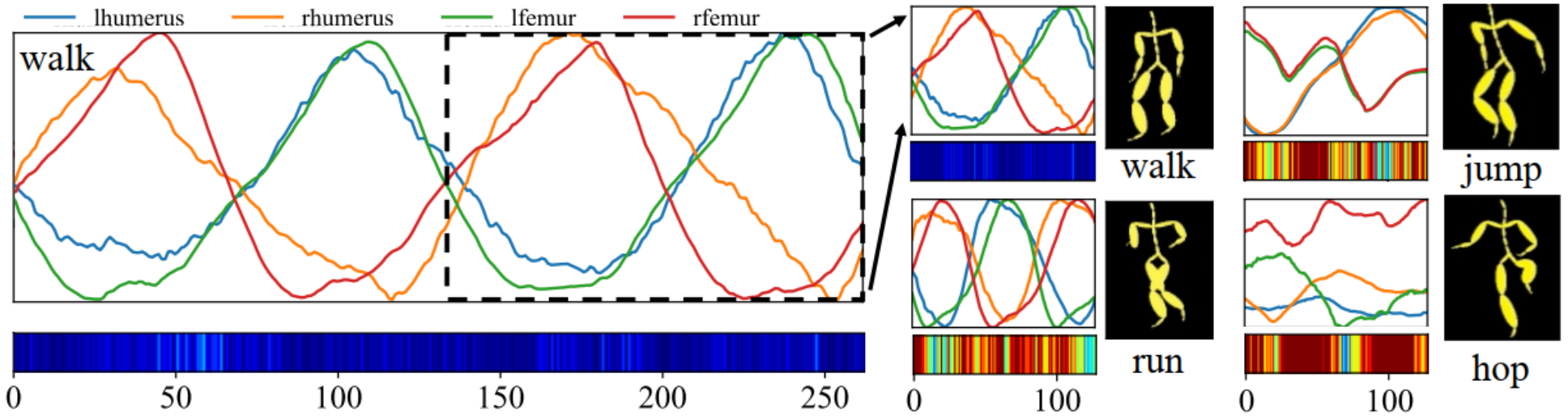
Data Augmentation using Time Warping

- For each training beat, the authors sample uniformly at random a small number *k* of time ticks to “slow down” or “speed up”.
- For each time tick to “speed up”, the authors delete the data value at that time tick.
- For each time tick to “slow down”, the authors insert a new data value just before that time tick, whose value is set to the average of the data values at the 2 adjacent time ticks.

Experiment: Dataset

- **MIT-BIH ECG dataset**: The MIT-BIH arrhythmia dataset contains 48 ECG (Electrocardiogram) records from test subjects from Beth Israel Hospital (https://physionet.org/content/mitdb/1.0.0/). The dataset has 28.6 million time ticks and 97568 beats.

- **CMU Motion Capture dataset:** The dataset contains motion-captured subjects performing different motions (walking, jogging, running,…). The authors choose 4 dimensions from different sensors (left-right arms and legs). The dataset has 16 walking records of 6616, 10 jogging records of 1608 and 1 jumping record of 2085 ticks. Totally, there are 10309 time ticks (http://mocap.cs.cmu.edu/)



Experiment: Setup (1)

- Setup for training ECG Dataset:
  - Choosing 320-time ticks as the window size for a beat.
  - Set the dimension size of latent space as 50
  - $\lambda = 1$ for regularization, $k = 16$ for data augmentation.
  - G_E's structure is a mirror version of G_D
  - D has the same architectural details of G_E
  - Optimizer: Adam with learning rate 0.0001, momentums 0.5 and 0.999
  - 5-fold cross-validation

```
class Decoder(nn.Module):
    def __init__(self, ngpu,opt):
        super(Decoder, self).__init__()
        self.ngpu = ngpu
        self.main=nn.Sequential(
            # input is Z, going into a convolution
            nn.ConvTranspose1d(opt.nz,opt.ngf*16,10,1,0,bias=False),
            nn.BatchNorm1d(opt.ngf*16),
            nn.ReLU(True),
            # state size. (ngf*16) x10
            nn.ConvTranspose1d(opt.ngf * 16, opt.ngf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ngf * 8),
            nn.ReLU(True),
            # state size. (ngf*8) x 20
            nn.ConvTranspose1d(opt.ngf * 8, opt.ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ngf * 4),
            nn.ReLU(True),
            # state size. (ngf*2) x 40
            nn.ConvTranspose1d(opt.ngf * 4, opt.ngf*2, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ngf*2),
            nn.ReLU(True),
            # state size. (ngf) x 80
            nn.ConvTranspose1d(opt.ngf * 2, opt.ngf , 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ngf ),
            nn.ReLU(True),
            # state size. (ngf) x 160
            nn.ConvTranspose1d(opt.ngf , opt.nc, 4, 2, 1, bias=False),
            nn.Tanh()
            # state size. (nc) x 320


        )
```

Experiment: Setup (2)

- Setup for training Motion Capture Dataset:
  - Walking is considered as our normal class.
  - Choosing 64-time ticks as the window size for a beat (4-dimenion).
  - Set the dimension size of latent space as 10
  - $\lambda = 0.01$ for regularization, $k = 16$ for dat augmentation.
  - Concatenate $x \in R^{4\times64}$ as a 256-dimenstional vector as input

```
self.encoder = nn.Sequential(
    # input is (nc) x 64
    nn.Linear(nc * 64, 256),
    nn.Tanh(),
    nn.Linear(256, 128),
    nn.Tanh(),
    nn.Linear(128, 32),
    nn.Tanh(),
    nn.Linear(32, 10),

)
self.decoder = nn.Sequential(
    nn.Linear(10, 32),
    nn.Tanh(),
    nn.Linear(32, 128),
    nn.Tanh(),
    nn.Linear(128, 256),
    nn.Tanh(),
    nn.Linear(256, nc * 64),
    nn.Tanh(),
)
```

```
class Discriminator(nn.Module):
    def __init__(self, nc):
        super(Discriminator, self).__init__()

        self.features = nn.Sequential(
            # input is (nc) x 64
            nn.Linear(nc * 64, 256),
            nn.Tanh(),
            nn.Linear(256, 128),
            nn.Tanh(),
            nn.Linear(128, 32),
            nn.Tanh(),

        )

        self.classifier=nn.Sequential(
            nn.Linear(32, 1),
            nn.Sigmoid()
        )
```

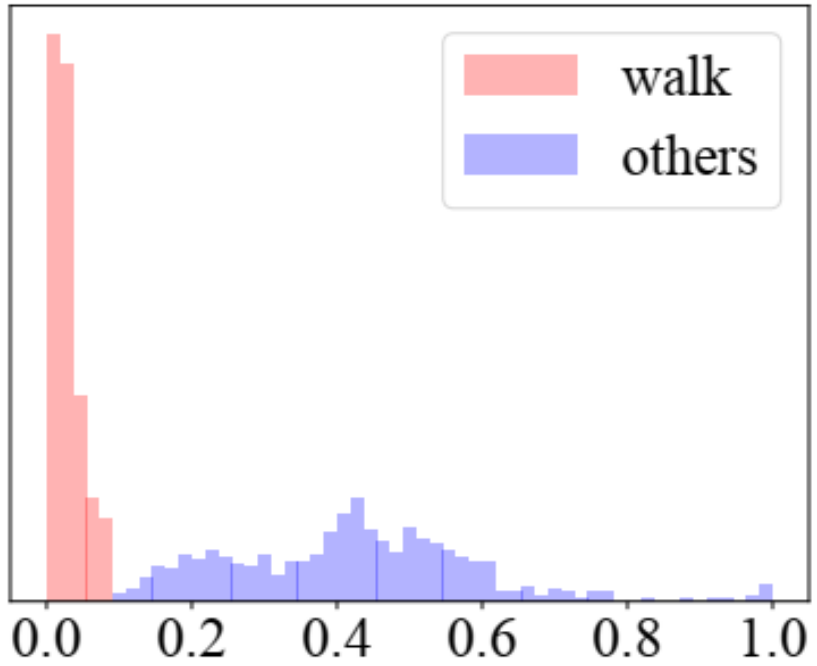
Result: Accuracy (ECG dataset)

- BeatGAN performs the best for anomalous heartbeat detection in ECG data.

Method	AUC	AP
PCA	0.8164 ± 0.0037	0.6522 ± 0.0061
OCSVM	0.7917 ± 0.0018	0.7588 ± 0.0027
AE	0.8944 ± 0.0128	0.8415 ± 0.0163
VAE	0.8316 ± 0.0025	0.7882 ± 0.0024
AnoGAN	0.8642 ± 0.0100	0.8035 ± 0.0069
Ganomaly	0.9083 ± 0.0122	0.8701 ± 0.0141
BeatGAN	0.9447 ± 0.0053	0.9108 ± 0.0049
BeatGAN$_{aug}$	$\mathbf{0.9475 \pm 0.0037}$	$\mathbf{0.9143 \pm 0.0047}$
BeatGAN$_{aug}^{0.1\%}$	0.9425 ± 0.0022	0.8973 ± 0.0042

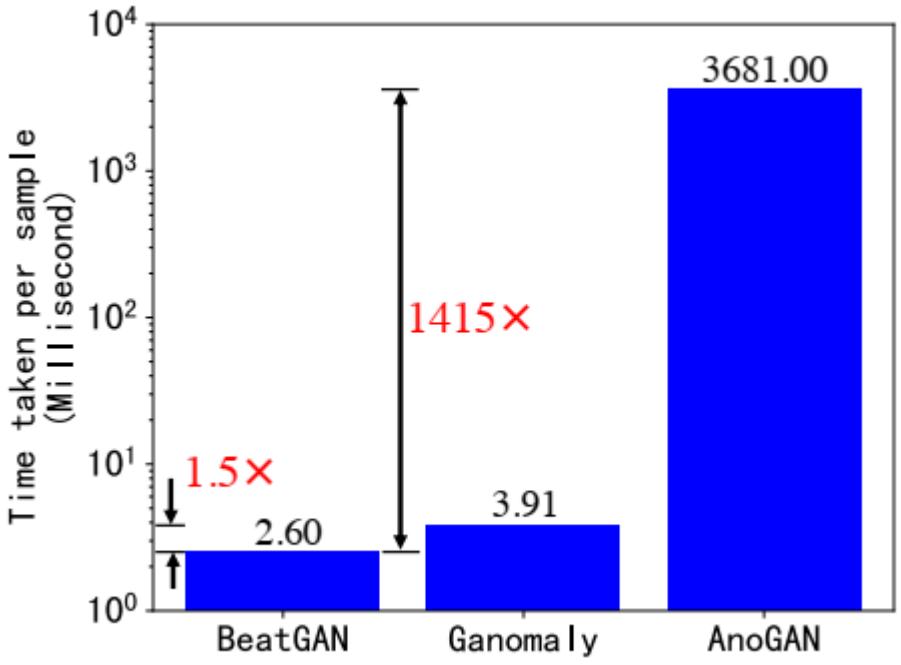
Result: Accuracy (Motion Capture Dataset)

- BeatGAN can perfectly discriminate between unusual motions (jogging/jumping) and usual motions (walking)
- AUC and AP metrics both achieve 1.0



Result: Efficiency

- Experiment uses Tesla K80 GPU
- BeatGAN is fast for inference
- Official source: https://github.com/hi-bingo/BeatGAN



References:

- [1]: Bin Zhou, Shenghua Liu, Bryan Hooi, Xueqi Cheng, and Jing Ye. ***BeatGAN: Anomalous rhythm detection using adversarially generated time series***. *In IJCAI, pages 4433–4439, 2019*
- [2]: X. Xia, X. Pan, N. Li, X. He, L. Ma, X. Zhang, and N. Ding, ***"Gan-based anomaly detection: A review,"*** *Neurocomputing, 2022*
- [3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. ***Generative adversarial nets***. *In NIPS, 2014*

Thank you!