ADAFLOW: DOMAIN-ADAPTIVE DENSITY ESTIMATOR WITH APPLICATION TO ANOMALY DETECTION AND UNPAIRED CROSS-DOMAIN TRANSLATION

Masataka Yamaguchi[1]*, Yuma Koizumi*[2]*, and Noboru Harada*[2]

[1]: NTT Communication Science Laboratories, Kanagawa, Japan
[2]: NTT Media Intelligence Laboratories, Tokyo, Japan

ABSTRACT

We tackle unsupervised anomaly detection (UAD), a problem of detecting data that significantly differ from normal data. UAD is typically solved by using density estimation. Recently, deep neural network (DNN)-based density estimators, such as Normalizing Flows, have been attracting attention. However, one of their drawbacks is the difficulty in adapting them to the change in the normal data's distribution. To address this difficulty, we propose *AdaFlow*, a new DNN-based density estimator that can be easily adapted to the change of the distribution. AdaFlow is a unified model of a Normalizing Flow and Adaptive Batch-Normalizations, a module that enables DNNs to adapt to new distributions. AdaFlow can be adapted to a new distribution by just conducting forward propagation once per sample; hence, it can be used on devices that have limited computational resources. We have confirmed the effectiveness of the proposed model through an anomaly detection in a sound task. We also propose a method of applying AdaFlow to the unpaired cross-domain translation problem, in which one has to train a cross-domain translation model with only unpaired samples. We have confirmed that our model can be used for the cross-domain translation problem through experiments on image datasets.

***Index Terms*—** Deep learning, normalizing flow, domain adaptation, anomaly detection, and cross-domain translation.

1. INTRODUCTION

Anomaly detection, also known as outlier detection, is a problem of detecting data that significantly differ from normal data [1–3]. Since such anomalies might indicate symptoms of mistakes or malicious activities, their prompt detection may prevent such problems. Therefore, anomaly detection has received much attention and been applied for various purposes.

In this paper, we specifically consider unsupervised anomaly detection (UAD), in which only normal data can be used for training anomaly detection models. UAD is typically solved by first training a normal model with normal data and then estimating the deviance of each testing sample with the trained model. In the anomaly detection field, many types of normal models have been investigated. In the early studies, a Gaussian distribution was used [4, 5], and recently, more flexible statistical models have been used such as a Gaussian mixture model (GMM) [6, 7]. More recently, deep neural network (DNN)-based methods have been investigated such as an Auto-Encoder (AE) [8, 9], a Variational Auto-Encoder (VAE) [10–12], and Generative Adversarial Networks (GAN) [13–15].

In the typical setting of UAD, one assumes that training and testing data are sampled from the same distribution. However, this assumption does not hold in certain practical scenarios. Let us consider the anomaly detection problem on facility equipments. Typically, such equipments have various operation patterns, and the environmental noise patterns around them may change due to certain factors such as seasons and the weather. In this case, the above assumption does not always hold; hence, simply applying existing normal models to such problems may significantly decrease the anomaly detection accuracy. A naïve method one can use to avoid this is to adapt normal models to a new distribution by conducting fine-tuning with newly-collected normal data. However, fine-tuning requires high memory and computational costs and cannot be easily conducted with devices installed in facility equipments that typically have only limited computational resources. Therefore, a more efficient adaptation method is needed.

To address this problem, we propose a new density estimator named *AdaFlow*, a unified model of Normalizing Flows (NFs) [16, 17], a powerful DNN-based density estimator, and the Adaptive Batch Normalization (AdaBN) [18], a module that enables DNNs to handle different domains' data. AdaBN alleviates the difference between domains by scaling and shifting each domain's input data so that each domain's mean and variance are zero and one, respectively. Since AdaBN can be adapted to a new domain by just adjusting its statistics with the domain's data, the adaptation step of AdaFlow can be done by just conducting forward-propagation only once per sample. Therefore, AdaFlow can be used on devices that have limited computational resources.

We also propose a method of applying AdaFlow to the unpaired cross-domain translation problem, in which one has to train a cross-domain translation model with only unpaired data. We show the effectiveness of using AdaFlow for this problem through cross-domain translation experiments on image datasets.

2. RELATED WORK

2.1. Unsupervised anomaly detection

In UAD, the deviation between a normal model and observation is computed; the deviation is often called the "*anomaly score*". One way of computing anomaly scores is a density estimation-based approach. This approach first trains a density estimator $q_\theta(\cdot)$, such as a Gaussian distribution function, with normal data, and then computes the negative log-likelihood of each testing data $\boldsymbol{x} \in \mathbb{R}^D$ with $q_\theta(\cdot)$. In this approach, its negative log-likelihood is used as its anomaly score $\mathcal{A}(\boldsymbol{x}, \theta)$, i.e.,

$$\mathcal{A}(\boldsymbol{x}, \theta) = -\ln q_\theta(\boldsymbol{x}). \tag{1}$$

Then, $\boldsymbol{x}$ is determined to be anomalous when the anomaly score exceeds a pre-defined threshold ϕ:

$$\mathcal{H}(\boldsymbol{x}, \phi) = \begin{cases} 0 \text{ (Normal)} & \mathcal{A}(\boldsymbol{x}, \theta) < \phi \\ 1 \text{ (Anomaly)} & \mathcal{A}(\boldsymbol{x}, \theta) \geq \phi \end{cases}. \tag{2}$$

Recently, deep learning has also been investigated for defining normal models for UAD. Several studies on deep-learning-based UAD employed an AE [8, 9] (or a VAE [11, 12]). The AE-based anomaly detection framework defines the anomaly score as follows:

$$\mathcal{A}(\boldsymbol{x}, \theta) = \|\boldsymbol{x} - \mathcal{D}_{\theta_D}(\mathcal{E}_{\theta_E}(\boldsymbol{x}))\|^2, \tag{3}$$

where $\|\cdot\|$ denotes the L_2 norm, $\mathcal{E}$ and $\mathcal{D}$ are the encoder and decoder of the AE, and θ_E and θ_D are its parameters, namely $\theta = \{\theta_E, \theta_D\}$. Then, θ is trained to minimize the anomaly scores of normal data as follows:

$$\theta \leftarrow \arg\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \mathcal{A}(\boldsymbol{x}_n, \theta), \tag{4}$$

where $\boldsymbol{x}_n$ is the n-th training sample and N is the number of training samples.

Although it has been empirically shown that anomaly detection can be addressed by AE-based anomaly detection, one of its drawbacks is that there is no guarantee that minimizing Eq. (4) encourages anomaly scores of normal data to be less than those of anomaly data, because anomaly scores of anomaly data are not considered in Eq. (4). In contrast, in the density estimation-based approach, minimizing NLLs of normal data encourages to maximize NLLs of the other data, including anomal data, since the integral value of the likelihood in the input space is always 1. Therefore, instead of using the AE-based anomaly detection approach, we adopt the density estimation-based approach. Specifically, in this paper, we adopt a Normalizing Flow (NF), a DNN-based flexible density estimator. We explain its details in Section 3.

2.2. Domain adaptation on DNN-based density estimator

Although a DNN is a powerful tool for anomaly score computation, it may be problematic for practical use. One problem occurs when adjusting the normal model to a new domain. The distribution of normal data often varies due to aging of the target and/or change in environmental noise. Therefore, we need to adapt the normal model to such fluctuations. Let us formulate this problem. Suppose that we have a normal model q_θ trained on $K \geq 1$ dataset(s) collected in individual domains. When the distribution changes, we need to adapt q_θ to the new domain ($(K+1)$-th domain) to obtain a new normal model q'_θ. This problem can be regarded as an analogy of *domain adaptation* [19]. Although several domain adaptation methods have been investigated [20–22], most require iterative optimization and huge memory, and such methods cannot be easily used with devices installed in most practical conditions, which typically have limited computational resources. Therefore, in terms of the computational cost and required memory, a more efficient adaptation method is needed.

3. PROPOSED METHOD

3.1. Normalizing Flow

We adopt a Normalizing Flow (NF) as a density estimator. NF represents a probabilistic density by transforming a base probabilistic density function $q_0(\boldsymbol{z}^{(0)})$ with a series of M invertible projections $\{f_m\}_{m=1}^M$ with each parameter $\{\theta_m\}_{m=1}^M$. In NF, $\boldsymbol{x}$ is regarded as a transformed variable with $\{f_m\}_{m=1}^M$ as follows:

$$\boldsymbol{x} = \boldsymbol{z}_M = f_{M,\theta_M} \circ \cdots \circ f_{1,\theta_1}(\boldsymbol{z}^{(0)}); \tag{5}$$

thus, $\boldsymbol{z}^{(0)}$ can be obtained by the inverse transform of (5). Following prior works [16, 17], we employ a Gaussian distribution

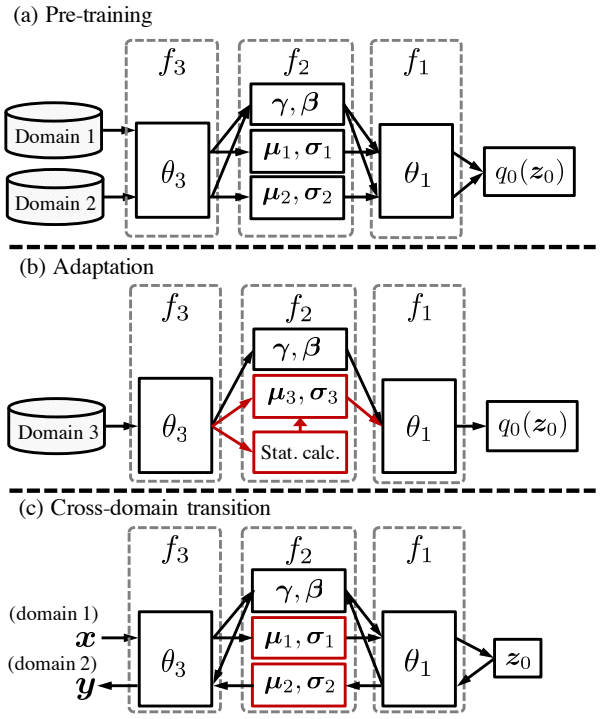

Fig. 1. Simplified concept of AdaFlow; (1) pre-training, (2) adaptation, and (3) cross-domain transition. In pre-training, all parameters $\{\theta_m\}_{m=1}^3$ are trained with $K = 2$ domain datasets. For adaptation, BN statistics of second projection f_2 are computed from third domain dataset. For cross-domain transition, BN statistics of input domain is used for inverse projection, and that of target domain is used for forward projection.

$\mathcal{N}(\boldsymbol{z}^{(0)}; \boldsymbol{0}, \boldsymbol{I})$ for $q_0(\boldsymbol{z}^{(0)})$. Then, the likelihood of the given sample $\boldsymbol{x}$ is obtained by repeatedly applying the rule for *change of variables* as follows:

$$q_\theta(\boldsymbol{x}) = q_0(\boldsymbol{z}^{(0)}) \prod_{m=1}^{M} \left| \frac{\partial f_{m,\theta_m}}{\partial \boldsymbol{z}^{(m-1)}} \right|^{-1}, \tag{6}$$

Thus, the anomaly score computed by NF can be expressed as

$$\mathcal{A}(\boldsymbol{x}, \theta) = -\ln q_0(\boldsymbol{z}^{(0)}) - \sum_{m=1}^{M} \ln \left| \frac{\partial f_{m,\theta_m}}{\partial \boldsymbol{z}^{(m-1)}} \right|^{-1}. \tag{7}$$

Parameters $\theta = \{\theta_m\}_{m=1}^M$ can be trained by minimizing the anomaly scores as follows:

$$\theta \leftarrow \arg\min_{\theta} \sum_{k=1}^{K} \frac{1}{N_k} \sum_{n=1}^{N_k} \mathcal{A}(\boldsymbol{x}_{n,k}, \theta), \tag{8}$$

where $\boldsymbol{x}_{n,k}$ and N_k are the n-th training sample and the number of training samples of the k-th dataset, respectively.

3.2. AdaFlow

We consider domain adaptation for NF. A naïve method of adapting NF to the $(K+1)$-th dataset is to fine-tune all $\{\theta_m\}_{m=1}^M$ with that dataset. However, fine-tuning requires high memory and computational costs and cannot be easily conducted with devices installed

in facility equipments that typically have only limited computational resources. Therefore, a more efficient adaptation method is needed.

To address this problem, we propose AdaFlow, a Normalizing Flow-based density estimator that utilizes Adaptive Batch Normalizations (AdaBNs). An AdaBN converts data as follows:

$$f^{-1}_{m,\theta_m}(\boldsymbol{z}) = \mathrm{diag}(\boldsymbol{\gamma})\mathrm{diag}(\boldsymbol{\sigma}_k)^{-\frac{1}{2}}(\boldsymbol{z} - \boldsymbol{\mu}_k) + \boldsymbol{\beta}, \tag{9}$$

where $\boldsymbol{\mu}_k \in \mathbb{R}^D$ and $\boldsymbol{\sigma}_k \in \mathbb{R}^D$ are vectors of mean and variance computed with the data in the k-th domain, respectively, and $\boldsymbol{\gamma} \in \mathbb{R}^D$ and $\boldsymbol{\beta} \in \mathbb{R}^D$ are learnable parameters shared in all domains. The function $\mathrm{diag}(\boldsymbol{\lambda})$ denotes an operator that converts $\boldsymbol{\lambda}$ into a diagonal matrix of which (i,j)-th entry is $(\boldsymbol{\lambda})_i$ if $i = j$, otherwise 0. Note that $\boldsymbol{\mu}_k$ and $\boldsymbol{\sigma}_k$ are individually calculated for each domain, whereas same $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are used for all K domains. By training the whole projections in this manner, $\boldsymbol{\mu}_k$ and $\boldsymbol{\sigma}_k$ alleviate the difference up to the second-order moment for each domain in the hidden layers. In addition, adapting AdaFlow to the given $(K+1)$-th domain can be achieved by just computing AdaBNs' statistics $\boldsymbol{\mu}_{K+1}$ and $\boldsymbol{\sigma}_{K+1}$ with data sampled from that domain.

We summarize the overall procedure of pre-training and adapting AdaFlow as follows and in Fig 1: (i) pre-train AdaFlow projections with K datasets by (8), (ii) adapt the statistics of AdaBNs $\boldsymbol{\mu}'$ and $\boldsymbol{\sigma}'$ with the $(K+1)$-th dataset.

3.3. Examples of projection implementations

We next explain projections that can be used for implementing AdaFlow. If each projection is easy to invert and the determinant of its Jacobian is easy to compute, exact density estimation at each data point can be easily conducted. We introduce two projections that satisfy the above requirements.

Linear Transformation: Linear transformation can be used as a projection for NFs as follows:

$$f^{-1}_{m,\theta_m}(\boldsymbol{z}) = \boldsymbol{W}\boldsymbol{z} + \boldsymbol{b}, \tag{10}$$

where $\boldsymbol{W} \in \mathbb{R}^{D\times D}$ and $\boldsymbol{b} \in \mathbb{R}^D$ is a weight matrix and a bias vector, respectively. The determinant of the Jacobian of this projection is $|\boldsymbol{W}|^{-1} = 1/|\boldsymbol{W}|$. Since its computational complexity is $O(D^3)$, we reparametrize $\boldsymbol{W}$ as a LDU decomposition form $\boldsymbol{W} = \boldsymbol{L}\mathrm{diag}(\boldsymbol{d})\boldsymbol{U}$, where $\boldsymbol{L}$ and $\boldsymbol{U}$ is a lower and upper triangular matrix of which all diagonal elements are one, respectively, and $\boldsymbol{d} \in \mathbb{R}^D$. Since $|\boldsymbol{U}| = |\boldsymbol{L}| = 1$ and $|\mathrm{diag}(\boldsymbol{d})| = \prod_i^D (\boldsymbol{d})_i$, the computational complexity of the determinant of the Jacobian can be reduced to $O(D)$ by using this reparametrization form.

Leaky ReLU: A Leaky Rectified Linear Unit (Leaky ReLU) is a module used for DNNs, defined as follows:

$$f^{-1}_{m,\theta_m}(\boldsymbol{z}) = \max(\boldsymbol{z}, \alpha\boldsymbol{z}), \tag{11}$$

where $\alpha \in (0,1)$ is a hyper parameter, and $\max(\boldsymbol{\lambda}^{(1)}, \boldsymbol{\lambda}^{(2)})$ is an operator that outputs element-wise maximum of $\boldsymbol{\lambda}^{(1)}$ and $\boldsymbol{\lambda}^{(2)}$, respectively. Since Leaky ReLU is easy to invert and the determinant of its Jacobian is easy to compute, it can also be used as a projection for NFs. The determinant of its Jacobian is $\alpha^{-\tau}$, where τ is the number of elements that are less than 0.

4. EXPERIMENTS

4.1. Experimental Settings

4.1.1. Dataset

To verify the effectiveness of AdaFlow, we conducted experiments on an anomaly detection in sound (ADS) task. For the training and test datasets, we constructed a toy-car-running sound dataset in a simulated room of a factory, as shown in Fig. 2. The toy cars were placed at in the room, and two loudspeakers were arranged around a toy car to emit factory noise. For the target and noise sound, we individually collected four types of car-running sounds and four types of factory noise data emitted from two loudspeakers. Then, $K = 9$ types of pre-training datasets were generated by mixing three of the four types of car sounds and three environmental sounds at a signal-to-noise (SNR) of 0 dB. The adaptation and test datasets were generated by mixing the remaining car sound and environmental noise at an SNR of 0 dB. All sounds were recorded at a sampling rate of 16 kHz.

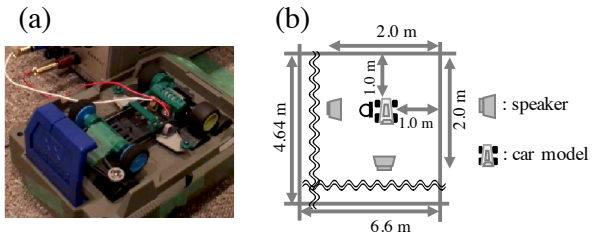

Fig. 2. Photograph of toy car (left) and arrangement of toy car and loudspeakers for simulating environmental noise (right).

Since it is difficult to generate various types of anomalous sounds, we created synthetic anomalous sounds in the same manner as in a previous study [9]. A part of the training dataset for the task of DCASE-2016 [23, 24] was used as anomalous sounds; 140 sounds including *slamming doors* , *knocking at doors* , *keys put on a table*, *keystrokes on a keyboard*, *drawers being opened*, *pages being turned*, and *phones ringing*) were selected. To synthesize the test data, the anomalous sounds were mixed with normal sounds at anomaly-to-normal power ratios (ANRs) of -20 dB. We used the area under the ROC curve (AUROC) as an evaluation metric. We also used the negative log-likelihood (NLL). Note that the higher AUROC, the better the model, whereas the lower NLL, the better the model.

The frame size of the discrete Fourier transformation was 512 points, and the frame was shifted every 256 samples. The input vectors were the log amplitude spectrum of 64-dimensional Mel-filterbank outputs with a context-window size of 5. Thus, the dimension of input vector $\boldsymbol{x}$ was $D = 704$.

4.1.2. Comparison methods

We compared the following models.

- AdaFlow: each model is first trained with data sampled from the nine pre-training datasets and then adapted with data sampled from the target dataset. The architecture is a sequence of linear transformation, AdaBN, leaky ReLU, linear transformation, and AdaBN. For adapting this model, the number of samples used was set to $N = 10, 100, 1000$.
- Normalizing Flow: each model is trained with data sampled from the nine pre-training datasets (the target dataset is not included). The architecture is a sequence of linear transformation, BN, leaky ReLU, linear transformation, and BN.
- Normalizing Flow: a model is first trained in the same manner as above, and then fine-tuned with data sampled from the target dataset. The architecture is the same as above. For fine-tuning this model, the number of samples used was set to $N = 1000$.

Table 1. Results from anomaly detection experiments.

Method	NLL	AUROC
(Chance Rate)	N/A	0.5
Norm. Flow (Trained with 9 other datasets)	53.9	0.835
Auto-encoder (Trained with 9 other datasets)	N/A	0.805
AdaFlow (Adapted with 10 samples)	92.4	0.816
AdaFlow (Adapted with 100 samples)	21.4	0.875
AdaFlow (Adapted with 1000 samples)	15.3	0.882
Norm. Flow (Fine-tuned with 1000 samples)	**13.9**	**0.887**

Table 2. Computational Time for adapting each model to the target dataset. We ran these experiments with Intel Xenon CPU (2.30GHz) on a single thread.

Method	Time [sec.]
Norm. Flow (Fine-tuned with 1000 samples)	3.23
AdaFlow (Adapted with 1000 samples)	**0.09**

- Auto-encoder: each model is trained with data sampled from the nine pre-training datasets. Since this model cannot be used for density estimation, we only evaluate AUROC. The architecture is a sequence of linear transformation (the output dimension is 128), ReLU, linear transformation (the output dimension is 64), ReLU, linear transformation (the output dimension is 128), ReLU, and linear transformation (the output dimension is 704).

4.2. Objective evaluations

The experimental results are shown in Tables 1 and 2. From these results, we observed the following things:

- Both Normalizing Flows and AdaFlow outperformed Auto-encoder. This observation indicates the superiority of Normalizing Flows over Auto-encoder in anomaly detection.
- AdaFlow outperformed Normalizing Flow trained with nine pre-training datasets, even when it was trained with 10 samples. This indicates the superiority of AdaFlow over non-fine-tuned Normalizing Flow.
- The larger the amount of data used for adapting AdaFlow, the better both the metrics were. This indicates that the amount of data used for adaptation should be as large as possible.
- AdaFlow can be adapted to a new dataset about 36 times faster than fine-tuning-based Normalizing Flow adaptation, with slight accuracy decrease. This indicates that AdaFlow is equally accurate yet much more efficient than fine-tuning-based adaptation.

5. APPLICATION TO UNPAIRED CROSS-DOMAIN TRANSLATION

Though AdaFlow was originally designed for conducting density estimation on multiple domains, we demonstrate that it can be also used for the unpaired cross-domain translation problem, in which one has to train a cross-domain translation model without paired data. We propose the unpaired cross-domain translation framework with AdaFlow in Fig. 1 (c). Given a trained AdaFlow model, data belonging to one domain is first projected to the latent space with that domain's AdaBN statistics, and after that the obtained latent variable is reprojected to the data space with the target domain's AdaBN statistics.

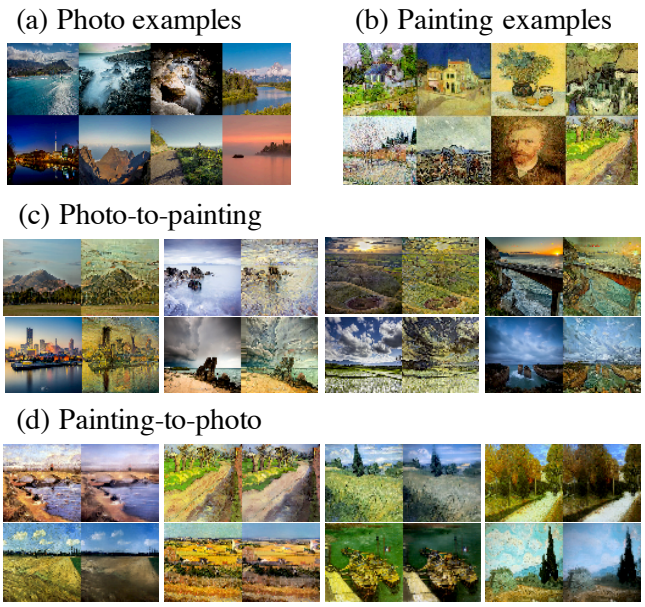

Fig. 3. Result examples of unpaired cross-domain translation. (a) training data examples of photos, (b) training data examples of paintings, (c) translation result examples of photo to painting, and (d) translation result examples of painting to photo. In both (c) and (d), input images are shown on the left side, and output images are shown on the right side. Best viewed in monitor.

We used two datasets for these experiments: the first one consisted of 400 photos, and the second one consisted of 400 paintings drawn by Van Gogh. Examples are shown in Fig. 3 (a) and (b). As an architecture for AdaFlow, we employed a variant of Glow [25], in which activation normalization layers are replaced with AdaBN.

The cross-domain translation results are shown in Fig. 3 (c, d). We can see that unpaired cross-domain translation can be achieved via AdaFlow, even when it is trained without paired data. These results indicate that AdaFlow can be a density-based alternative to other methods for this problem, such as CycleGAN [26].

6. CONCLUSIONS

We proposed a new DNN-based density estimator called *AdaFlow*; a unified model of the NF and AdaBN. Since AdaFlow can be adapted to a new domain by just adjusting the statistics used in AdaBNs, we can avoid iterative parameter update for adaptation, unlike fine-tuning. Therefore, a fast and low-computational cost domain adaptation is achieved. We confirmed the effectiveness of the proposed method through an anomaly detection in a sound task. We also proposed a method of applying AdaFlow to the unpaired cross-domain translation problem. We demonstrated the effectiveness of using AdaFlow for the task through cross-domain translation experiments on photo and painting datasets.

AdaFlow has the potential to resolve some problems of other important tasks. One possible example is source enhancement [27–29]. It is known that the performance of DNN-based source enhancement is degraded when target/noise characteristics of test data are different from those of training data. This problem is also domain-adaptation problem, thus it might be resolved by using AdaFlow. Therefore, in the future, we plan to apply AdaFlow to other tasks including source enhancement.

7. REFERENCES

[1] Victoria Hodge and Jim Austin, "A survey of outlier detection methodologies," *Artificial intelligence review*, vol. 22, no. 2, pp. 85–126, 2004.

[2] Animesh Patcha and Jung-Min Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[3] Jiawei Han, Jian Pei, and Micheline Kamber, *Data mining: concepts and techniques*, Elsevier, 2011.

[4] Walter Andrew Shewhart, *Economic control of quality of manufactured product*, ASQ Quality Press, 1931.

[5] Bovas Abraham and George EP Box, "Bayesian analysis of some outlier problems in time series," *Biometrika*, vol. 66, no. 2, pp. 229–236, 1979.

[6] Deepak Agarwal, "Detecting anomalies in cross-classified streams: a bayesian approach," *Knowledge and information systems*, vol. 11, no. 1, pp. 29–44, 2007.

[7] Yuma Koizumi, Shoichiro Saito, Hisashi Uematsu, and Noboru Harada, "Optimizing acoustic feature extractor for anomalous sound detection based on neyman-pearson lemma," in *EUSIPCO*, 2017.

[8] Chong Zhou and Randy C Paffenroth, "Anomaly detection with robust deep autoencoders," in *KDD*, 2017.

[9] Yuma Koizumi, Shoichiro Saito, Hisashi Uematsu, Yuta Kawachi, and Noboru Harada, "Unsupervised detection of anomalous sound based on deep learning and the neyman-pearson lemma," *IEEE/ACM Trans. ASLP*, 2018.

[10] Diederik P Kingma and Max Welling, "Auto-encoding variational bayes," in *ICLR*, 2014.

[11] Jinwon An and Sungzoon Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special Lecture on IE*, vol. 2, pp. 1–18, 2015.

[12] Yuta Kawachi, Yuma Koizumi, and Noboru Harada, "Complementary set variational autoencoder for supervised anomaly detection," in *ICASSP*, 2018.

[13] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, "Generative adversarial nets," in *NIPS*, 2014.

[14] Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Ursula Schmidt-Erfurth, and Georg Langs, "Unsupervised anomaly detection with generative adversarial networks to guide marker discovery," in *IPMI*, 2017.

[15] Swee Kiat Lim, Yi Loo, Ngoc-Trung Tran, Ngai-Man Gemma Roig Cheung, and Yuval Elovici, "Doping: Generative data augmentation for unsupervised anomaly detection with gan," in *ICDM*, 2018.

[16] Danilo Rezende and Shakir Mohamed, "Variational inference with normalizing flows," in *ICML*, 2015.

[17] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio, "Density estimation using real nvp," in *ICLR*, 2017.

[18] Yanghao Li, Naiyan Wang, Jianping Shi, Jiaying Liu, and Xiaodi Hou, "Revisiting batch normalization for practical domain adaptation," in *ICLR Workshop*, 2016.

[19] Sinno Jialin Pan, Qiang Yang, et al., "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[20] Yaroslav Ganin and Victor Lempitsky, "Unsupervised domain adaptation by backpropagation," in *ICML*, 2015.

[21] Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan, and Dumitru Erhan, "Domain separation networks," in *NIPS*, 2016.

[22] Kuniaki Saito, Yoshitaka Ushiku, and Tatsuya Harada, "Asymmetric tri-training for unsupervised domain adaptation," in *ICML*, 2017.

[23] `http://www.cs.tut.fi/sgn/arg/dcase2016/`

[24] `http://www.cs.tut.fi/sgn/arg/dcase2016/download`

[25] Diederik P Kingma and Prafulla Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," in *NIPS*, 2018.

[26] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *ICCV*, 2017.

[27] Yuma Koizumi, Kenta Niwa, Yusuke Hioka, Kazunori Kobayashi, and Yoichi Haneda, "Dnn-based source enhancement to increase objective sound quality assessment score," *IEEE/ACM Trans. ASLP*, 2018.

[28] Yuma Koizumi, Noboru Harada, Yoichi Haneda, Yusuke Hioka, and Kazunori Kobayashi, "End-to-end sound source enhancement using deep neural network in the modified discrete cosine transform domain," in *ICASSP*, 2018.

[29] Shinichi Mogami, Hayato Sumino, Daichi Kitamura, Norihiro Takamune, Shinnosuke Takamichi, Hiroshi Saruwatari, and Nobutaka Ono, "Independent deeply learned matrix analysis for multichannel audio source separation," in *EUSIPCO*, 2018.