Pressure Sensor Data Modeling with Recurrent Conditional Generative Adversarial Networks

1st B Sirisha
Department of Computer Science and Engineering
Maturi Venkata Subba Rao (MVSR) Engineering College
sirishavamsi@gmail.com

2nd B Sandhya
Department of Computer Science and Engineering
Maturi Venkata Subba Rao (MVSR) Engineering College
sandhyab16@gmail.com

***Abstract*—Several critical systems in an autonomous car strongly rely on pressure sensors to track and measure critical specifications, which has winded up to be the crucial aspect in making secure roadways, enhancing the driving experience, and minimizing pollution. Autonomous car driving experience would be completely different without all types of pressure sensors employed in the contemporary car, these sensors aid to handle braking systems to auto electrical windows, exhaustive emissions to the power steering. Monitoring the health of these pressure sensors is done by physically collecting the time series data at different scenarios. This approach is expensive and time-consuming. As a result, autonomous vehicle corporations are focusing on virtual authentication where synthetic data is generated for several scenarios. RCGAN is exploited in this work, for generating time series synthetic sensor data. This algorithm employs RNN, which trains the network model on supplementary information. This change permits RCGAN to learn and create sensible, time-series sensor data.**

***Index Terms*—Generative Adversarial Networks (GAN), Recurrent Neural Networks(RNN) , Recurrent Conditional Generative Adversarial Networks (RC-GAN), RGAN, Sensor Modeling, Time-series**

I. Introduction

In recent years the popularity of machine and deep learning algorithms is increasing exponentially in the field of autonomous vehicles. These learning frameworks have several applications in automotive automobile industries such as ADAS- advanced driver-assistance systems (ADAS), automated traffic jam pilot, automatic emergency braking [1]. The main challenge in these domains is to extract meaningful information from the processed data, which learning methods like DeepNets outdo. A deep learning algorithm requires machine learning frameworks that make use of deep neural networks with several layers (hidden) for resolving prediction and classification problems. [2]. All these learning algorithms need huge data for training the model. It is observed that for predicting autonomous car safety with a certain degree of confidence, several hundred kilometers have to be driven. This is an expensive and time-consuming task that is practically not feasible. Thus software automotive vehicle companies rely on virtual vehicle verification, which in turn relies on test models(test bench) of the actual environment and sensors. The sensor data required for training the model is very complex, noisy, and secured. Lack of gold standard datasets, data accessibility resists the growth of learning models specific to the task. Sharing physical sensor data in the automotive vehicle domain is extremely hard, as it comes from real-time applications, and is certainly extremely sensitive and legally protected by the manufacturer. One way to resolve this data issue is to generative model to create sufficiently realistic real-valued synthetic time series data. The GANs have demonstrated optimistic outcomes in the reproduction of time series data [3][4]. GAN is a popular generative model which is focused on reproducing synthetic information that is very much identical to the actual information[5]. Time series data are a set of data points we observe at every succeeding point in time, typically looking at data as evolving over time. This time series synthetic data could be cast-off to be reproducible to train and build a machine learning model. Conventional models like state-based, Hidden Markov models were once used for generating time series data [6][7]. Contemporary models like GANs are neural networks based. GANs are victorious in learning two-dimensional image distribution, and hence very popular in generating image data. Applications involving image data like text-to-image analysis [8][9] image translation [10][11], image super-resolution [12][13] are trained to learn 2D intensity distribution. This has motivated researchers to use generative models to learn any spatial distribution. Time series data is generated by learning the sequential distribution of data values at every instant of time. O. Mogren [14] and C. Esteban, S. L. Hyland, and G. Rtsch[15] have designed a network structure for generating real-valued continuous time-series data. The principal purpose of this work is :

- To study the aptness/suitability of RC GAN in generating synthetic time series physical sensor data (pressure sensor).
- Demonstrate the robustness of Recurrent Conditional GAN model in a Hydraulic system sensor dataset and exhibit quantitatively and visually that they can effectively generate multi-dimensional representative time-series synthetic data.

II. Related Work

Time Series(TS) Sensor Data Generation:In this research paper we are involved /concerned in developing a **generative model (GM)**, which is capable of generating time series of random length. In addition, the model is necessarily adaptable in order to accommodate TS with a considerable degree of

978-93-80544-44-1/22/$31.00 ©2022 IEEE

interference(noise). The prime characteristic of the GM is the output data, generated by GM should not be predetermined, i.e. we should not generate similar output for two independent runs for the same input data sequence. The data extracted from any physical sensor is affected by extrinsic and intrinsic interference receding from the neighborhood. A predetermined data reproduction model is inappropriate to generate the noise feature of the time series data. The main design objective of the GM is the signal trend generated at the output should follow the real-time series trend but it should yield very unique time-series data for each individual run.

The GM formulated in this paper is termed recurrent conditional (RC) generative adversarial network. This model tracks the original GAN architecture of Prof I. J. Goodfellow (2014) [5]. A good number of modifications have been applied to design the GM-generative model. Original GANs are deep neural networks designed to create new data instances with input P(n) training data distribution. This framework has two major neural network models, a Generator (G) and a Discriminator (D) network model as shown in Figure 1. Both the network models have very distinct and differing objectives.

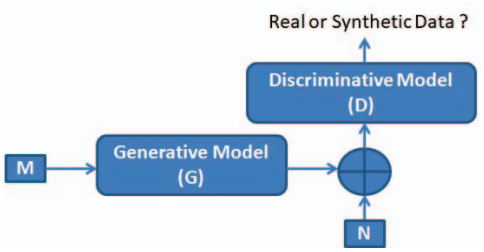

Fig. 1. A schematic view of the Generative Adversarial Networks (GAN) framework.

The D network model is fed with real-time series data as well as synthetic time-series data generated by the G network model. The goal of the D network model is to determine the probability of the Time series signal sample in the real-time signal or generated synthetic signal from the G network model. The G model is well trained to acquire the data point distribution by maximizing the D network model error. Back-propagation is applied to the G and D network model; this improves the D model probability to tell the data sample is real or synthetic. This will also improve the G network model by synthesizing realistic time series data samples. It is observed from the figure that the G network model takes randomly sampled vector M as input and tries to output a sample that is adjudged by the D network model. D model takes one of two signals i.e. real(n) signal or synthetic G(M) is input at the given time sample. Output with the D network model amounts to the probability that the part of the test time-series signal is real or synthetic. GANs are trained to resolve the optimization problem, mathematically expressed as:

$$Min_G Max_D \Xi_{n\sim p_n}[LogD(n)]+\Xi_{m\sim p_m}[Log(1-D(G(m)))] \quad (1)$$

Where n is the original data from Pr distribution and m is the instance obtained in previous distribution p(m), which is adapted from N(0,1). In other words, the loss function should be maximized for D network model & minimized for the G network model. The network is trained in two stages, D network model is trained in 1st stage followed by the G network model (2nd Stage). Samples from original time series data n and from dormant space m are obtained, where n is fed as input to the G network model. Both n and G(m) are provided via D, to yield D(n) and D(G(m)). The loss in the equation below and gradient are computed succeeded by upgrading the D weights.

$$\Xi_{n\sim p_n}[LogD(n)] + \Xi_{m\sim p_m}[Log(1 - D(G(m)) \quad (2)$$

When the G network model is trained time-series samples from the dormant space m are obtained and given as input over the G model, out put of this is supplied as input to the D network model. The signal obtained from the D network model is used as a loss function in the below equation. The weights of G are upgraded and gradients computed.

$$\Xi_{m\sim p_m}[Log(1 - D(G(m)) \quad (3)$$

III. Time Series Sensor Data Generation

RCGAN follows the framework of the original GAN developed by I. J. Goodfellow, D. Warde-Farley, J. Pouget-Abadie, Y. Bengio, M. Mirza, S. Ozair, B. Xu and A. Courville [5]. Following modifications are done to the Original GAN explained in the above section.

- Both G and D network models are RNNs, not convolutional nets or multilayer perceptrons. Both G and D network models are RNNs are based on LSTM.
- Need to incorporate conditional inputs to both network models obtain knowledgeable predictions for input. The conditional input to the network modules is the continuous time-series signal of the pressure sensor.

.

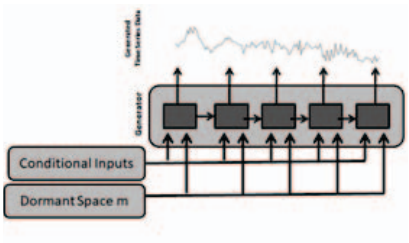

Fig. 2. A schematic architecture of the G and D model networks.

Figure 2 and Figure 3 show the schematic architecture of the G and D model networks. RCGAN generator takes a

dormant space vector m as one of the inputs and conditional input r from known distribution p(r). In addition, we provide discriminator either synthetic or real-time series along with the conditional input, this network provides a probability for every interval of time irrespective of input signal nature. The outcome of this network is used for computing loss function. The RCGAN generator network(G_{RC}) composed of noise and context network(See Figure-2). The G network model accepts an instance, m, out of dormant space that progressed to a single RNN layer, for time being input based on condition-C_1 is passed to a multiple layered RNN at the t-time interval. The signal generated out of C_1 and RNNs is stacked horizontally and fed to an FCL. The outcome of this network is a signal which is a linear function(activation) of inputs from FCL.

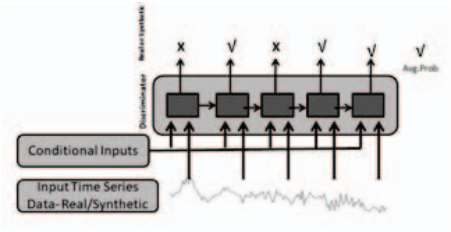

Fig. 3. A schematic architecture of the G and D model networks.

The discriminator (D_{RC}) (See Figure 3) accepts a sample n_t obtained from time series, synthetic or real data, with the conditional input at every time interval t. The inputs are stacked horizontally and passed to a multilayered Deep RNN. The output is passed into an FC layer along with conditional input. In the end, one output neuron is obtained for the prediction of each time interval.

The loss function-LF for generator and discriminator are expressed as:

$$L_G = LogD[G[m, C]; C] \tag{4}$$

$$L_D = LogD[n, C] + Log(1 - D[G[m, C]; C]) \tag{5}$$

c = condition-based input, n =data instance, m = random data instance, these loss functions are maximized in CGAN. The LF is adapted a little in time series data application. The new loss function is attained by computing arithmetic average(conditional loss) w.r.t time, where- n length of time series

$$L_G = \frac{1}{N}\Sigma_{i=1}^{N} Log(D[G[m_i, c_i]; c_i]) \tag{6}$$

$$L_D = \frac{1}{N}\Sigma_{i=1}^{N} Log(D[n_i, c_i]) - Log(D[G[m_i, c_i]; c_i]) \tag{7}$$

IV. Results and Discussion

The work is partitioned into three modules, 1.Module:1 Build and implement the RC GAN model, 2. Module:2 create an appropriate training method and 3. Module:3 evaluates the trained RCGAN model.

1) Dataset: The Sensor data set reports the health assessment of a hydraulic machine testbed constructed on multiple sensors. In this dataset 4 fault categories are overlaid with numerous sternness grades hindering discerning quantification. From this dataset, six pressure sensors are considered for experimentation. The raw pressure sensor data are organized as matrices with rows and columns demonstrating cycle and data values generated in a cycle. [16][17][18][19]. In this project, we consider only Pressure sensor(PS) bar values for data generation. The PS sensor produces 6000 samples in 60 sec. Figure 4 and 5 shows the original pressure signal for 60 seconds time duration and its sample instances.

- All six pressure sensors work at a sampling rate 100Hz.
- The characteristics of the dataset are Multivariate time series in nature.
- Number of data instances:2205
- The Characteristics of the sensor attribute: real
- Number of sensor attributes:43680
- Source:https://archive.ics.uci.edu/ml/datasets

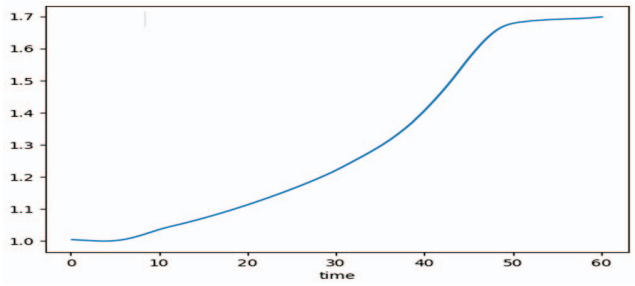

Fig. 4. Original Pressure Sensor Signal .

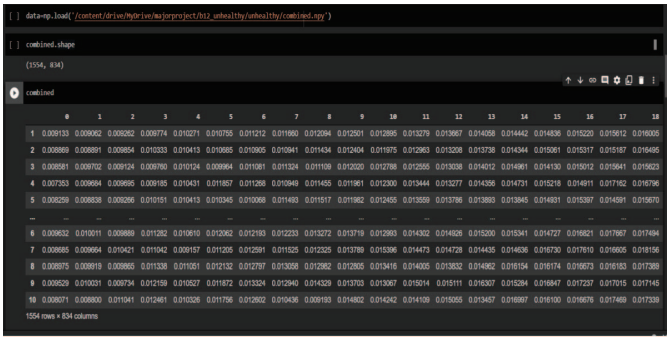

Fig. 5. Sample Pressure Sensor Data.

2) Implementation: The model is executed in the Python 3.6V IDE (Tensorflow-1.5). The network size used are Three(3) layers, 64 Long Short Term Memory network nodes

in the deep recurrent neural network (RNNs) subsequently followed by the (FCL) fully-connected layer. 64 Neurons for Generator and the discriminator. Leaky ReLU activation function on hidden convolution layers and Tanh activation function on output convolution layers. Figure 6 and Figure 8 shows the schematic of Generator and Discriminator network layers. Figure 7 and Figure 9 display the code to implement Generator and Discriminator network layers.

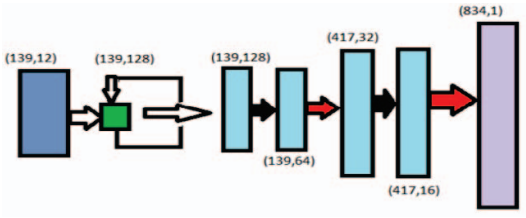

Fig. 6. A schematic of Generator Layers.

```
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Input(shape=(139, 12)))

    model.add(layers.Bidirectional(layers.LSTM(64, return_sequences=True)))

    model.add(layers.Conv1D(filters=128, kernel_size=16, strides=1, padding='same'))
    model.add(layers.LeakyReLU())

    model.add(layers.Conv1D(filters=64, kernel_size=16, strides=1, padding='same'))
    model.add(layers.LeakyReLU())

    model.add(layers.UpSampling1D(3))

    model.add(layers.Conv1D(filters=32, kernel_size=16, strides=1, padding='same'))
    model.add(layers.LeakyReLU())

    model.add(layers.Conv1D(filters=16, kernel_size=16, strides=1, padding='same'))
    model.add(layers.LeakyReLU())

    model.add(layers.UpSampling1D(2))

    model.add(layers.Conv1D(filters=1, kernel_size=16, strides=1, padding='same', activation='tanh'))

    model.add(layers.Permute((2, 1)))

    return model
```

Fig. 7. Code for Generator Network.

3) Evaluation Metrics:

- RMSE-Root Mean Squared Error: It is the standard deviation(SD) that occurs in forecasted data. def rmse (targets,forecast): return np.sqrt(np.mean((targets-forecast)**2)). In this function, we pass the actual pressure signal value and synthetically generated pressure signal value as parameters and we find Root Mean Square for all the values, and these values are appended into an array. Finally, we find Maximum value, Minimum Value, Mean Value from this array

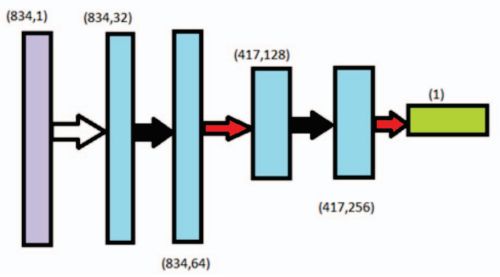

Fig. 8. A schematic of Discriminator Layers.

```
def make_discriminator_model():
    model = tf.keras.Sequential()

    model.add(layers.Input(shape=(1, 834)))
    model.add(layers.Permute((2, 1)))

    model.add(layers.Conv1D(filters=32, kernel_size=16, strides=1, padding='same'))
    model.add(layers.LeakyReLU())

    # model.add(layers.Dropout(0.4))

    model.add(layers.Conv1D(filters=64, kernel_size=16, strides=1, padding='same'))
    model.add(layers.LeakyReLU())

    model.add(layers.MaxPool1D(pool_size=2))

    model.add(layers.Conv1D(filters=128, kernel_size=16, strides=1, padding='same'))
    model.add(layers.LeakyReLU())

    model.add(layers.Dropout(0.4))

    model.add(layers.Conv1D(filters=256, kernel_size=16, strides=1, padding='same'))
    model.add(layers.LeakyReLU())

    model.add(layers.MaxPool1D(pool_size=2))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model
```

Fig. 9. Code for Discriminator Network.

- PRD Metrics : It is defined as Square root of the ratio of sum of squares of difference between actual output and predicted output to the sum of squares of actual output.

$$\sqrt{\frac{\Sigma_{i=1}^{inputshape.length}(X_i - Y_i^2)}{\Sigma_{i=1}^{inputshape.length}(X_i^2)}} * 100 \qquad (8)$$

where

1) X_i– Actual output
2) Y_i– Predicted output
3) Description: def prd(targets, predictions): s1 = np.sum((targets-predictions)**2) s2 = np.sum(targets**2) return np.sqrt(s1 / s2 * 100)

- Accuracy: It is defined as ratio of number of correct/true prediction by total no of predictions.

$$Accuray = \frac{NumberofTruePredictions}{TotalnumberofPredictions} \qquad (9)$$

- Loss : The generator network tries to minimize the following loss function while the discriminator tries to

maximize it

$$L_G = \frac{1}{N}\Sigma_{i=1}^{N} Log(D[G[m_i, c_i]; c_i]) \quad (10)$$

$$L_D = \frac{1}{N}\Sigma_{i=1}^{N} Log(D[n_i, c_i]) - Log(D[G[m_i, c_i]; c_i]) \quad (11)$$

In this function:

1) D(n)= Discriminator estimation of likelihood that original signal data instance (n) is true(real).
2) G(m)= The outcome of Generator(G)at interference noise(z).
3) E_x= Anticipated value over original signal data instance.
4) D(G(m))= Discriminator estimation of likelihood that synthetic signal data instance (n) is true(real).
5) E_m = Anticipated value over input signal data instance(random) to generator
6) The outcome is obtained from cross-entropy among the original and synthetic distributions. (See Figure 10)

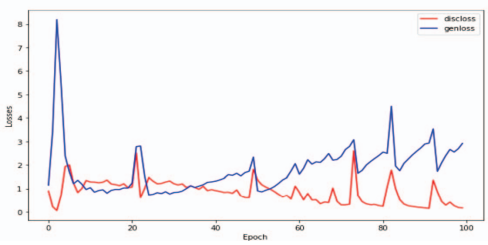

Fig. 10. Loss Metrics

- Test Case: Test the generator model using the testing data which is obtained from real data set. The generator model is tested against the test cases by making the generator model training status false. Figure 11 shows the Generator value against the given test case.

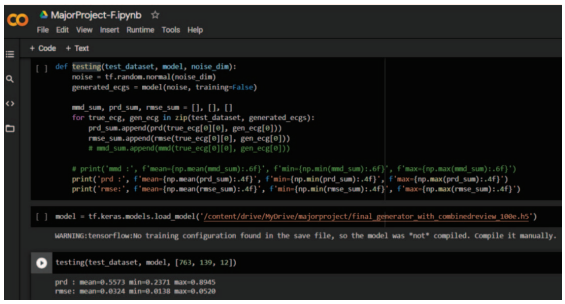

Fig. 11. Generator value against test cases

- Results: The Scatter plot of the original data distribution is given in the following figure (See Figure 12)

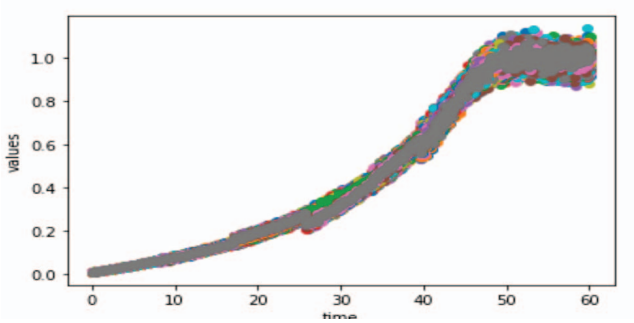

Fig. 12. Scatter Plot of Original Pressure Sensor Data

- The generator before training, the generated value is shown by following Figure 13.

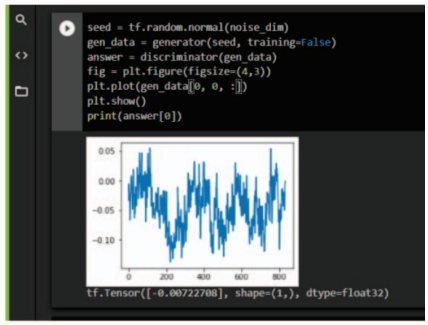

Fig. 13. Generated Data before Training

- The generator model has **generated 5000 samples of pressure sensor data.** A sample of generated value is shown in the Figure 14 below.

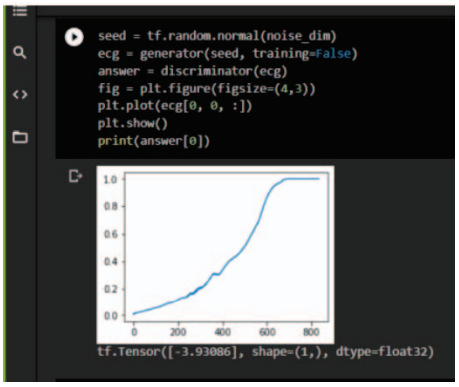

Fig. 14. Generated Synthetic data after training

TABLE I
MEAN, MINIMUM AND MAXIMUM VALUES OBTAINED FOR PRD AND RMSE

Metric Name	Mean	Minimum	Maximum
PRD	0. 5573	0. 2371	0.8945
RMSE	0. 0324	0. 0138	0.0520

V. CONCLUSION

In this paper RCGAN framework for modeling pressure sensor data has been developed. This framework satisfies the anticipated characteristics of the time series GAN i.e. framework can handle TS data of random length, the framework can tune noise level (Gaussian noise) to the precise time series dissemination. The pressure sensor dataset has two common trends of data sequences, opening at lower values and proceeding towards higher values and vice-versa. The trained framework acquires the two data tends displayed. The generator network has proved to work very well and produce a virtuous outcome because it possesses the proficiency of distribution learning.

VI. FUTURE WORK

The proposed Recurrent Conditional Generative Adversarial Networks for time series data exhibited in this paper will pursue to prosper as it has exhibited good results for producing synthetic sensor time series data. Firstly a novel initializing approach or adaptation of the intrinsic architecture need to be created which permit the proposed model to generate a highly sensible preliminary transient. Supplementary action need to be taken to make use of micro and mini batch training to make it possible to utilize mini-batch training for progressions of varied lengths. This training will help in learning stabilization.

REFERENCES

[1] A. L. et al., Deep learning in the automotive industry: Applications and tools, IEEE International Conference on Big Data, pp. 37593768, 2016.

[2] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016. http: //www.deeplearningbook.org.

[3] A. Graves, Generating sequences with recurrent neural networks, CoRR, vol. abs/1308.0850, 2013.

[4] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, Wavenet: A generative model for raw audio, CoRR, vol. abs/1609.03499, 2016.

[5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, Generative Adversarial Networks, ArXiv e-prints, June 2014.

[6] E. L. Zec, N. Mohammadiha, A. Schliep, Modelling Autonomous Driving Sensors Using Hidden Markov Models, under review, 2018.

[7] E. Karlsson, N. Mohammadiha, A Statistical GPS Error Model for Autonomous Driving, in Proc. IEEE Intelligent Vehicles (IV), June 2018.

[8] S. E. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, Generative adversarial text to image synthesis, CoRR, vol. abs/1605.05396, 2016.

[9] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. N. Metaxas, Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks, CoRR, vol. abs/1612.03242, 2016.

[10] J. Zhu, T. Park, P. Isola, and A. A. Efros, Unpaired image-to-image translation using cycle-consistent adversarial networks, CoRR, vol. abs/1703.10593, 2017.

[11] Y. Taigman, A. Polyak, and L. Wolf, Unsupervised cross-domain image generation, CoRR, vol. abs/1611.02200, 2016. 12. [13] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. P. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, Photo-realistic single image super-resolution using a generative adversarial network, CoRR, vol. abs/1609.04802, 2016.

[12] C. K. Snderby, J. Caballero, L. Theis, W. Shi, and F. Huszr, Amortised MAP inference for image super-resolution, CoRR, vol. abs/1610.04490, 2016.

[13] O. Mogren, C-RNN-GAN: continuous recurrent neural networks with adversarial training, CoRR, vol. abs/1611.09904, 2016.

[14] C. Esteban, S. L. Hyland, and G. Rtsch, Real-valued (medical) time series generation with recurrent conditional gans, CoRR, vol. abs/1706.02633, 2017.

[15] Nikolai Helwig, Eliseo Pignanelli, Andreas Schtze,Condition Monitoring of a Complex Hydraulic System Using Multivariate Statistics, in Proc. I2MTC-2015 - 2015 IEEE International Instrumentation and Measurement Technology Conference, paper PPS1-39, Pisa, Italy, May 11-14, 2015, doi: 10.1109/I2MTC.2015.7151267.

[16] N. Helwig, A. Schtze, Detecting and compensating sensor faults in a hydraulic condition monitoring system", in Proc. SENSOR 2015 - 17th International Conference on Sensors and Measurement Technology, oral presentation D8.1, Nuremberg, Germany, May 19-21, 2015, doi: 10.5162/sensor2015/D8.1.

[17] Tizian Schneider, Nikolai Helwig, Andreas Schtze, Automatic feature extraction and selection for classification of cyclical time series data, tm - Technisches Messen (2017), 84(3), 198206, doi: 10.1515/teme-2016-0072.

[18] Nikolai Helwig, Eliseo Pignanelli, Andreas Schtze, Condition Monitoring of a Complex Hydraulic System Using Multivariate Statistics, in Proc. I2MTC-2015 - 2015 IEEE International Instrumentation and Measurement Technology Conference, paper PPS1-39, Pisa, Italy, May 11-14, 2015, doi: 10.1109/I2MTC.2015.7151267.