# RESOLVING INTRA-CLASS IMBALANCE FOR GAN-BASED IMAGE AUGMENTATION

*Lijyun Huang, Kate Ching-Ju Lin and Yu-Chee Tseng*

Department of Computer Science, National Chiao Tung University, HsinChu, Taiwan

**ABSTRACT**

Advanced machine learning and deep learning techniques have increasingly improved accuracy of image classification. Most existing studies have investigated the data imbalance problem among classes to further enhance classification accuracy. However, less attention has been paid to data imbalance within every single class. In this work, we present AC-GAN (Actor-Critic Generative Adversarial Network), a data augmentation framework that explicitly considers heterogeneity of intra-class data. AC-GAN exploits a novel loss function to weigh the impacts of different subclasses of data in a class on GAN training. It hence can effectively generate fake data of both majority and minority subclasses, which help train a more accurate classifier. We use defect detection as an example application to evaluate our design. The results demonstrate that the intra-class distribution of fake data generated by our AC-GAN can be more similar to that of raw data. With balanced training for various subclasses, AC-GAN enhances classification accuracy for no matter uniformly or non-uniformly distributed intra-class data.

## 1. INTRODUCTION

With the recent advances of machine learning and deep learning techniques, machines can now perform automatic classification accurately. By collecting a large number of training data, we can train a model that quickly classifies samples into several pre-defined classes. However, data imbalance among classes is a typical problem that may disturb training and worsen model accuracy. Several recent studies [1, 2] hence have investigated how to balance data distribution so as to enhance classification accuracy.

However, for classification problems, data imbalance may not only be a problem across different classes, but also a critical issue within every class. In particular, though several samples belong to the same class, their variety, however, could be fairly high. An evident example is automatic defect detection in manufacturing environments. As different subclasses of errors are all classified as defects, those defective equipments however could be very diverse. Other classification problems may face similar issues. For example, a typical object detection problem may include various species of flowers in the "flower" class. However, the numbers of training data for various species could be very different. While inter-class data imbalance has been widely investigated, less attention has been paid to *intra-class data imbalance*.

To resolve this issue, in this work, we propose AC-GAN, a data augmentation mechanism that explicitly considers intra-class data heterogeneity. We leverage a recently proposed technique, Generative Adversarial Network (GAN) [3], to augment data. However, unlike conventional GAN, which learns the original distribution of every class and may overlook minority subclasses, we derive a novel loss function that adapts to intra-class data distribution and better learns the variety of samples within a class. By doing this, AC-GAN efficiently improves similarity of fake data of all subclasses, no matter majority or minority, which can then be used to train a classification model with a better accuracy.

To achieve this goal, we should address some practical challenges. First, the subclasses of a class are usually not well defined. For example, for defect inspection, a factory usually only knows whether an equipment is defect or not, but has little clue about what type of errors it is and what the root cause is making such defect. Hence, to learn the distribution of heterogeneous subclasses of data within a class, we leverage an unsupervised clustering scheme to infer the number of subclasses and their sizes. Second, given the intra-class data distribution, it is still challenging to tune training accordingly. To resolve this problem, we propose a reinforcement learning framework, which leverages the *Actor-Critic* network [4], to systematically learn the diverse impacts of different subclasses during GAN training.

We use defect detection as an example application to evaluate the proposed AC-GAN. Our evaluation demonstrates that the fake samples generated by AC-GAN follow a distribution more similar to that of the raw data. Hence, by using those balanced fake samples for classifier training, we enhance classification accuracy no matter intra-class data are balanced or skewed as compared to conventional GAN.

## 2. RELATED WORK

Several recent efforts have investigated the data imbalance problem. There exist two traditional methods for mitigat-

This work is partially supported by Pervasive Artificial Intelligence Research (PAIR) Labs under contract No. 105-2218-E-009-003 (MoST), 107-EC-17-A-22-I1-0010 (MoEA), 106-2628-E-009-004-MY3 and 107-2221-E-009-023-MY2, and "Center for Open Intelligent Connectivity" (MoE).

978-1-5386-9552-4/19/$31.00 ©2019 IEEE
DOI 10.1109/ICME.2019.00171

ing the degree of data imbalance: downsampling and oversampling, respectively. Downsampling tries to reduce data of majority classes, while oversampling increases data of minority classes. Downsampling may lose some important information of majority classes. Though oversampling may avoid information losses, it however could lead to overfitting. Some studies have researched how to resolve the weakness of oversampling. Synthetic Minority Oversampling Technique (SMOTE) [1] generates a small number of new data by random interpolation and synthesizes the same number of samples for each class. Since SMOTE does not explicitly consider the distribution of data among classes, later work then proposes adaptive synthetic sampling (ADASYN) [2] to generate different numbers of samples for each class. Nevertheless, all the above approaches mainly generate data by duplicating and averaging existing data.

Generative Adversarial Network (GAN) [3, 5, 6] is a recently proposed unsupervised learning algorithm that generates fake data. A GAN consists of a generator network and a discriminator network that fight with each other. The trained GAN model can then be used to generate data for the minority classes [7, 8]. However, GAN mainly generates data of different classes. It hence only addresses data imbalance across various classes, but cannot characterize data imbalance within a single class. Therefore, minority subclasses of a class may not be trained well due to lack of sufficient samples. Our goal is to consider heterogeneous data distribution within every class in GAN training and improve similarity of fake data even for minority subclasses.

## 3. BACKGROUND

We summarize some background about GAN [3] and Actor-Critic (AC) reinforcement learning [4].

### 3.1. Generative Adversarial Network

GAN is a two-player zero sum game, which consists of a generator network and a discriminator network interacting with each other. The generator receives a random noise vector $z$ that follows either uniform or normal distribution to generate data. The discriminator then determines whether the data output by the generator is real or not. For the generator, the training goal is to generate fake data as similar to real data as possible so as to deceive the discriminator. For the discriminator, the task is to correctly identify whether the input data is real or fake. By such a two-player game, we can converge to both a good data generator and a good data classifier.

The model achieves the global optimum when the distribution of generated data $p_x$ converges to the distribution of real data $p_{data}$. To achieve this goal, GAN leverages the combination of the objective functions of the generator and the discriminator as its loss function as follows:

$$\begin{aligned}\min_G \max_D V(D,G) &= E_{x\sim p_{data}}[\log D(x)] \\ &\quad + E_{z\sim p_z}[\log(1-D(G(z)))], \end{aligned} \tag{1}$$

where $x$ is real data following the distribution $p_{data}$, $z$ is the input noise vector of the generator, $G(z)$ is the distribution of generated data and $D(x)$ indicates the probability of real data output by the discriminator. By identifying $D$ maximizing $V(D,G)$ and $G$ minimizing $V(D,G)$, we maximize $D(x)$, which improves the capability of the discriminator and, meanwhile, maximizes $D(G(x))$, which pushes the generator to generate similar fake data under convergence.

### 3.2. Actor Critic

Reinforcement learning is a branch of machine learning techniques. Using reinforcement learning, we can learn how to get better performance by interacting with environments. Reinforcement learning accumulates experiences through trial and error. A traditional reinforcement learning model consists of an agent, an environment, actions, states and a reward. The agent takes an action based on the observed state in the environment. Then, the environment will give the agent a reward after selecting this action. Through those interactions, more and more experiences will be accumulated, and then the action will be updated accordingly.

The AC algorithm [4] is a variant of reinforcement learning that mimics human thought behaviors. Human usually takes actions under the guidance of their own values and instincts, and the values are constantly updated by experiences. The AC model hence consists of two parts, *the actor module* and *the critic module*, respectively. The actor module is the executive mechanism of the brain, which reads the external state $s$ and outputs an action $a$. On the other hand, the critic module is considered as the values of the brain. It is often approximated by neural networks to score the output actions of the actor module based on the state $s$ and the corresponding reward $r$. Every update under an action $a$ and a state $s$ can be evaluated by temporal difference error (TD error) $\delta^a$ [9]:

$$\delta^a = r + \gamma V^a(s') - V^a(s), \tag{2}$$

where $V$ is the current value function implemented by the critic and $\gamma, 0<\gamma<1$, is a time decay factor. A positive TD error indicates that the selected action $a$ is helpful for updating. The model then adjusts itself based on historical information and feedback from the environment to train the actor module.

## 4. METHODOLOGY

We now discuss how to analyze intra-class distribution and detail the design of AC-GAN. Our design consists of four steps: distribution analysis, AC-GAN, data generation and data classification. The goal of our AC-GAN is to resolve intra-class data imbalance by using GAN. To this end, we should first analyze heterogeneity of data within a class. Given this intra-class distribution, we then propose a novel framework, AC-GAN, which adapts its training according to heterogeneity among data within the same class. After training, we can augment data by using AC-GAN to generate fake data of various subclasses and better capture the diversity of data within a class. The true raw data and fake generated data are finally together fed to a convolutional neural network that is trained for classification. The following sections will elaborate the details of each step.

**Distribution Analysis:** The traditional data imbalance problem is referred to as imbalance among different classes. However, for some simple classification problems, such as defect detection, there may exist diverse subclasses of defects, which are all classified into the same class. This is because different subclasses of defect usually cannot be clearly defined and labeled individually. To analyze their heterogeneity, we should at least learn how many samples belong to each subclass, even without prior knowledge characterizing those samples.

A simple way to learn subclasses of a class is to perform unsupervised clustering, such as k-means clustering, which classifies similar samples into the same subclass. However, when data are of a large dimension, they cannot be efficiently clustered by k-means. A common way to reduce the dimension is to apply Principal Components Analysis (PCA), which can still remain the key features of data even after dimension reduction. However, we found that the visualization performance of PCA clustering is not very good, probably because PCA performs linear dimension reduction transformation, which may not be effective if correlation between features is non-linear, leading to underfitting.

With this observation, we choose to adopt the t-SNR algorithm [10] for dimension reduction. t-SNE approximates high-dimensional data by a Gaussian probability density function. Given $N$ high dimensional data, $x_1, x_2, \cdots, x_N$, t-SNE calculates the probability $p_{ij}$ proportional to the similarity between the densities of any two samples, $x_i$ and $x_j$. Then, say we want to generate $N$ low dimensional data, $y_1, y_2 \cdots, y_N$. We can similarly identify the similarities $q_{i,j}$ between any two points $y_i$ and $y_j$. While low-dimensional data may loss certain amount of information, the Gaussian distribution may not allow dissimilar points to be modeled apart. Hence, instead of using the Gaussian distribution, t-SNE lets those similarities between low-dimensional points follow the following t-distribution

$$f(t) = \frac{\Gamma\left(\left(\nu+1\right)/2\right)}{\sqrt{\nu\pi}\Gamma\left(\nu/2\right)}\left(1+t^2/\nu\right)^{-(v+1)/2}, \qquad (3)$$

where $\nu = n-1$ ($n$ is the sample size) and $v$ is the degree of freedom, which can be set to 1. We can obtain a random variable $t$ by

$$t = \frac{\bar{x}-\mu}{\sigma/\sqrt{n}}, \qquad (4)$$

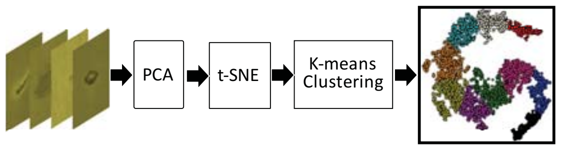


**Fig. 1**: Distribution analysis

where $\bar{x}$ is sample mean and $n$ is the sample size. To identify a proper low dimensional data that can better characterize the given high dimensional data, t-SNR calculates the similarity between the two probability distributions, $p_{ij}$ and $q_{ij}$, by using the KL divergence estimation and tries to minimize this KL divergence.

Though t-SNE achieves effective dimension reduction, it however has a much higher computational complexity. Therefore, to benefit from t-SNE but control the computational time, we use PCA to project data to a slightly lower dimension, and then input the transformed data to t-SNE to obtain two-dimensional data. Finally, the converted two-dimensional data are classified by k-means. The optimal number of clusters can be searched. Fig. 1 illustrates the flow of our intra-class distribution analysis and an example clustering result.

**AC-GAN:** After data preprocessing, we know the number of subclasses and their sizes, i.e., number of samples. We next discuss how to leverage those information to adjust GAN training. Traditional GAN training generates massive data following the original data distribution to deceive the discriminator. Such data generation may train a model failing to recognize data in a minority subclass, but rather overfit the data in majority subclasses. To resolve this intra-class imbalanced problem, we propose to adapt the training loss function to intra-class data distribution as follows:

$$\int_x [P_{data}\times W_i(x)\times\log D(x)+P_G\times W_i(x)\times\log(1-D(G(z)))]dx, \qquad (5)$$

where $i$ is the index of the subclass that a sample $x$ belongs to and $W_i(x)$ is a weight assigned to subclass $i$ used to balance training. The rationale of this equation is to emphasize the training results of certain subclasses but lower the influence of some other subclasses. To avoid confusion between $W_i$ and the weights of neurons, we call $W_i$ an *impact* hereafter. Note that we still uniformly randomly sample images as training data, but just adapt their weights in the loss function.

The remaining question is how to determine a proper impact for each subclass. Intuitively, we aim at amplifying the training effect of minority subclasses. This can be intuitively done by configuring the impacts inversely proportional to the size of a subclass as follows:

$$W_i(x) = \frac{N_{all}/K}{N_i}, \qquad (6)$$

where $N_{all}$ is the size of all data, $N_i$ is the size of subclass $i$ and $K$ is the number of subclasses belonging to the same

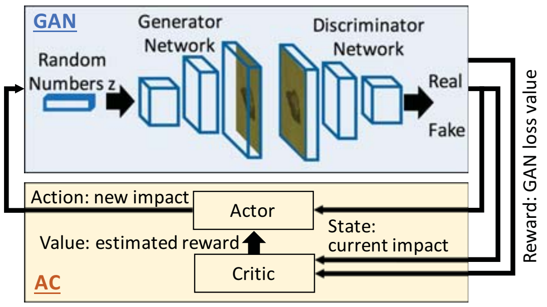


**Fig. 2**: Architecture of AC-GAN

class. The term $N_{all}/K$ can be thought of the average subclass size. Hence, this function gives a smaller subclass a higher impact, and vice versa. Note that if we consider only one class, then there is no need to have a subscript for $K$. If we consider multiple classes, then it needs a subscript indicating the corresponding class. By doing this, we can better balance training for heterogeneous subclass sizes.

While such impact assignment is intuitive, it is quite ad-hoc. We wonder whether there exists a more systematic method that can identify a proper impact setting. To achieve this goal, we further propose to leverage reinforcement learning to automatically adjust those impacts. In particular, we propose actor-critic GAN (AC-GAN), which incorporates a reinforcement learning model, called the actor-critic (AC) algorithm [4], with GAN, as illustrated in Fig. 2. The purpose of including this AC network is to train the impacts of the loss function in parallel with GAN training.

In traditional GAN, the loss function is used to jointly evaluate how well the fake data output by the generator is and how well the discriminator can inspect. As we leverage a weighted loss function to resolve intra-class data imbalance, we now aim at training the impacts using the AC model after each epoch of GAN training. To do so, we regard these impacts $W_i$ as the state $s$, which is the input of the actor network. The size of input is $K * C$, where $K$ is the number of subclasses of a class and $C$ is the number of classes. The values of those impacts range between 0 and 1, which are initially set to some random values. To control the scale of the loss function, we further normalize the impacts such that the sum of all the impacts equals 1. The output of the actor network is an action $a = [a_1, a_2, \cdots]$, each of which is defined as the gradient of each impact $W_i$. The size of output is also $K * C$. We then use the weighted loss function of the GAN (Eq. 5) after impact update as the reward function for the AC model to evaluate the performance of its action.

The calculated reward $r$ and state $s$ are used as the input of the critic network in the next iteration. The output values of the critic network $v(s)$ is used to evaluate whether the impacts updated by the actor make the loss function converge to a smaller value. In order to get a better impact value, we must update the actor and the critic iteratively. We use the

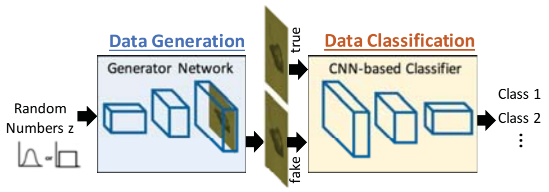


**Fig. 3**: Data generation and classification

TD error formula to evaluate whether the updated impacts are beneficial, which can be expressed as

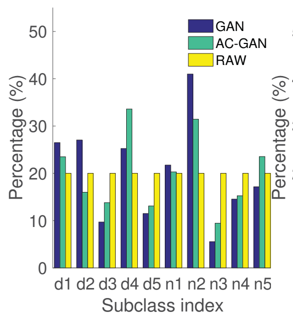
$$\delta^a = r + \gamma V^a(s') - V^a(s), \quad (7)$$

where the reward $r$ is set to the inverse of GAN's loss since the objective of weight adaptation is to converge to a smaller GAN loss. After each update, we again normalize the impacts in order to make sure that the sum of impacts of subclasses within a class is still 1. The above GAN training and AC training will be done iteratively until they together converge.

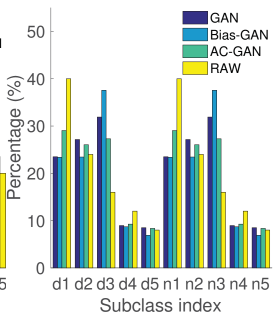
**Data Augmentation and Classification** When AC-GAN training terminates, we extract only the generator network for data augmentation. To generate a massive amount of fake data, we input one-dimensional vectors of Gaussian random numbers to the generator network, which fixes its weights trained by AC-GAN. Those fake data can then be used to train another convolutional neural network (CNN) classification model. Though we do not explicitly control the number of samples of various subclasses, our model can generate data of higher similarity no matter for minority subclasses and majority subclasses. The generated fake data and the true raw data will together be fed into CNN training, as illustrated in Fig. 3, for improving classification accuracy.

## 5. EVALUATION

We use a metal foil defects database from a local manufacturing company to evaluate the performance of our design. The task is to detect defects on the surface of metal foil automatically. There are in total 20,000 samples in the dataset, each of which is an RGB image with the resolution of $100 \times 100$ pixels. Among them, 10,000 are defective images and 10,000 are non-defective images. To control the degree of data imbalance, we first use t-SNR to analyze intra-class data distribution and classify data in each class (defect/non-defect) into five subclasses. The percentages of five subclasses of the defect class are 8%, 12%, 16%, 24% and 40%, respectively. After clustering, we then sample 500 defect and 500 non-defect images according to their subclasses to create training data with different degrees of imbalance. In particular, we try uniform sampling and skew sampling. For uniform sampling, we let all the subclasses of the equal size. Hence, we sample the same number of images from each subclass. For skew sampling, we sample data from different subclasses following

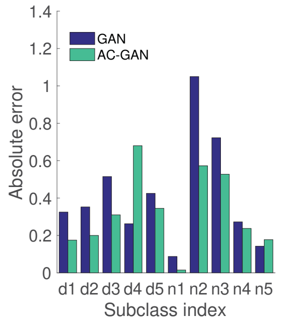


(a) Uniform sampling

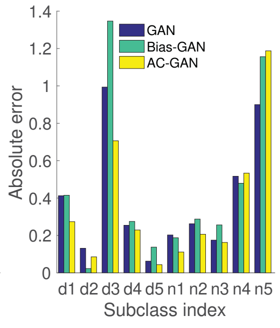


(b) Skew sampling

**Fig. 4**: Histogram of raw and fake data



(a) Uniform sampling



(b) Skew sampling

**Fig. 5**: Absolute error of intra-class distribution

their original distribution. That is, there are different numbers of samples for heterogeneous subclasses. After such data filtering, we then adopt a split ratio of 80 to 20 for training and testing. That is, 1,000 detective/non-detective images are randomly selected for training, while additionally randomly-selected 250 detective/non-detective images are used for testing. In the class of defective images, there exist faults of different types, which are not explicitly labeled.

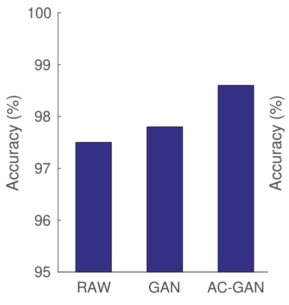
Our framework consists of three major parts: a GAN, an AC network and a CNN for classification. For GAN, we use the input size of 72, which includes 64 dimensional random numbers following Gaussian distribution and two labels, i.e., detect or non-detect. The label information is used to specify which class of images should be generated. In GAN, the generator is a CNN with 10 residual layers, and outputs an image of size $100 \times 100$ pixels, which will be the input of the discriminator. The discriminator is a CNN with 9 residual layer and one fully-connected NN layer, and outputs a real number, ranging between 0 and 1, that represents the likelihood of the input images being true. The value of '1' means that the discriminator completely ensures that the input image is real, while the value of '0' indicates a fake input image. The learning rate of the generator and the discriminator are both set to 0.0002. We use Adam for optimization and adopt ReLU as the activation function. GAN is trained for 600 epochs.

For the AC network, both the actor and critic network have one input layer, one hidden layer, which has 20 units of neuron, and one output layer. Since we cluster each class of data into five subclasses, the input size of the actor network is hence a 10-dimensional vector representing the impacts of different subclasses used in AC-GAN. The output of the actor network is the 10-dimensional gradient of the input impact vector. The learning rates of the actor and the critic networks are set to 0.001 and 0.01, respectively. The reward discount in TD error is set to 0.9. We again use Adam optimization and adopt ReLU as the activation function.

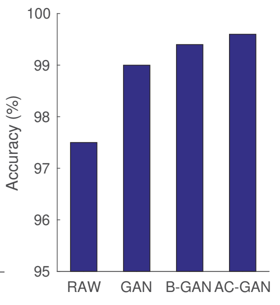
For CNN, we use the generated images and raw images for training. The model then outputs the decision of defect or non-defect. As some defects are actually very small, we resize the images to a resolution of $200 \times 200$ for training. The learning rate is set to 0.00017. Again, Adam is used for optimization, and ReLU is used as the activation function. We use an early stopping mechanism to terminate the training procedure. When the change in validation losses is less than 0.001 and the number of epochs exceeds 15, we terminate training.

**Intra-Class Distribution of Generated Data:** We first check whether the data generated by AC-GAN can follow an intra-class distribution similar to that of raw data. Since we do not have the label (ground truth) of the intra-class subclass of each sample, we train a CNN model to infer this label. Specifically, after our intra-class data clustering, we obtain $K$ subclasses of samples within each class. In each iteration of CNN training, we input an image of the raw data set, and its corresponding label is the subclass it belongs to. By doing this, we obtain a CNN model that can label an input image a subclass. We then use this CNN model to estimate the intra-class data distribution of raw data and fake data generated by GAN-based algorithms. In particular, we feed 1,000 raw data into the CNN model to find the percentage of samples in each subclass. Similarly, we let each GAN-based algorithm generate 5,000 fake images and use the same CNN model to label them and find their distribution.

Fig. 4a and Fig. 4b plot the histogram of different subclasses of raw data and fake data generated by GAN, Bias-GAN and AC-GAN when the training data are sampled in a uniform and skew manner, respectively. X-axis represents the index of each intra-class subclass, while y-axis shows the percentage of data belonging to this subclass. Bias-GAN also uses our proposed weighted loss function, but simply assigns the impacts inversely proportional to the subclass sizes. When the intra-class data are uniformly distributed, Bias-GAN is actually equivalent to traditional GAN and hence omitted in the figure of uniform sampling. The figures show that the fake data generated by AC-GAN follow a distribution more similar to that of the original raw data. This explains that, by adaptive impact adaptation, we can balance the training to learn from different subclasses of samples within a class. Hence, AC-GAN can enhance similarity between fake data and raw



(a) Uniform Sampling



(b) Skew Sampling

**Fig. 6**: Performance comparison

data. This phenomenon is especially obvious in the case of skew sampling, which confirms that, when intra-class data are heterogeneously distributed, it becomes necessary to consider various impacts of training data belonging to heterogeneous subclasses.

To further show the similarity between the intra-class distribution of raw data and that of fake data, we calculate their absolute errors by $e_i = |p_i - q_i|/p_i$, where $p_i$ and $q_i$ are the percentage of raw and fake images, respectively, of subclass $i$. We illustrate this absolute error of different subclasses in Fig. 5a and Fig. 5b for uniform sampling and skew sampling, respectively. The results demonstrate that, for most of subclasses, AC-GAN produces a smaller absolute error, which implies that its intra-class distribution is more similar to that of the raw data. Moreover, the error of AC-GAN in the case of skew sampling is also smaller than that in the case of uniform sampling. This confirms the necessity of using AC-GAN's impact adaptation to consider intra-class heterogeneity.

**Performance Comparison:** We next examine whether the more similar distribution of data generated by our AC-GAN can improve accuracy of defect detection. Here, we compare the performance of AC-GAN with conventional GAN and Bias-GAN. We again evaluate the performance for different intra-class data distribution, i.e., uniform sampling and skew sampling. Without data augmentation, we use 1,000 randomly sampled raw data for CNN training. With augmentation, we use each GAN-based model to generate 2,000 additional fake data to be trained together with the raw data.

Fig 6a and Fig 6b plot the classification accuracy of uniform sampling and skew sampling, respectively. Bais-GAN is again excluded in the figure of uniform sampling since it is equivalent to GAN in this case. The results show that our AC-GAN and Bias-GAN are both more effective than traditional GAN since they better learn from both majority and minority subclasses. The gain is higher when the intra-class data distribution is skew. We hence conclude that the impact-adaptive loss function of AC-GAN is effective for addressing intra-class data imbalance.

## 6. CONCLUSION

In this work, we present AC-GAN, a GAN-based algorithm that augments data with consideration of intra-class data imbalance. Unlike existing solutions that mainly address data imbalance across different classes, we focus on analyzing heterogeneity among data within the same class. By learning the intra-class data distribution, AC-GAN then leverages a novel weighted loss function to adjust the impacts of various intra-class subclasses on training. We then investigate two impact adaptation options: proportional impact assignment and AC-based weight tuning. The experimental results show that, while augmenting data via Bias-GAN is light weight and achieves better classification accuracy, AC-GAN can learn more proper impacts for heterogeneous intra-class subclasses and further improve classification accuracy. Also, higher heterogeneity among data within a class, larger improvement AC-GAN can achieve.

## 7. REFERENCES

[1] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “Smote: synthetic minority over-sampling technique,” *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[2] H. He, Y. Bai, E. A. Garcia, and S. Li, “Adasyn: Adaptive synthetic sampling approach for imbalanced learning,” in *IEEE International Joint Conference on Neural Networks*, 2008.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in Neural Information Processing Systems*, 2014.

[4] V. R. Konda and J. N. Tsitsiklis, “Actor-critic algorithms,” in *Advances in Neural Information Processing Systems*, 2000.

[5] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” *arXiv preprint arXiv:1411.1784*, 2014.

[6] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein gan,” *arXiv preprint arXiv:1701.07875*, 2017.

[7] J. Wang and L. Perez, “The effectiveness of data augmentation in image classification using deep learning,” Tech. Rep., 2017.

[8] G. Douzas and F. Bacao, “Effective data generation for imbalanced learning using conditional generative adversarial networks,” *Expert Systems with Applications*, vol. 91, pp. 464–471, 2018.

[9] R. S. Sutton and B. Tanner, “Temporal-difference networks,” in *Advances in Neural Information Processing Systems*, 2005.

[10] L. v. d. Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.