

# Handling Class Imbalance Using Regularized Auto-Encoders with Weighted Calibration

*A dissertation submitted in  
partial fulfilment for the degree of*

**Master of Technology**

in

**Computer Science**

*by*

**Anup Mandal**

Roll no. - [CS2204]

*under the supervision of*

**Dr. Swagatam Das**

Professor

Electronics and Communication Sciences Unit



INDIAN STATISTICAL INSTITUTE, KOLKATA

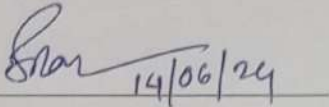
**June, 2024**



## CERTIFICATE

This is to certify that the dissertation entitled "Handling Class Imbalance Using Regularized Auto-Encoders with Weighted Calibration" submitted by Anup Mandal to the Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of Master of Technology in Computer Science is an authentic and genuine record of the research work conducted by the candidate under my supervision and guidance. I affirm that the dissertation has met all the necessary requirements in accordance with the regulations of this institute.

June, 2024



Dr. Swagatam Das

Professor

Electronics and Communication Sciences Uni,  
Indian Statistical Institute,  
Kolkata-700108

# Acknowledgement

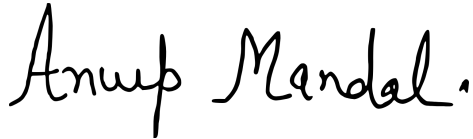
I would like to thank my supervisor, Dr. Swagatam Das for the opportunity provided to complete my dissertation. Most importantly, I would like to express my utmost gratitude to Mr. Priyobrata Mondal, SRF, who helped me with every difficulty I faced and guided me with his knowledge of deep learning techniques and PyTorch coding.

I also acknowledge the computational resources provided to me which helped me carry out my research work. I would like to thank my friends for their help, emotional support and invaluable companionship.

# Declaration

I, **Anup Mandal**, with Roll No. **CS2204** hereby declare that the material presented in the dissertation titled **Handling Class Imbalance Using Regularized Auto-Encoders with Weighted Calibration** represents original work carried out by me for the degree of **Master of Technology, Computer Science** at **Indian Statistical Institute, Kolkata**.

Furthermore, I affirm that no sections of this report have been sourced or copied from external references without proper attribution. I am aware that any instances of plagiarism or the utilization of unacknowledged materials from third parties will be treated with utmost seriousness and consequences.



June, 2024

---

**Anup Mandal**

Roll no.- CS2204

# Abstract

DeepSmote uses the SMOTE technique in the latent space of an Autoencoder-Decoder model to produce high fidelity images for imbalanced data. But it is limited by 2 essential artillery: over-fitting the data and a lack of continuity of the latent space thus giving bad results. To overcome this, a number of regularized autoencoders have been proposed. Furthermore, the latent space was oversampled using a variety of approaches. Finally, a new method is a weighted calibration to the latent space of minority classes and has proven to be pretty accurate compared to other tested methods.

**Keywords:** Classification, Class imbalance, Calibration, Latent Space, Regularized auto-encoders

# Contents

<b>Certificate</b>	i
<b>Acknowledgement</b>	ii
<b>Abstract</b>	iv
<b>1 Introduction</b>	1
1.1 Imbalanced data and its problem	1
1.2 Problems and limitations of usual SMOTE for imbalanced real image data	2
1.3 DeepSmote: Deep Learning and SMOTE	3
1.4 DeepSmote challenges and possible solutions.	4
1.4.1 Challenges	4
1.4.2 Solutions	4
<b>2 Literature Review</b>	6
2.1 Imbalanced Data in Machine Learning	6
2.2 Data-Level Approaches: Under-Sampling and Over-Sampling	6
2.2.1 Undersampling Techniques	6
2.2.2 Oversampling Techniques	7
2.3 Ensemble Approaches	7
2.4 Algorithm-Level Approaches	7
2.5 Advances in Deep Learning and Imbalanced Data	8
2.6 Generative Models for Oversampling	8
2.6.1 BAGAN and GAMO	8
2.6.2 DeepSmote	8
2.6.3 Addressing Overfitting and Continuity	9
2.7 Calibration	9
<b>3 Preliminaries and some useful algorithms</b>	10
3.1 Synthetic Minority Over-sampling Technique (SMOTE)	11
3.2 DeepSmote	11

3.2.1	DeepSmote Algorithm	13
3.3	ADASYN	13
3.4	Denoising Autoencoder	15
3.5	Sparse Auto-encoder	16
3.6	Variational Autoencoder (VAE)	17
3.6.1	VAE Loss Function	17
3.7	Variational Autoencoder (VAE) Loss Function	19
3.7.1	Components	19
3.7.2	Dependency	20
3.7.3	Trade-Off	20
3.7.4	$\beta$ -VAE	20
3.7.5	Benefits of $\beta$ -VAE	20
3.8	Gaussian Mixture Model (GMM)	21
3.8.1	Significance of GMM	21
3.9	Gaussian Multivariate Normal Distribution	21
3.9.1	Prior and Posterior Distributions	22
<b>4</b>	<b>Our contribution and proposed methods</b>	<b>24</b>
4.1	Solutions to DeepSmote Problems	24
4.2	Regularized Auto-Encoders	24
4.2.1	Denoising Auto-Encoder	24
4.2.2	Sparse Auto-Encoder	24
4.3	Ensuring Latent Space Continuity	25
4.3.1	Training on Imbalanced Data	25
4.3.2	SMOTE in Latent Space	25
4.3.3	Decoding	25
4.4	Using ADASYN for Enhanced Oversampling	25
4.4.1	Training the Auto-Encoder	25
4.4.2	Applying ADASYN	25
4.4.3	Decoding and Classification	26
4.5	Weighted Calibration Balancing	26
4.5.1	Class Partitioning	26
4.5.2	Gaussian Mixture Model Partitioning	26
4.5.3	Minority Class Processing	27
4.5.4	Synthetic Latent Vectors	27
4.5.5	Median Class Processing	27
4.6	Algorithm	28
4.7	Overall Algorithm	29



<b>5 Experiments and Results</b>	<b>30</b>
5.1 Data Overview	30
5.2 Evaluation Metrics	31
5.2.1 Recall (Macro Average)	31
5.2.2 Precision (Macro Average)	31
5.2.3 F1 Score (Macro Average)	32
5.3 Training Details	32
5.4 Results	33
5.4.1 MNIST	33
5.4.2 CIFAR 10	35
<b>6 Conclusion and Future work</b>	<b>37</b>
<b>Bibliography</b>	<b>39</b>

# List of Figures

1.1 On training for multiple epochs, a neural network gradually learns the decision boundaries of the data points. . . . .	2
3.1 Pictorial view of SMOTE. . . . .	10
3.2 DeepSmote [1] . . . . .	13
3.3 Denoising Auto-encoder . . . . .	16
3.4 Sparse Auto-encoder . . . . .	18
5.1 Class distribution of data points in the Imbalanced MNIST dataset .	31
5.2 Class distribution of the Imbalanced CIFAR10 dataset . . . . .	32

# List of Tables

5.1 Comparison of DeepSmote results with other results on Imbalanced	
MNIST . . . . .	34
5.2 Comparison of DeepSmote results with other results including Weighted	
Calibration in Denoising Auto-encoder on Imbalanced MNIST . . . . .	34
5.3 Comparison of DeepSmote results with proposed method's results on	
Imbalanced CIFAR10 . . . . .	35
5.4 DeepSmote results with proposed method's results on Imbalanced	
CIFAR10 . . . . .	36
5.5 Comparison DeepSmote results with proposed method's results on	
Imbalanced CIFAR10 . . . . .	36



# Chapter 1

## Introduction

### 1.1 Imbalanced data and its problem

Teaching machines from imbalanced datasets is one of the most interesting open challenges in the field of machine learning [2]. For imbalanced class distributions, the classifiers are always inclined to the major class, which may cause great errors or even ignore the minor classes. This becomes an even larger problem in mission-critical applications such as healthcare, where minority class detection may represent identifying rare diseases to intrusion detection systems that have to uncover infrequent but dangerous security breaches, image recognition systems that need to correctly classify rare objects or fraud detection systems that are meant to expose rare fraudulent transactions. Therefore, dealing with the class imbalance problem has been one of the main interests to the research community for more than two decades [3].

It has become evident from recent developments that there are other problems than class disparity. Limited sample sizes, minor disjuncts, challenging and borderline cases, and the dynamic nature of streaming data are further contributing factors [3]. As new fraud strategies surface, for example, the minority class in credit card fraud detection—fraudulent transactions [4] is not only uncommon but also ever-changing. For solutions to be developed that can effectively address these data difficulties, these complexities call for ongoing research and innovation.

Deep learning has showed considerable potential in a variety of applications. It is known for its extraordinary powers in cognitive and recognition tasks. Deep learning models are particularly susceptible to unbalanced data distributions, notwithstanding their advantages. In addition, they must cope with complicated data structures, comprehend how unbalanced data affects the extracted embeddings, adjust to the

constantly changing nature of data, and absorb knowledge from a large number of classes [5], [6]. In the field of medical imaging [7], for instance, deep learning models must reliably identify uncommon illnesses from big, intricate image datasets in which the proportion of normal cases to abnormal ones is significantly higher [8].

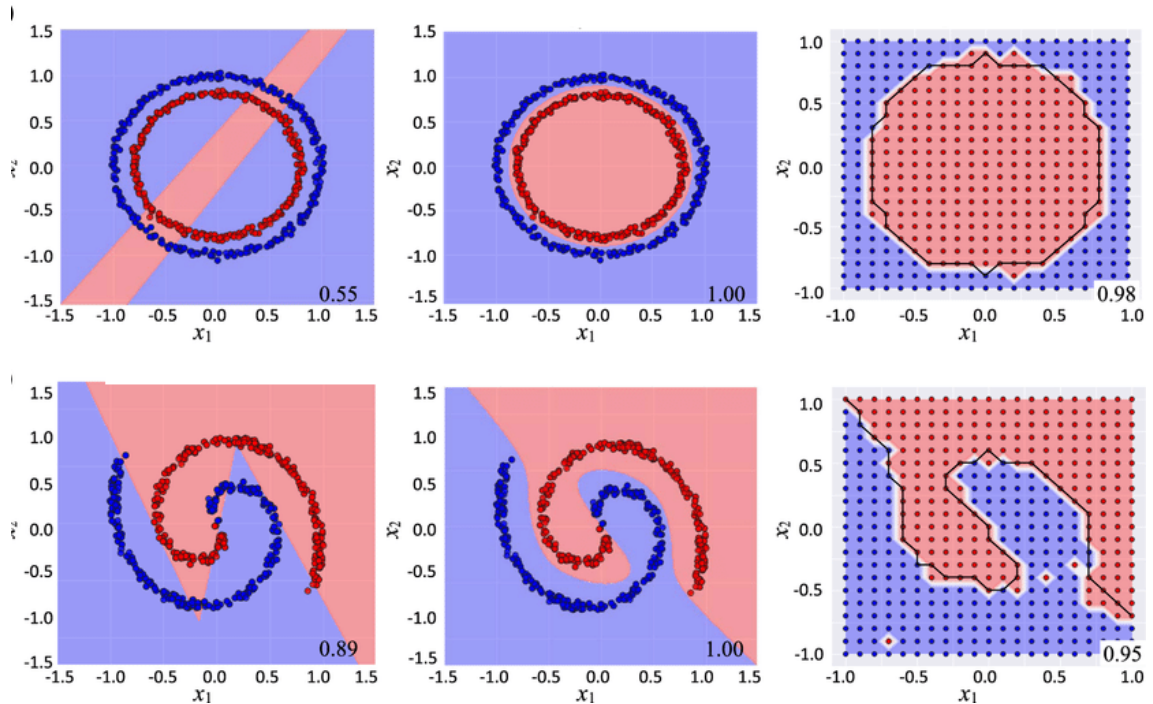


Figure 1.1: On training for multiple epochs, a neural network gradually learns the decision boundaries of the data points.

## 1.2 Problems and limitations of usual SMOTE for imbalanced real image data

The machine learning community was aware of the problem of class imbalance even before to the emergence of deep learning. Thus, while new models of deep learning were being created, scientists were also trying to make these models more resistant to class imbalance. Since deep learning models were developed using solutions already in existence for standard classifiers, this first appeared to be simple. Nevertheless, this method soon showed its shortcomings, particularly with regard to end-to-end deep learning models.

In the domain of deep learning, for example, SMOTE (Synthetic Minority Oversampling Technique) [9] and related oversampling techniques, which were very successful for conventional classifiers, proved less successful for two key reasons. Feature extraction and classification in end-to-end deep learning are linked processes that are taught concurrently using backpropagation of a single loss function. Initially,

SMOTE-type approaches create fresh samples in the feature space. Integrating SMOTE into deep learning frameworks is difficult because of this reliance.

Second, to generate new samples and surround areas depending on a distance measure, SMOTE use convex combinations of existing samples. However, there are limitations to this method because deep learning models are often used to non-vector datasets such as images, audio, and video. Convex combinations of samples can produce inconsequential findings, and standard distance measurements are not adequately characterized in these situations. For example, from the Fashion-MNIST dataset, two T-shirts or tops that are more visually comparable based on Euclidean distance may appear closer to each other than they actually are. Furthermore, the MNIST dataset’s convex merging of two images of the "9" yields no meaningful "9," much less a recognizable number [10].

### 1.3 DeepSmote: Deep Learning and SMOTE

Even after twenty years of progress, handling unbalanced data is still a major problem for contemporary machine learning models. The necessity to solve this problem has been made much more apparent by the development of deep learning. Oversampling instances and altering loss functions are the two primary methods that are frequently employed. However, oversampling may lead to mode collapse and other issues. Generative Adversarial Networks (GANs), for instance [11]. Consequently, an oversampling method is needed to balance the dataset, enhance minority classes, and process raw photographs while preserving their properties. This method must also support deep learning models.

DeepSmote [1] successfully addresses this issue by utilizing an autoencoder’s latent space. It consists of three main components: (i) an encoder-decoder architecture; (ii) SMOTE-based oversampling used in the latent space, which guarantees that convex combinations are only used to combine the necessary encoded features; and (iii) a specific loss function with a penalty term. This approach decodes images from a latent space where only significant features are modified, hence resolving the earlier problem of creating unrealistic images with SMOTE. In contrast to GAN-based methods(e.g., BAGAN, GAMO [12], [13]), DeepSmote doesn’t need a discriminator. It creates excellent synthetic images that are visually realistic and information-rich, improving minority class representation and yielding a more balanced training dataset.

## 1.4 DeepSmote challenges and possible solutions.

### 1.4.1 Challenges

Even though SMOTE oversampling is used in the latent space, DeepSmote has two major flaws. Overfitting is the first thing that limits you. Since DeepSmote uses a simple vanilla auto-encoder, it overfits training data, which makes it less accurate when it comes to test or new samples. DeepSmote might learn too much about the specifics of the training images, for instance, if it is taught on a set of handwritten numbers. So, when it comes across new images, like numbers written in slightly different styles or variations, it works much less well.

The second problem is that there is no continuity in the hidden space. In the latent space, when SMOTE is used, synthetic latent feature vectors are made. Still, the input training examples are used to teach the auto-encoder how to make a certain set of encoded feature latent vectors. The decoder may make silly or unrealistic pictures when it decodes generated latent vectors. As an example, the auto-encoder could be taught to store data in a facial identification dataset about things like eye position, nose shape, and mouth width. If SMOTE makes a fake latent vector that is not in the same range as these learned properties, the decoder might make a face or expression that doesn't look right, with eyes or expressions that are in the wrong place.

These problems make it clear how hard it is to use SMOTE and regular auto-encoders in the latent space. They also show how important it is to find more reliable methods that can work with new data and keep the continuity of the latent space so that realistic fake images can be made.

### 1.4.2 Solutions

We talk about these problems in this report in a step-by-step way. We used two different kinds of regularized autoencoders, each with its own unique penalty loss function, to avoid overfitting. The first one is a Deep Denoising Autoencoder [14] that adds Gaussian noise as a loss. The second type is a Deep Sparse Autoencoder [15], which penalizes each node based on its average absolute value in the flattened hidden layers. This pushes nodes toward zero to promote sparsity. In a later part, we'll go over these approaches in more mathematical detail.

Next, we used SMOTE on the latent space of these regularized autoencoders to make fake latent vectors. Then, these vectors were turned into real pictures to fix



the dataset’s imbalance by oversampling it. The results are much better when this method is used, according to our research.

We then addressed the problem of continuity preservation in the latent space. In order to do this, we used a Variational Autoencoder (VAE) [16], which makes use of the KL divergence penalty loss to guarantee that the distribution of the latent feature vectors is comparable to that of a multivariate standard Gaussian distribution. After applying SMOTE in the latent space, the autoencoder’s reconstruction capacity may be compromised by adding the KL divergence penalty, leading to hazy images. This problem is most noticeable when using intricate image datasets such as CIFAR-10. Finally, we made the results even better by replacing the latent space SMOTE with ADASYN [17], a method for adaptive oversampling.

At the end We introduced a novel technique called **”Weighted Calibration Balancing”** in the latent space of the minority classes in order to obtain better results. Details will be discussed later in this report.

# Chapter 2

## Literature Review

### 2.1 Imbalanced Data in Machine Learning

Imbalanced data is a big problem in machine learning. Researchers like researchers like Japkowicz (2000) [18] and Chawla et al. (2002) [9] were the first to notice this issue. The first studies showed that unequal class distributions can throw off model predictions, which usually leads to poor results for minority classes. Traditionally, this problem was dealt with by doing things like not sampling enough of the majority class and sampling too much of the minority class. The Synthetic Minority Over-sampling Technique (SMOTE), which was created by Chawla et al. (2002) and is a key part of this work, creates fake samples by connecting examples from different minority groups.

### 2.2 Data-Level Approaches: Under-Sampling and Over-Sampling

At the data level, under-sampling and over-sampling are two ways to deal with class mismatch. Under-sampling tries to balance the dataset by reducing the number of samples from the majority class. It causes the information loss of the majority class. On the other hand, too much sampling increases the number of samples from minority classes. Advanced methods have been created in this area, including:

#### 2.2.1 Undersampling Techniques

Undersampling techniques aim to balance the class distribution by reducing the number of majority class instances. One common method is Random Undersampling

(RUS), which involves randomly removing majority class instances to match the number of minority class instances.

Edited Nearest Neighbors (ENN), described by Wilson (1972) [19], removes majority class instances that are misclassified by their k-nearest neighbors, thereby cleaning the dataset and improving classifier performance.

Tomek Links, introduced by Tomek (1976) [20], identify pairs of instances from different classes that are each other's nearest neighbors. Removing the majority class instance in each Tomek Link pair helps to clean the class boundary, improving classifier accuracy.

### 2.2.2 Oversampling Techniques

For example, Chawla et al. (2002)'s SMOTE (Synthetic Minority Over-sampling Technique) creates fake samples by connecting minority class instances. ADASYN, or Adaptive Synthetic Sampling, was created by He et al. (2008) [17] and focuses on creating fake samples for more difficult minority class situations. KMeans-SMOTE, created by Last et al. (2017) [21], combines k-means clustering with SMOTE to create samples. According to Han et al. (2005) [22], Borderline-SMOTE aims to create synthetic samples close to the decision border.

## 2.3 Ensemble Approaches

When working with uneven datasets, ensemble approaches combine many models. It also works well. Specific variations on techniques like bagging, boosting, and stacking can be made to help the minority class do better. Two well-known ensemble methods that deal with class mismatch are Balanced Random Forest (BRF) [23] and EasyEnsemble by Liu et al [24]. These methods integrate resampling techniques into the ensemble framework to effectively handle imbalanced classes.

## 2.4 Algorithm-Level Approaches

Algorithm-level methods are techniques used to enhance learning algorithms in order to effectively deal with imbalanced data. These methods involve modifying decision thresholds, changing class weights in loss functions, and integrating cost-sensitive learning techniques [25]. These changes guarantee that models provide greater focus to minority classes during the training process.

## 2.5 Advances in Deep Learning and Imbalanced Data

With the rise of deep learning, the problem of uneven data has become even more difficult. Deep learning models, which are great at finding trends in large amounts of data, can become very biased toward majority classes. This bias makes it hard for minority class estimates to work, as Buda, Maki, and Mazurowski (2018) [26] showed. To deal with this, ideas have been put forward like changing the loss function. Lin et al. (2017) [27] came up with the Focal Loss, which shifts the learning focus to examples that are hard to identify. This makes the model work better on datasets that are not balanced.

## 2.6 Generative Models for Oversampling

For example, Goodfellow et al. (2014) presented generative models called Generative Adversarial Networks (GANs) that have been studied for their ability to make fake data that can even out class distributions. Within a competitive setup, GANs use a generator and a discriminator to create real-life data sets. GANs still can experience mode collapse, which limits the output variety and makes it harder to make synthetic samples that are varied.

### 2.6.1 BAGAN and GAMO

GAMO (2018) and Balanced GAN (BAGAN) by Mariani et al. (2018) are more advanced methods that try to make GAN work better with uneven data. While GAMO focuses on minority classes through an adversarial method, BAGAN uses balancing techniques within the GAN framework to pull together diverse samples. The mode collapse and training difficulty still exist for these new methods.

### 2.6.2 DeepSmote

An alternative to standard GAN-based methods called DeepSMOTE was created by Damien Dablain, Bartosz Krawczyk, and Nitesh V. Chawla in 2021. DeepSMOTE creates high-quality synthetic pictures by using SMOTE in the latent space of regularized auto-encoders. This doesn't let the mode fall apart. The three main parts of the method are a system for encoders and decoders, oversampling based on SMOTE in the latent space, and a better loss function with a penalty term. This way of making latent vectors makes them more likely to be correct and keep the qualities of the original data. This helps you work with numbers that aren't balanced.

### 2.6.3 Addressing Overfitting and Continuity

Overfitting, where models perform well on training data but poorly on unseen data, is a common deep learning issue. Regularized autoencoders, such as denoising autoencoders (Vincent et al., 2008) and sparse autoencoders (Ranzato et al., 2007), add noise or enforce sparsity in hidden layers to enhance generalization. We used a VAE with a KL divergence penalty to impose continuity in the latent space, hence encouraging a smooth distribution in the latent space. By guaranteeing that synthetic samples preserve the characteristics of the original data, this improves the performance of the model as a whole.

## 2.7 Calibration

When the data is not balanced, calibration techniques are very important because they change the classifier’s output probabilities to better match the real probabilities. Isotonic regression and Platt scaling are two methods that are used to re-calibrate models. The 2021 work ”Free Lunch for Few-shot Learning: Distribution Calibration” by Yang et al [28]. talks about advanced calibration methods that can be used with datasets that are not balanced.

A lot of work has been done in the literature to deal with uneven data in machine learning, but there are still problems, especially when trying to add these methods to deep learning frameworks. Our method is based on these ideas and uses regularized auto-encoders and new ways of manipulating latent space to make high-quality fake data. We picked over-sampling methods because they avoid the information loss that comes with under-sampling. This gives us a richer dataset for training and makes the model much better at handling datasets that are not balanced.

# Chapter 3

## Preliminaries and some useful algorithms

In this chapter we are going to discuss some related methods and useful algorithms.

**Definition 1. Imbalance Ratio (IR):**  $IR = \max\{\frac{N_i}{N_j} | i, j \in C; i \neq j\}$ , where  $N_i, N_j$  are the total no. of data points in class  $i$  and  $j$  respectively, when  $x$  is partitioned into some classes. Therefore,  $X$  can be considered as imbalanced if  $IR > 1$ .

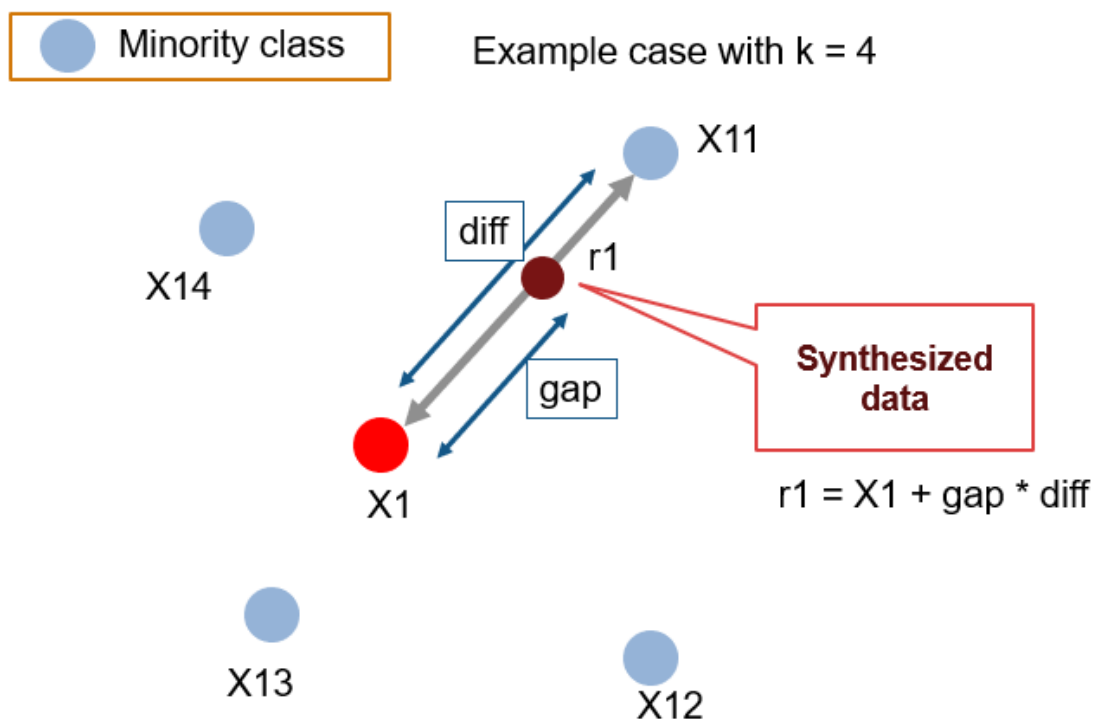


Figure 3.1: Pictorial view of SMOTE.

## 3.1 Synthetic Minority Over-sampling Technique (SMOTE)

---

### Algorithm 1 SMOTE

---

```

1: procedure SMOTE( $T, N, k$ )  $\triangleright T$ : Minority samples,  $N$ : Amount of SMOTE,
    $k$ : Number of nearest neighbors in the minority class
2:    $N \leftarrow \min(N, 100)$ 
3:    $N \leftarrow N/100$ 
4:    $Synth \leftarrow []$   $\triangleright$  Array to hold synthetic samples
5:    $NewIndex \leftarrow 0$ 
6:   for each sample  $x$  in  $T$  do
7:      $nnarray \leftarrow k$  nearest neighbors of  $x$  within the minority class
8:     for  $i$  from 1 to  $N$  do
9:        $nn \leftarrow$  random number from 1 to  $k$ 
10:       $Synth[NewIndex] \leftarrow \text{INTERPOLATE}(x, nnarray[nn])$ 
11:       $NewIndex \leftarrow NewIndex + 1$ 
12:     end for
13:   end for
14:   return  $Synth$ 
15: end procedure
16: function INTERPOLATE( $x_1, x_2$ )
17:    $\delta \leftarrow$  random number between 0 and 1
18:   return  $x_1 + \delta \times (x_2 - x_1)$ 
19: end function

```

---

## 3.2 DeepSmote

DeepSmote is an advanced oversampling technique that addresses class imbalance by generating synthetic samples through a deep learning approach. It combines the benefits of traditional SMOTE with deep neural networks. DeepSmote incorporates permutation loss to ensure diversity and prevent overfitting by generating synthetic samples that are more varied and representative of the minority class. By leveraging deep learning, it can capture complex data distributions, leading to improved performance in imbalanced classification tasks. This method is particularly effective in high-dimensional datasets where traditional oversampling techniques may fall short.

---

**Algorithm 2** DeepSmote
 

---

```

1: Data:  $B$ : batches of imbalanced training data
2:  $B = \{b_1, b_2, \dots, b_n\}$ 
3: Input:
4: Model parameters:  $\Theta = \{\Theta_0, \Theta_1, \dots, \Theta_j\}$ 
5: Learning Rate:  $\alpha$ 
6: Output: Balanced training set.
7: Train the Encoder / Decoder:
8: for  $e \leftarrow$  epochs do
9:   for  $m \leftarrow B$  do
10:     $E_B \leftarrow$  encode( $B$ )
11:     $D_B \leftarrow$  decode( $E_B$ )
12:     $C_D \leftarrow$  sample( class data )
13:     $E_S \leftarrow$  encode( $C_D$ )
14:     $P_E \leftarrow$  permute-order( $E_S$ )
15:     $D_P \leftarrow$  decode( $P_E$ )
16:     $P_L = \frac{1}{n} \sum_{i=1}^n (D_{Pi} - C_{Di})^2$ 
17:     $T_L = R_L + P_L$ 
18:     $\Theta := \Theta - \alpha \frac{\partial T_L}{\partial \Theta}$ 
19:   end for
20: end for

```

---



---

```

1: Generate Samples:
2: for  $i \leftarrow$  number of minority classes do
3:    $C \leftarrow$  select( class data )
4:    $E \leftarrow$  encode( $C$ )
5:    $G \leftarrow$  SMOTE( $E$ )
6:    $S \leftarrow$  decode( $G$ )
7: end for

```

---



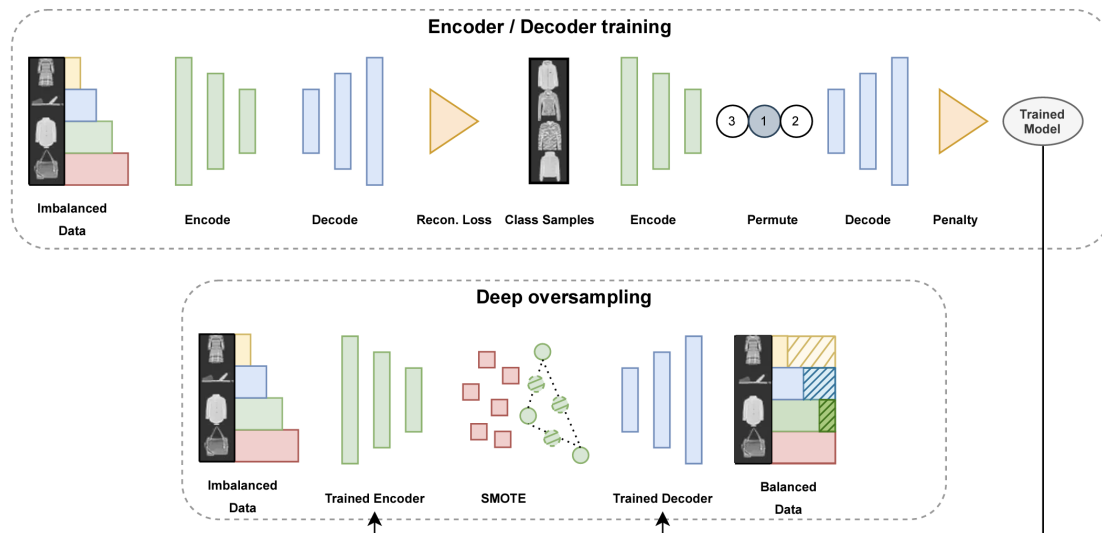


Figure 3.2: DeepSmote [1]

### 3.2.1 DeepSmote Algorithm

## 3.3 ADASYN

The ADASYN (Adaptive Synthetic Sampling) algorithm improves class balance by adaptively generating synthetic samples according to the difficulty of learning each minority class instance. Instead of creating an equal number of synthetic samples for each minority instance as in SMOTE, ADASYN uses a density distribution to determine how many synthetic samples to generate for each minority instance. This density distribution shows how hard it is to learn each instance. Instances that are harder to learn, like those with more majority class peers, get more synthetic samples. This targeted method not only evens out the dataset to the level of balance that is set by a parameter, but it also forces the learning algorithm to focus on the harder minority class examples, which leads to better performance all around. ADASYN makes a more useful and varied dataset by focusing on difficult cases. This helps the learning algorithm generalize better, which makes it better than SMOTE's uniform sampling method.

The core principle of the ADASYN algorithm is to use a density distribution  $\hat{r}_i$  to automatically determine the number of synthetic samples needed for each minority class example. The  $\hat{r}_i$  value measures the weight distribution of different minority class examples based on their learning difficulty. Essentially, it indicates how challenging each minority instance is to learn. After applying ADASYN, the resulting dataset not only achieves a balanced representation according to the desired balance level set by the  $\beta$  coefficient, but also directs the learning algorithm's focus toward

---

**Algorithm 3** ADASYN
 

---

- 1: **Input:** Training dataset  $D_{tr}$  with  $m$  samples  $\{(x_i, y_i)\}$  where  $x_i \in \mathbb{R}^n$  is an instance in the  $n$ -dimensional feature space, and  $y_i \in \{1, -1\}$  is the class label.  $m_s$  is the number of minority class samples, and  $m_l$  is the number of majority class samples such that  $m_s \leq m_l$  and  $m_s + m_l = m$ .
  - 2: **procedure** ADASYN( $D_{tr}, d_{th}, \beta, K$ )
  - 3:     Calculate the degree of class imbalance:  $d = \frac{m_s}{m_l}$ .
  - 4:     **if**  $d < d_{th}$  **then**
  - 5:         Calculate the number of synthetic samples needed:  $G = (m_l - m_s) \times \beta$ .
  - 6:         **for** each minority class sample  $x_i$  **do**
  - 7:             Find  $K$  nearest neighbors of  $x_i$  in the minority class.
  - 8:             Calculate  $r_i = \frac{\Delta_i}{K}$ , where  $\Delta_i$  is the number of majority class samples among the  $K$  nearest neighbors.
  - 9:         **end for**
  - 10:         Normalize  $r_i$  to get  $\hat{r}_i = \frac{r_i}{\sum_{i=1}^{m_s} r_i}$ .
  - 11:         **for** each minority class sample  $x_i$  **do**
  - 12:             Calculate the number of synthetic samples for  $x_i$ :  $g_i = \hat{r}_i \times G$ .
  - 13:             **for**  $j = 1$  to  $g_i$  **do**
  - 14:                 Randomly select one of the  $K$  nearest neighbors of  $x_i$ , denoted as  $x_{zi}$ .
  - 15:                 Generate synthetic sample:  $s_i = x_i + \lambda \times (x_{zi} - x_i)$ , where  $\lambda$  is a random number between 0 and 1.
  - 16:             **end for**
  - 17:         **end for**
  - 18:     **end if**
  - 19: **end procedure**
-

the more difficult-to-learn instances. If you compare to the SMOTE technique, which creates an equal amount of synthetic samples for each minority instance without taking into account their unique learning problems, our adaptive sampling strategy is a major improvement. ADASYN improves the model’s performance on imbalanced datasets and helps it generalize by emphasizing the more difficult-to-learn examples.

### 3.4 Denoising Autoencoder

Denoising autoencoders are neural network models specifically designed to learn robust data representations by reconstructing the original input from its noisy version. This approach compels the model to extract the most important features of the data, enhancing its resilience to noise and corruption.

In this model, let  $x$  denote the original clean input,  $\tilde{x}$  the noisy version of the input, and  $\hat{x} = g(f(\tilde{x}))$  the reconstructed output, where  $f$  is the encoder function and  $g$  is the decoder function.

The reconstruction loss measures how well the autoencoder can recreate the original input from the noisy input, typically using the Mean Squared Error (MSE):

$$L_{\text{reconstruction}}(x, g(f(x))) = \frac{1}{n} \sum_{i=1}^n (x_i - g(f(x_i)))^2 \quad (3.1)$$

where  $n$  is the number of elements in  $x$ .

The denoising loss, which evaluates the model’s capability to denoise the input, is often the same as the reconstruction loss when using MSE:

$$L_{\text{denoising}}(x, \hat{x}) = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{x}_i)^2 \quad (3.2)$$

The total loss for a denoising autoencoder combines the reconstruction loss and the denoising loss, which can be weighted by a regularization parameter  $\lambda$ :

$$L_{\text{total}} = L_{\text{reconstruction}} + \lambda \cdot L_{\text{denoising}} \quad (3.3)$$

Denoising autoencoders provide several advantages:

- **Feature Learning:** They efficiently learn compressed representations of features, even from noisy data.
- **Noise Robustness:** They can effectively reconstruct the original data from corrupted inputs, improving the model’s resilience to noise.

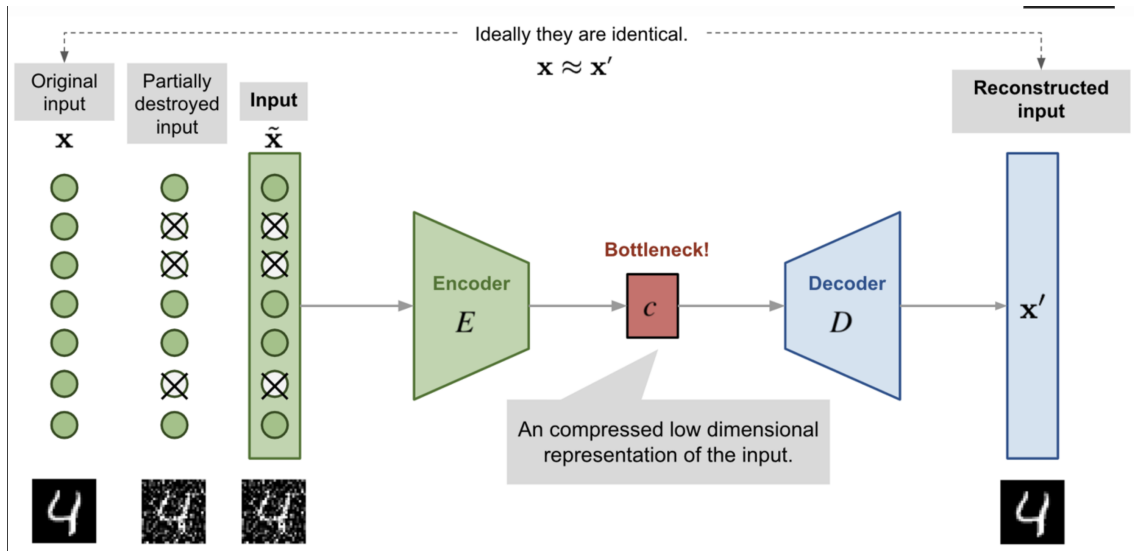


Figure 3.3: Denoising Auto-encoder

- **Generalization:** By focusing on denoising, they prevent the network from learning a trivial identity mapping, promoting the discovery of meaningful data representations.
- **Improved Performance:** The robustness to noise and ability to extract essential features often result in improved performance on subsequent tasks.

### 3.5 Sparse Auto-encoder

A sparse autoencoder includes a specific loss function to encourage sparsity in the hidden layers. This loss function is composed of:

- **Reconstruction Loss:** This measures how well the autoencoder can reconstruct the original input from the encoded representation, typically using Mean Squared Error (MSE):

$$L_{\text{recon}}(x, \hat{x}) = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{x}_i)^2$$

where  $x$  is the original input,  $\hat{x}$  is the reconstructed output, and  $n$  is the number of samples or dimensions.

- **Sparsity Penalty:** This term encourages the activation of fewer neurons, promoting a sparse representation:

$$L_{\text{sparse}} = \sum_l \frac{1}{m_l} \sum_{j=1}^{m_l} |a_j^l|$$

where  $a_j^l$  represents the activation of the  $j$ -th neuron in layer  $l$ , and  $m_l$  is the number of neurons in layer  $l$ .

- **Total Loss Function:** The total loss combines the reconstruction loss and the sparsity penalty:

$$L_{\text{total}} = L_{\text{recon}} + \lambda L_{\text{sparse}}$$

where  $\lambda$  is a regularization parameter that weights the importance of the sparsity penalty.

There are several good things that happen when you add a sparsity limit to an autoencoder:

- **Activation with Selection** only activates some neurons, which lets neurons specialize in detecting certain traits or patterns.
- **Reduction of Overfitting:** Cutting down on overfitting: Cutting down on the amount of active neurons makes it less likely that the model will remember the training data. It can now handle new info better because of this.
- **Efficient Data Representation:** The best way to describe data is in a small, meaningful way that takes into account the most important parts of the data. This is useful for tasks like reducing the number of dimensions and extracting features.
- **Computational Benefits:** The good things about computers are: With sparse models, you can get things done faster and with less memory. This helps a lot when you're working with big numbers.

## 3.6 Variational Autoencoder (VAE)

A type of neural network called variational autoencoders (VAEs) learns a compressed version of the input data and uses it to create new data points. To do this, the VAE encodes the data into a hidden space and then decodes it back to its original pattern. It is important to use VAEs because they make sure that the latent space is smooth and continuous, and that similar points in the latent space match up with similar data points in the original space. This function is needed to make new samples that make sense.

### 3.6.1 VAE Loss Function

The loss function of a VAE is made of two main parts. The reconstruction loss and the Kullback-Leibler (KL) divergence loss. The reconstruction loss confirms

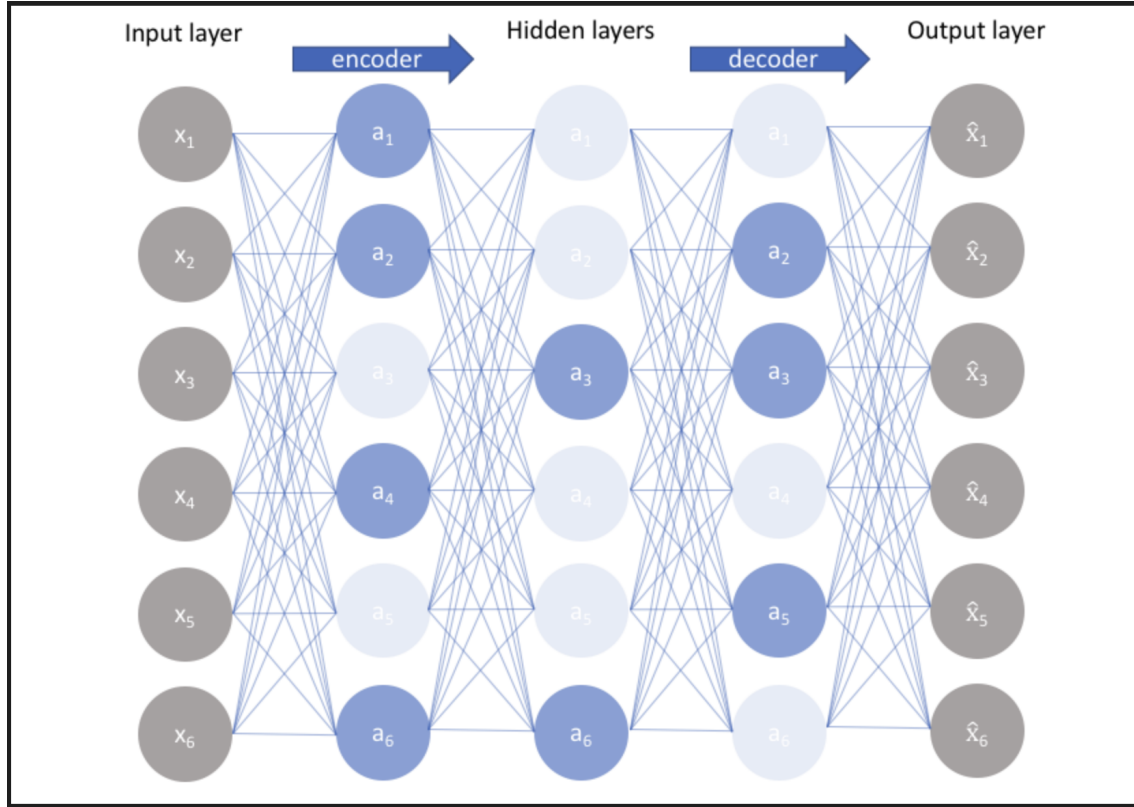


Figure 3.4: Sparse Auto-encoder

that the VAE accurately reconstructs the input data. And the KL divergence loss regularizes the latent space to follow a prior distribution, typically a standard normal distribution.

### Reconstruction Loss

The reconstruction loss  $L_{\text{recon}}$  is typically measured using Mean Squared Error (MSE):

$$L_{\text{recon}}(x, \hat{x}) = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{x}_i)^2 \quad (3.4)$$

where  $x$  is the original input,  $\hat{x}$  is the reconstructed output, and  $n$  is the number of elements in  $x$ .

### KL Divergence Loss

The KL divergence loss  $L_{\text{KL}}$  measures how much the learned latent distribution  $q(z|x)$  deviates from the prior distribution  $p(z)$ , usually a standard normal distribution:

$$L_{\text{KL}} = D_{\text{KL}}(q(z|x)||p(z)) \quad (3.5)$$

### Total Loss Function

The total loss function for a VAE is the sum of the reconstruction loss and the KL divergence loss:

$$L_{\text{total}} = L_{\text{recon}} + L_{\text{KL}} \quad (3.6)$$

#### Effect of KL Divergence Penalty on Reconstruction Loss

As long as the KL divergence term is there, the latent space of the VAE will follow the planned prior distribution. Regularization is needed to make sure that changes and continuity in the latent space stay smooth. This is important for getting real new samples. Focusing too much on the KL divergence can make the reconstruction loss worse because the model has to find a balance between correctly rebuilding the input data and keeping the latent space well-organized. When the KL divergence gets bigger, the Variational Autoencoder (VAE) might focus on keeping the structure of the latent space instead of correctly reconstructing the input, which can cause a higher reconstruction error.

Ultimately, Variational Autoencoders (VAEs) achieve a delicate equilibrium between faithfully reproducing the original input data and preserving a well-organized latent space. The KL divergence term encourages the development of a continuous and smooth latent space, which is crucial for producing coherent and consistent new samples. Nevertheless, this regularization may result in a higher reconstruction error, emphasizing the intrinsic trade-off involved in training VAEs.

## 3.7 Variational Autoencoder (VAE) Loss Function

In a Variational Autoencoder (VAE), the total loss function combines the reconstruction loss and the KL divergence loss. This can be expressed as:

$$\mathcal{L} = \mathbb{E}_{q(z|x)}[\log p(x|z)] - D_{\text{KL}}(q(z|x)||p(z)) \quad (3.7)$$

To convert this to a minimization problem, we use the negative of the ELBO:

$$L_{\text{total}} = -\mathbb{E}_{q(z|x)}[\log p(x|z)] + D_{\text{KL}}(q(z|x)||p(z)) \quad (3.8)$$

### 3.7.1 Components

1. Reconstruction Loss: Measures how well the model reconstructs the input data:

$$L_{\text{recon}} = -\mathbb{E}_{q(z|x)}[\log p(x|z)] \quad (3.9)$$

2. KL Divergence Loss: Regularizes the latent space to follow the prior distribution:

$$L_{\text{KL}} = D_{\text{KL}}(q(z|x)||p(z)) \quad (3.10)$$

### 3.7.2 Dependency

The KL divergence term and the rebuilding loss depend on: When you raise the KL divergence term, the latent space has to match the prior distribution more closely. This can make the reconstruction loss higher because the model has less room to move. Cutting down on the KL divergence term makes reconstruction better, but the latent space is less regularized as a result.

### 3.7.3 Trade-Off

Balancing these terms is key to the performance of a VAE:

$$L_{\text{total}} = L_{\text{recon}} + \beta \cdot L_{\text{KL}} \quad (3.11)$$

where  $\beta$  controls the trade-off. Higher  $\beta$  increases the emphasis on the KL divergence, potentially leading to higher reconstruction loss.

### 3.7.4 $\beta$ -VAE

The  $\beta$ -VAE (Beta Variational Autoencoder) [29] adds to the standard VAE by adding a hyperparameter  $\beta$  that sets the balance between how well the reconstruction works and how well the latent representations are separated. The loss function has been changed to include the term  $\beta$ :

$$L_{\text{total}} = L_{\text{recon}} + \beta \cdot L_{\text{KL}} \quad (3.12)$$

As  $\beta$  goes up, the KL divergence term becomes more important. This leads to better detangling and a more structured latent space, but it also causes more reconstruction loss.

### 3.7.5 Benefits of $\beta$ -VAE

Better Disentanglement: Latent factors are better separated. Trade-off with Control: You can fine-tune the balance between rebuilding accuracy and latent space regularization by changing  $\beta$ .



## 3.8 Gaussian Mixture Model (GMM)

A Gaussian Mixture Model (GMM) is a type of statistical model that thinks data comes from a mix of different Gaussian distributions, each with its own mean and covariance. In math terms, a GMM is written as:

$$p(x) = \sum_{k=1}^K \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \quad (3.13)$$

The mixed weights are shown by  $\pi_k$  and the Gaussian distribution for the  $k$ th component is shown by  $\mathcal{N}(x|\mu_k, \Sigma_k)$ . The mean is  $\mu_k$  and the covariance is  $\Sigma_k$ . The total number of Gaussian components is shown by  $K$ . Usually, the Expectation-Maximization (EM) algorithm is used to predict the GMM parameters. This algorithm improves the likelihood of the observed data over and over again.

### 3.8.1 Significance of GMM

GMMs are very useful because they can describe data with more than one underlying distribution. By combining several Gaussian distributions, they can correctly show data distributions with more than one peak. Because of this feature, GMMs can be used for grouping, where the model puts data points into groups based on how likely they are to be in those groups. Hard grouping is something that GMMs can do. Each data point has an equal chance of being in each cluster. This isn't the same as hard grouping, which assigns each point to a group.

There is another use for GMMs density estimation. This is the process of finding out the probability density function of a set of data. This feature is useful for many things, like finding "anomalies," which are things that don't fit the predicted distribution and show that something isn't right. In banking, GMMs are used to model risk. In biology, they are used to put gene expressions into groups. And in image processing, they are used to do things like split up images. Because it can work with incomplete data and use what it already knows, the model is accurate and flexible. This makes it more useful in many real-life scenarios.

**Note that:** In multi-dimensional data distribution, we have to deal with mean vector and covariance matrix instead.

## 3.9 Gaussian Multivariate Normal Distribution

The standard Gaussian, commonly referred to as the standard multivariate normal distribution, expands the one-dimensional normal distribution to encompass several dimensions. The concept is characterized by a mean vector and a covariance matrix.

The probability density function (PDF) of a multivariate normal distribution for a random vector  $\mathbf{x}$  in  $\mathbb{R}^d$  can be expressed mathematically as:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu})\right) \quad (3.14)$$

where:

- $\mathbf{x}$  is a  $d$ -dimensional random vector.
- $\boldsymbol{\mu}$  is the mean vector.
- $\Sigma$  is the covariance matrix, which is symmetric and positive-definite.
- $|\Sigma|$  is the determinant of the covariance matrix.
- $\Sigma^{-1}$  is the inverse of the covariance matrix.

Standard Gaussian Distribution

The standard Gaussian distribution is a specific example of the multivariate normal distribution. The mean vector  $\boldsymbol{\mu}$  is zero and the covariance matrix  $\Sigma$  is empty. This is an easier way to look at the probability density function (PDF) for a regular Gaussian distribution with dimensions  $d$ :

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{1}{2}\mathbf{x}^T \mathbf{x}\right) \quad (3.15)$$

In this special case:

- The mean vector  $\boldsymbol{\mu} = \mathbf{0}$ .
- The covariance matrix  $\Sigma = \mathbf{I}$ .

The standard Gaussian distribution is used widely in many domains, including as statistics and machine learning.

### 3.9.1 Prior and Posterior Distributions

**Prior Distribution:** This is what people knew or thought about a measure  $\theta$  before any new data was seen. It is written as  $p(\theta)$ . It's what we know or think we know about the parameter from past experience, expertise, or information that is specific to the subject. The posterior distribution is made by adding the prior distribution and the chance of the data in Bayesian statistics. In math terms, the prior distribution is written as:  $p(\theta)$ .

**Posterior Distribution:** The posterior distribution, shown by  $p(\theta|\mathbf{x})$ , shows how the new information about the parameter  $\theta$  changes the opinions about it after

observing the data  $\mathbf{x}$ . Bayes' theorem is used to mix the prior distribution and the likelihood of the observed data. By using new information, the posterior distribution gives a better estimate of the measure.

Bayes' theorem is given by:

$$p(\theta|\mathbf{x}) = \frac{p(\mathbf{x}|\theta)p(\theta)}{p(\mathbf{x})} \quad (3.16)$$

where:

- $p(\theta|\mathbf{x})$  is the posterior distribution.
- $p(\mathbf{x}|\theta)$  is the likelihood of the observed data given the parameter  $\theta$ .
- $p(\theta)$  is the prior distribution.
- $p(\mathbf{x})$  is the marginal likelihood or evidence, which normalizes the posterior distribution.

The posterior distribution takes into account both the old and new data, which helps us get a better sense of the parameter  $\theta$ .

# Chapter 4

## Our contribution and proposed methods

### 4.1 Solutions to DeepSmote Problems

In this chapter, we present solutions to the challenges faced by DeepSmote, namely overfitting and continuity in the latent space.

### 4.2 Regularized Auto-Encoders

To mitigate the issue of overfitting in DeepSmote, we employ two types of regularized auto-encoders:

#### 4.2.1 Denoising Auto-Encoder

This auto-encoder is designed to reconstruct input data from its noisy version. By doing so, it focuses on learning the essential features of the data, thereby reducing the likelihood of overfitting. The denoising mechanism ensures the model captures the core structure of the data rather than memorizing the noise.

#### 4.2.2 Sparse Auto-Encoder

This variant imposes a sparsity constraint on the activations within the network. By penalizing the activation of too many neurons, the model is encouraged to learn a more compact and generalized representation of the data, which helps in preventing overfitting.

We apply the SMOTE technique in the latent space of these auto-encoders to generate synthetic data. The regularization (denoising and sparsity) helps in maintaining a balanced dataset without overfitting. After generating a balanced dataset,

we retrain the model using a suitable classifier. Detailed experimental results are provided in subsequent chapters.

## 4.3 Ensuring Latent Space Continuity

The continuity issue in the latent space is addressed using Variational Auto-Encoders (VAEs) or Beta VAEs. These models are designed to maintain a continuous latent space through regularization.

### 4.3.1 Training on Imbalanced Data

We first train the VAE/Beta VAE on the imbalanced dataset. The inherent regularization in these models ensures a smooth and continuous latent space.

### 4.3.2 SMOTE in Latent Space

Post training, we apply SMOTE within the latent space to create synthetic latent feature vectors.

### 4.3.3 Decoding

These synthetic latent vectors are decoded back into data samples, generating new instances that help balance the minority classes.

After achieving a balanced dataset, the model is retrained using an appropriate classifier. Detailed experiments and their outcomes are discussed in later chapters.

## 4.4 Using ADASYN for Enhanced Oversampling

In addition to SMOTE, we explore ADASYN (Adaptive Synthetic Sampling) within the latent space of the Denoising Auto-Encoder to improve the representation of minority classes.

### 4.4.1 Training the Auto-Encoder

Initially, the Denoising Auto-Encoder is trained on the imbalanced dataset to produce robust latent representations.

### 4.4.2 Applying ADASYN

ADASYN is then used in the latent space to generate synthetic samples, focusing on creating more challenging samples that enhance minority class representation.

### 4.4.3 Decoding and Classification

The synthetic latent vectors generated by ADASYN are decoded to create new data samples. These samples are used to balance the dataset, which is then employed to retrain the classifier.

This approach aims to improve the classifier’s performance by focusing on difficult-to-classify samples. The effectiveness of this method is validated through experiments and detailed results presented in later chapters.

## 4.5 Weighted Calibration Balancing

In the end, we developed a novel algorithm to balance imbalanced data based on DISTRIBUTION CALIBRATION. We will elaborate on this novel technique called “Weighted Calibration Balancing” to balance each dataset class. Below, I discuss the whole method.

### 4.5.1 Class Partitioning

First, mark the data classes as Majority class, Median class, and Minority class according to their cardinality. The Majority class is the class with the highest cardinality. Now let  $c$  be the cardinality of a particular class and  $m$  be the cardinality of the Majority class. Then if  $10c < m$ , the corresponding class with cardinality  $c$  is defined as the Minority class. The remaining classes are marked as Median classes.

### 4.5.2 Gaussian Mixture Model Partitioning

Next, partition all the classes other than the Minority class into  $n_i$  components of a Gaussian Mixture Model (GMM). Now let  $c_i$  be the cardinality of such a class and suppose that  $s$  be the minimum cardinality among those Minority classes. Then  $n_i$  is defined as follows:

$$n_i = \frac{c_i}{s}$$

After each class’s GMM components, we would have a mean vector and a covariance matrix. After, save the particular mean vector and the covariance matrices of a specific class in two lists and save their indices. Do this according to the increasing class labels and append the lists after the execution of each class.

**Note that all of these classes represented here are the encoded latent representations of them.**

### 4.5.3 Minority Class Processing

For each sample in a Minority class:

1. Find the  $k$ -nearest neighbors from the saved list of mean vectors.
2. Compute  $\mu_{x_c}$  and  $\Sigma_{x_c}$  for each sample using the following formulas:

$$\mu_{x_c} = \frac{\sum_{i=1}^k \left( \frac{1}{d_{x_c,i}} \right) \mu_i + x_c}{1 + \sum_{i=1}^k \left( \frac{1}{d_{x_c,i}} \right)} \quad (4.1)$$

$$\Sigma_{x_c} = \frac{\sum_{i=1}^k \left( \frac{1}{d_{x_c,i}} \right) \Sigma_i + \Sigma_c}{1 + \sum_{i=1}^k \left( \frac{1}{d_{x_c,i}} \right)} \quad (4.2)$$

where  $d_{x_c,i}$  is the distance of  $x_c$  from the  $i$ -th mean vector in its  $k$ -nearest neighbors and  $\Sigma_i$  is the corresponding covariance matrix for the same index  $i$ . The  $\Sigma_c$  is the covariance matrix of a minority class after taking one component from the Gaussian mixture distribution.

### 4.5.4 Synthetic Latent Vectors

Use  $\mu_{x_c}$  and  $\Sigma_{x_c}$  pairs as parameters of a multivariate Cauchy distribution to sample synthetic latent vectors. Sample synthetic latent vectors to balance a particular Minority class with cardinality  $c$ . Let  $k$  number samples in Minority have to be sampled. Then  $k$  can be written as follow:

$$k = \frac{m - c}{c}$$

synthetic latent vectors for each  $\mu_{x_c}, \Sigma_{x_c}$  pair.

### 4.5.5 Median Class Processing

Apply SMOTE in the latent space to generate synthetic latent vectors for the Median classes.

**Decoding:** Finally, the decoder decodes each synthetic latent vector from each imbalanced class to balance the dataset.

## 4.6 Algorithm

---

### Algorithm 4 Weighted Calibration Balancing

---

```

1: procedure WEIGHTEDCALIBRATIONBALANCING
2:   Input: Dataset with multiple classes
3:   Output: Balanced dataset
4:   Step 1: Class Partitioning
5:   Mark classes as Majority, Median, and Minority.
6:   Majority class: Class with the highest number of samples.
7:   Minority classes: Classes where  $10 * \text{cardinality} < \text{Cardinality}$  of the Majority class.
8:   Median classes: All other classes.
9:   Step 2: Gaussian Mixture Model Partitioning
10:  for each class that is not Minority do
11:    Partition class into  $n_i$  components of a Gaussian Mixture Model, where  $n_i =$ 
     $\frac{\text{Cardinality of the class}}{\text{cardinality of the minimal minority class}}$ .
12:    Save the mean vectors and covariance matrices.
13:  end for
14:  Step 3: Minority Class Processing
15:  for each sample  $x_c$  in each Minority class do
16:    Find the  $k$ -nearest neighbors from the saved list of mean vectors.
17:    Compute  $\mu_{x_c}$  and  $\Sigma_{x_c}$  using:

```

$$\mu_{x_c} = \frac{\sum_{i=1}^k \left( \frac{1}{d_{x_c,i}} \right) \mu_i + x_c}{1 + \sum_{i=1}^k \left( \frac{1}{d_{x_c,i}} \right)}$$

$$\Sigma_{x_c} = \frac{\sum_{i=1}^k \left( \frac{1}{d_{x_c,i}} \right) \Sigma_i + \Sigma_c}{1 + \sum_{i=1}^k \left( \frac{1}{d_{x_c,i}} \right)}$$

```

18:     $d_{x_c,i}$  is the distance of  $x_c$  from the  $i$ -th mean vector in its  $k$ -nearest neighbors.
19:     $\Sigma_i$  is the corresponding covariance matrix for the same index  $i$ .
20:     $\Sigma_c$  is the covariance matrix of a minority class after taking one component from the
    Gaussian mixture distribution.
21:  end for
22:  Step 4: Synthetic Latent Vectors Generation
23:  for each Minority class do
24:    for each  $\mu_{x_c}, \Sigma_{x_c}$  pair do
25:      Sample synthetic latent vectors using multivariate Cauchy distribution.
26:      Generate  $\frac{(m-c)}{c}$  synthetic latent vectors for each  $\mu_{x_c}, \Sigma_{x_c}$  pair.
27:    end for
28:  end for
29:  Step 5: Median Class Processing
30:  Apply SMOTE in the latent space to generate synthetic latent vectors for the Median
    classes.
31:  Step 6: Decoding
32:  Decode the synthetic latent vectors to balance the dataset.
33: end procedure

```

---



## 4.7 Overall Algorithm

The ”**Weighted Calibration Balancing**” algorithm is designed to address class imbalance in datasets by generating synthetic samples. It begins by classifying data into Majority, Median, and Minority classes based on their sample sizes. Majority classes have the most samples, while Minority classes have fewer than one-tenth the samples of the Majority class.

Next, it applies Gaussian Mixture Models (GMM) to partition Majority and Median classes, saving their mean vectors and covariance matrices. For each sample in the Minority class, the algorithm computes new mean vectors and covariance matrices by considering the nearest neighbors from the saved mean vectors.

Using these parameters, the algorithm generates synthetic latent vectors via a multivariate Cauchy distribution for Minority classes. For Median classes, it applies SMOTE in the latent space to create synthetic vectors. Finally, it decodes these synthetic latent vectors to achieve a balanced dataset.

# Chapter 5

## Experiments and Results

This chapter presents the results of our extensive experiments and our observations and interpretation of those results. We have mainly worked with two datasets ( Imbalanced MNIST and Imbalanced CIFAR10 ) and compared the performance on some key metrics with existing methods known in the literature.

### 5.1 Data Overview

This set of data from MNIST includes handwritten pictures of numbers. This set has 60,000 pictures for training and 10,000 images for testing. With a resolution of 28x28 pixels, each picture is grayscale, and there are 10 different classes that show the numbers 0 through 9. To make the dataset less balanced, a certain number of samples were picked at random from the training set's respective classes. Using the MNIST dataset, the numbers of unbalanced samples in each class were [4000, 2000, 1000, 750, 500, 350, 200, 100, 60, 40]. Therefore, there was a mismatch of 100:1, meaning that the largest minority class had 100 times fewer samples than the smallest majority class.

The CIFAR-10 dataset is a widely used benchmark dataset in computer vision and machine learning. It is designed for fine-grained object recognition tasks. CIFAR-10 consists of 10 classes, each containing 6000 images, resulting in a total of 60,000 images. The dataset is divided into a training set of 50,000 images and a test set of 10,000 images. Each image in CIFAR-10 has a fixed size of 32x32 pixels. The images are RGB (color) images, meaning they have three color channels (red, green, and blue) for each pixel. Using the CIFAR10 dataset, the numbers of unbalanced samples in each class were [5000, 2500, 1000, 1000, 600, 400, 300, 150, 90, 50]. Therefore, there was a mismatch of 100:1, meaning that the largest minority class had 100 times fewer samples than the smallest majority class.

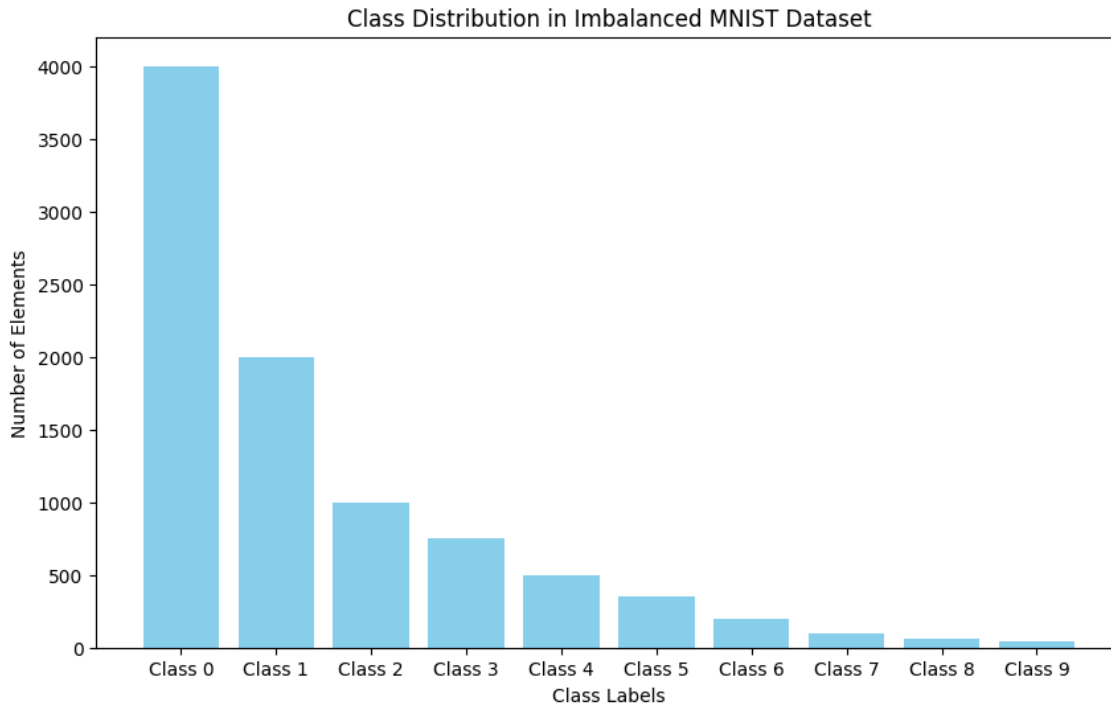


Figure 5.1: Class distribution of data points in the Imbalanced MNIST dataset

## 5.2 Evaluation Metrics

To check the performances of imbalanced MNIST and CIFAR10 datasets. The performance metrics used are recall(macro average), Precision(macro average), F1 score(macro average). These metrics are very popular in the literature.

### 5.2.1 Recall (Macro Average)

Recall measures the ability of the model to correctly identify all relevant instances within a class. The macro average recall is computed by taking the average recall of each class, treating all classes equally, regardless of their size. It is defined as:

$$\text{Recall (macro average)} = \frac{1}{N} \sum_{i=1}^N \frac{\text{True Positives}_i}{\text{True Positives}_i + \text{False Negatives}_i}$$

where  $N$  is the number of classes.

### 5.2.2 Precision (Macro Average)

Precision indicates the accuracy of the positive predictions made by the model. The macro average precision is the average of precision values for each class, giving equal

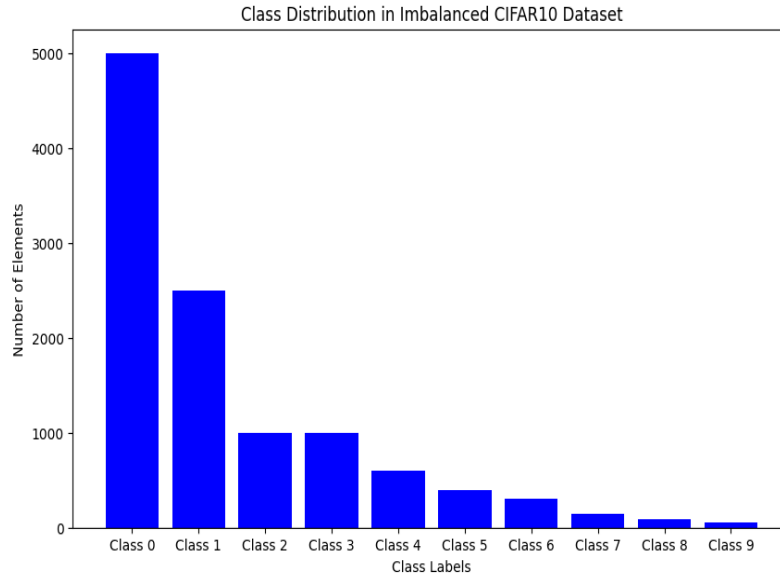


Figure 5.2: Class distribution of the Imbalanced CIFAR10 dataset

weight to all classes. It is calculated as:

$$\text{Precision (macro average)} = \frac{1}{N} \sum_{i=1}^N \frac{\text{True Positives}_i}{\text{True Positives}_i + \text{False Positives}_i}$$

### 5.2.3 F1 Score (Macro Average)

The F1 score is the harmonic mean of precision and recall, providing a single measure of a test’s accuracy. The macro average F1 score is the average of F1 scores for each class, ensuring equal consideration for all classes. It is given by:

$$\text{F1 Score (macro average)} = \frac{1}{N} \sum_{i=1}^N \frac{2 \times \text{Precision}_i \times \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}$$

These metrics are particularly useful in imbalanced datasets to provide a balanced view of model performance across all classes.

## 5.3 Training Details

We designed our auto-encoders using convolutional layers for the encoder, followed by a latent space representation, and inverse layers for the decoder. The encoder architecture comprises four convolutional layers, each followed by batch normalization and the LeakyReLU activation function. The dimensions of the hidden layers are set to 64. The final layer is a linear layer, resulting in a latent dimension of 300.

For the decoder, we used four convolutional transpose layers, mirroring the structure of the encoder. These layers also utilize batch normalization and the Rectified

Linear Unit (ReLU) activation function, except for the final layer, which employs the Tanh activation function.

The models were trained for a duration of 50 to 350 epochs, stopping once the training loss plateaued. The training was conducted using the Adam optimizer with a learning rate of 0.0003. All implementations, including DeepSMOTE, were carried out using the PyTorch framework.

Since MNIST’s digits are gray-scale images we use a Multi-Layer-Perceptron(MLP) with two hidden layers for the classification task of imbalanced MNIST data set.

For the classification purpos of CIFAR10 data, we take a ResNet-18[?] architecture with Adam optimiser and a learning rate of  $1e^{-1}$ . ResNet-18 is a part of the ResNet (Residual Network) family of Convolutional Neural Networks. The basic building block of this family of neural networks is the ‘residual block’ which has skip connections across layers to promote better flow of information between layers. Some modifications are made to adapt the network to the CIFAR data image size of 32x32 pixels, like the initial 7x7 convolutional layer with stride 2 and max pooling is replaced with a 3x3 convolutional layer with stride 1 and no max pooling. ResNet-18 has been shown to achieve high accuracy on CIFAR-10 and is widely used as a baseline model for benchmarking image classification performance on this dataset.

## 5.4 Results

### 5.4.1 MNIST

The results, detailed in Table 5.1, demonstrate the superior performance of regularized auto-encoders compared to the Baseline and DeepSmote methods. The success of regularized auto-encoders can be attributed to their ability to prevent overfitting to the training data. Specifically, DeepSparseSmote showed better performance than DeepSmote, while DeepDenoisingSmote outperformed all other methods.

Additionally, BetaVAE also surpassed DeepSmote in terms of performance. However, it was observed that BetaVAE produced a higher reconstruction loss during training. This increased reconstruction loss is a consequence of the trade-off between disentanglement and reconstruction accuracy. As a result, BetaVAE sometimes generated blurrier images, which occasionally led to confusion for the classifier, particularly in the case of imbalanced MNIST data.

These findings underline the importance of regularization in auto-encoders for handling imbalanced datasets. Regularized auto-encoders not only improve the quality of generated samples but also enhance the overall performance of the model by ensuring it does not overfit to the training data. The trade-off observed in BetaVAE highlights the challenges of balancing different aspects of model performance, such

Table 5.1: Comparison of DeepSmote results with other results on Imbalanced MNIST

Methods	Recall	Precision	F1 score
DeepSmote	0.8286	0.8363	0.8251
$\beta$ VAE, $\beta=1$	0.8608	0.0.8713	0.8533
$\beta$ VAE, $\beta=2$	0.8243	0.8462	0.8164
DeepSparseSmote	0.8546	0.8628	0.8529
DeepDenoisingSmote	<b>0.8637</b>	<b>0.8715</b>	<b>0.8612</b>

Table 5.2: Comparison of DeepSmote results with other results including Weighted Calibration in Denoising Auto-encoder on Imbalanced MNIST

Methods	Recall	Precision	F1 score
DeepSmote	0.8286	0.8363	0.8251
$\beta$ VAE, $\beta=1$	0.8608	0.0.8713	0.8533
$\beta$ VAE, $\beta=2$	0.8243	0.8462	0.8164
DeepDenoising with Weighted Calibration	<b>0.9230</b>	<b>0.9202</b>	<b>0.9216</b>

as disentanglement and reconstruction accuracy.

From Table 5.2 we can be assured that our novel "Weighted Calibration Balancing" method beats all other methods. The "Weighted Calibration Balancing" method works really well because it carefully changes the weight of each synthetic sample. This makes sure that minority classes are better reflected without adding too much noise to the model. This calibration works really well at keeping the quality and variety of the samples that are made, which makes the classification model more accurate and fair.

### 5.4.2 CIFAR 10

From the Table 5.3 it is clear that DeepDenoisingSmote beats others here also. For the same reason in the case of MNIST. From From the Table 5.4 it is observed that our DeepDenoising and Weighted Calibration also performs better than DeepSmote. Finally from the Table 5.4 we can say that DeepAdasyn outperforms all.

DeepADASYN outperformed other methods on the CIFAR-10 dataset, which contains complex color images with diverse objects, by effectively addressing its unique challenges through adaptive synthetic sampling. Unlike traditional SMOTE, ADASYN focuses on generating more synthetic samples for the minority class instances that are harder to learn, ensuring the classifier pays more attention to these challenging examples. This adaptive approach promotes greater diversity and a more balanced training set by targeting underrepresented and difficult-to-classify instances, enhancing the classifier’s ability to generalize. Furthermore, integrating deep learning models for feature extraction before applying ADASYN captures intricate patterns in the data, producing higher-quality synthetic samples that are especially beneficial for high-dimensional datasets like CIFAR-10.

The Weighted Calibration Balancing algorithm works here but not significantly. The CIFAR-10 dataset presented the greatest challenge for deep oversampling algorithms. We believe this is due to the diverse nature of the CIFAR-10 classes, which do not share common attributes, unlike datasets such as MNIST where all classes represent different digits. For instance, CIFAR-10 includes varied categories such as cats, dogs, airplanes, and frogs. This diversity means that models cannot easily transfer knowledge from the majority class, which has more samples, to the minority class, which has fewer samples. Additionally, we observed that there is significant feature overlap among some of the CIFAR-10 classes, further complicating the task for these models.

Table 5.3: Comparison of DeepSmote results with proposed method’s results on Imbalanced CIFAR10

Methods	Recall	Precision	F1 score
DeepSmote	0.4443	0.3547	0.3945
$\beta$ VAE, $\beta=1$	0.4304	0.3370	0.3780
$\beta$ VAE, $\beta=2$	0.4438	0.3549	0.3944
DeepSparseSmote	0.4651	0.3708	0.4126
DeepDenoisingSmote	<b>0.4725</b>	<b>0.3844</b>	<b>0.4239</b>

Table 5.4: DeepSmote results with proposed method’s results on Imbalanced CIFAR10

Methods	Recall	Precision	F1 score
DeepSmote	0.4443	0.3547	0.3945
$\beta$ VAE, $\beta=1$	0.4304	0.3370	0.3780
$\beta$ VAE, $\beta=2$	0.4438	0.3549	0.3944
DeepDenoising with Weighted Calibration	<b>0.4468</b>	<b>0.3709</b>	<b>0.4053</b>

Table 5.5: Comparison DeepSmote results with proposed method’s results on Imbalanced CIFAR10

Methods	Recall	Precision	F1 score
DeepSmote	0.4443	0.3547	0.3945
$\beta$ VAE, $\beta=1$	0.4304	0.3370	0.3780
$\beta$ VAE, $\beta=2$	0.4438	0.3549	0.3944
DeepDenoising with Weighted Calibration	0.4468	0.3709	0.4053
DeepAdasyn	<b>0.5048</b>	<b>0.4225</b>	<b>0.4600</b>



# Chapter 6

## Conclusion and Future work

Using Deep Denoising SMOTE for oversampling gave us better results than other auto-encoders in terms of macro-average recall, accuracy, and F1 score when we tested it on unbalanced MNIST data. This achievement shows that regularization is a good way to stop overfitting. Also, our new "Weighted Calibration Balancing" method worked 4 percent better than Deep Denoising SMOTE when used in the Deep Denoising Auto-Encoder scheme. This shows how effective this method is.

When compared to normal Deep SMOTE, the Beta VAE also did better. By using SMOTE in the Beta VAE's latent space, we were able to make samples with more variation while keeping the latent space continuous and smooth. This is very important for getting true and varied data samples. It is important to keep in mind, though, that Beta VAE's gains in disentanglement and variability often come at the cost of more reconstruction loss. This makes samples less clear, making it hard for classifiers to correctly put them into groups.

We had the same problems when we used unbalanced CIFAR-10 data. The rebuilding loss in Beta VAE made it hard to classify the samples because they were not as different from each other.

To address these challenges and further improve our methods, we plan to explore advanced auto-encoders such as Beta-TCVAE and Wasserstein Autoencoders (WAE). These models aim to maintain a continuous latent space while ensuring smooth and high-quality sample generation. We will apply our current techniques to these models and investigate whether they can yield even better results. Aslo note that literature shows Beta-TCVAE can get better disentanglement managing the reconstruction loss.

Additionally, we will continue experimenting with the latent spaces of these auto-encoders, applying various oversampling techniques to discover new methods that can further enhance performance. Specifically, we will focus on identifying novel techniques that can improve both the quality and diversity of generated samples

while minimizing reconstruction loss.

We also intend to refine and develop better strategies for Deep SMOTE. By experimenting with different configurations and incorporating additional regularization methods, we aim to enhance its capability to handle highly imbalanced datasets more effectively.

In conclusion, our work demonstrates the potential of combining regularization techniques with advanced auto-encoders to address the challenges of imbalanced data. The promising results from our "Weighted Calibration Balancing" technique encourage us to continue exploring and innovating in this area, striving for improved methodologies that provide robust solutions to imbalanced data problems.

# Bibliography

- [1] D. Dablain, B. Krawczyk, and N. V. Chawla, “Deepsmote: Fusing deep learning and smote for imbalanced data,” *IEEE Transactions on Neural Networks and Learning Systems*, 2022.
- [2] B. Krawczyk, “Learning from imbalanced data: open challenges and future directions,” *Progress in Artificial Intelligence*, vol. 5, no. 4, pp. 221–232, 2016.
- [3] A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk, and F. Herrera, *Learning from imbalanced data sets*, vol. 10. Springer, 2018.
- [4] U. Fiore, A. De Santis, F. Perla, P. Zanetti, and F. Palmieri, “Using generative adversarial networks for improving classification effectiveness in credit card fraud detection,” *Information Sciences*, vol. 479, pp. 448–455, 2019.
- [5] F. Bao, Y. Deng, Y. Kong, Z. Ren, J. Suo, and Q. Dai, “Learning deep landmarks for imbalanced classification,” *IEEE transactions on neural networks and learning systems*, vol. 31, no. 8, pp. 2691–2704, 2019.
- [6] X.-Y. Jing, X. Zhang, X. Zhu, F. Wu, X. You, Y. Gao, S. Shan, and J.-Y. Yang, “Multiset feature learning for highly imbalanced data classification,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 1, pp. 139–156, 2019.
- [7] L. A. Bugnon, C. Yones, D. H. Milone, and G. Stegmayer, “Deep neural architectures for highly imbalanced data in bioinformatics,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 8, pp. 2857–2867, 2019.
- [8] Z. Wang, X. Ye, C. Wang, Y. Wu, C. Wang, and K. Liang, “Rsdne: Exploring relaxed similarity and dissimilarity from completely-imbalanced labels for network embedding,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.
- [9] 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.

- [10] S. Das, S. S. Mullick, and I. Zelinka, “On supervised class-imbalanced learning: An updated perspective and some key challenges,” *IEEE Transactions on Artificial Intelligence*, vol. 3, no. 6, pp. 973–993, 2022.
- [11] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans,” *Advances in neural information processing systems*, vol. 29, 2016.
- [12] G. Mariani, F. Scheidegger, R. Istrate, C. Bekas, and C. Malossi, “Bagan: Data augmentation with balancing gan,” *arXiv preprint arXiv:1803.09655*, 2018.
- [13] S. S. Mullick, S. Datta, and S. Das, “Generative adversarial minority oversampling,” in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1695–1704, 2019.
- [14] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Deep learning with denoising autoencoders,” *Journal of Machine Learning*, vol. 27, pp. 49–50, 2008.
- [15] A. Ng *et al.*, “Sparse autoencoder,” *CS294A Lecture notes*, vol. 72, no. 2011, pp. 1–19, 2011.
- [16] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.
- [17] H. He, Y. Bai, E. A. Garcia, and S. Li, “Adasyn: Adaptive synthetic sampling approach for imbalanced learning,” in *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, pp. 1322–1328, Ieee, 2008.
- [18] N. Japkowicz *et al.*, “Learning from imbalanced data sets: a comparison of various strategies,” in *AAAI workshop on learning from imbalanced data sets*, vol. 68, pp. 10–15, AAAI Press, Menlo Park, 2000.
- [19] D. L. Wilson, “Asymptotic properties of nearest neighbor rules using edited data,” *IEEE Transactions on Systems, Man, and Cybernetics*, no. 3, pp. 408–421, 1972.
- [20] I. Tomek, “An experiment with the edited nearest-neighbor rule.,” 1976.
- [21] G. Douzas, F. Bacao, and F. Last, “Improving imbalanced learning through a heuristic oversampling method based on k-means and smote,” *Information sciences*, vol. 465, pp. 1–20, 2018.

- [22] H. Han, W.-Y. Wang, and B.-H. Mao, “Borderline-smote: a new over-sampling method in imbalanced data sets learning,” in *International conference on intelligent computing*, pp. 878–887, Springer, 2005.
- [23] M. Bader-El-Den, E. Teitei, and T. Perry, “Biased random forest for dealing with the class imbalance problem,” *IEEE transactions on neural networks and learning systems*, vol. 30, no. 7, pp. 2163–2172, 2018.
- [24] T.-Y. Liu, “Easyensemble and feature selection for imbalance data sets,” in *2009 international joint conference on bioinformatics, systems biology and intelligent computing*, pp. 517–520, IEEE, 2009.
- [25] F. Li, X. Zhang, X. Zhang, C. Du, Y. Xu, and Y.-C. Tian, “Cost-sensitive and hybrid-attribute measure multi-decision tree over imbalanced data sets,” *Information Sciences*, vol. 422, pp. 242–256, 2018.
- [26] M. Buda, A. Maki, and M. A. Mazurowski, “A systematic study of the class imbalance problem in convolutional neural networks,” *Neural networks*, vol. 106, pp. 249–259, 2018.
- [27] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection,” in *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.
- [28] S. Yang, L. Liu, and M. Xu, “Free lunch for few-shot learning: Distribution calibration,” *arXiv preprint arXiv:2101.06395*, 2021.
- [29] C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner, “Understanding disentangling in  $\beta$ -vae,” *arXiv preprint arXiv:1804.03599*, 2018.
- [30] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap, “Db-smote: density-based synthetic minority over-sampling technique,” *Applied Intelligence*, vol. 36, pp. 664–684, 2012.
- [31] L. Ma and S. Fan, “Cure-smote algorithm and hybrid algorithm for feature selection and parameter optimization based on random forests,” *BMC bioinformatics*, vol. 18, pp. 1–18, 2017.
- [32] R. T. Chen, X. Li, R. B. Grosse, and D. K. Duvenaud, “Isolating sources of disentanglement in variational autoencoders,” *Advances in neural information processing systems*, vol. 31, 2018.
- [33] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf, “Wasserstein auto-encoders,” *arXiv preprint arXiv:1711.01558*, 2017.

- [34] K. Boonchuay, K. Sinapiromsaran, and C. Lursinsap, “Decision tree induction based on minority entropy for the class imbalance problem,” *Pattern Analysis and Applications*, vol. 20, pp. 769–782, 2017.
- [35] S. Datta, S. Nag, and S. Das, “Boosting with lexicographic programming: Addressing class imbalance without cost tuning,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 5, pp. 883–897, 2019.
- [36] X. Zhang, Y. Zhuang, W. Wang, and W. Pedrycz, “Transfer boosting with synthetic instances for class imbalanced object recognition,” *IEEE transactions on cybernetics*, vol. 48, no. 1, pp. 357–370, 2016.
- [37] M. Sokolova and G. Lapalme, “A systematic analysis of performance measures for classification tasks,” *Information processing & management*, vol. 45, no. 4, pp. 427–437, 2009.