MASTERS THESIS

Addressing Class-Imbalance using Generative Adversarial Networks

by

Aatif Nisar Dar

Under the Supervision of

Dr. Reshma Rastogi

Submitted in partial fulfillment of the requirements
for the award of the degree of
Master of Science in Computer Science

to the

SAU
SOUTH ASIAN UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SOUTH ASIAN UNIVERSITY, NEW DELHI - 110021, INDIA
May, 2022

MASTERS THESIS

Addressing Class-Imbalance using Generative Adversarial Networks

by

Aatif Nisar Dar
SAU/CS(M)/2020/01

Under the Supervision of

Dr. Reshma Rastogi

Submitted in partial fulfillment of the requirements for the award of the degree of

Master of Science in Computer Science

to the

Department of Computer Science
Faculty of Mathematics and Computer Science
South Asian University, New Delhi - 110021, India
May, 2022

Declaration

I hereby declare that the thesis entitled "**Addressing Class-Imbalance using Generative Adversarial Networks**" being submitted to the Department of Computer Science, Faculty of Mathematics and Computer Science, South Asian University, New Delhi in partial fulfillment of the requirements for the award of the degree of **Master of Science in Computer Science** contains the original work carried out by me under the supervision of Dr. Reshma Rastogi. The research work reported in this thesis is original and has not been submitted either in part or full to any university or institution for the award of any degree or diploma.

Aatif

Name: Aatif Nisar Dar

Enrollment No: SAU/CS(M)/2020/01

Abstract

A common problem while training deep learning models is the lack of labeled training data. Often real-life datasets such as multilabel datasets suffer from class Imbalance problem, which is inescapable. The limited minority data may not be sufficient for efficient learning and often can cause the networks to overfit. This dissertation considers the potential application of Generative Adversarial Networks (GANs) to restore balance in imbalanced multilabel datasets. We will generate new data for the minority labels and use multilabel learning algorithms to handle multilabel data. We compare our model with MLSMOTE to validate the effectiveness of our model. Experiments over six real datasets using five different multilabel learning algorithms and seven evaluation measures show that our strategy of resampling the multilabel data constantly outperforms MLSMOTE. Results indicate that imbalance in multilabel datasets is reduced in a classifier-independent way; that is, the classifier should have a deplorable influence on the effectiveness of the resampling strategy. While generating new samples from GAN architecture, we use representative class samples to represent each class distribution which further reduces the training time.

Acknowledgement

I would first like to thank my supervisor, Dr. Reshma Rastogi, for her endless optimism concerning this work, enthusiasm, and encouragement. This research would not have been possible without her constant support and the interesting and insightful discussions I had with her. Her way of guidance always encouraged me to think in-depth about my research topic. Through our meetings, I got a space to think, understand and argue my point of view, and her encouragement kept me motivated throughout the process. From her, I have learned to think critically, select problems, solve them, and present their solutions. I admire all her contributions of time and ideas to make my research work more productive and stimulating. Her enthusiasm for her research was contagious and motivational for me, even during challenging times in the M.Sc. pursuit. I would also like to thank all the faculty members, my seniors, and peers for creating an environment of learning which is crucial for any research. The standards set by the Faculty of Computer Science members, both professors and students, motivated me to work harder. Lastly, I would like to thank my lab seniors, Mr. Sanjay Kumar and Mr. Sambhav Jain, for their continuous support and guidance.

Aatif Nisar Dar
SAU/CS(M)/2020/01
May, 2022

CERTIFICATE

This is to certify that the thesis entitled **"Addressing Class-Imbalance using Generative Adversarial Networks"** submitted by Aatif Nisar Dar to the Department of Computer Science, Faculty of Mathematics and Computer Science, South Asian University, New Delhi in partial fulfilment of the requirements for the award of the degree of Master of Science in Computer Science, is a record of the bonafide research work carried out by him under my supervision and guidance. The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree or diploma.

Reshma
29/5/22

Dr. Reshma Rastogi
(Supervisor)
Department of Computer Science

Faculty of Mathematics and Computer Science.
South Asian University, Delhi, India.

Contents

Declaration **iv**

Abstract **v**

Acknowledgement **vi**

1 Introduction **1**

1.1 Overview 1

1.2 Dissertation Outline 3

2 Related Work **4**

2.1 Generative Adversarial Networks 4

2.2 Data Augmentation 6

2.3 Imbalance in Multilabel Classification 6

3 Deep Convolutional Generative Adversarial Networks (DCGAN) **8**

3.1 Improved Techniques for Training GANs 10

3.2 DCGAN (Deep Convolutional Generative Adversarial Network) 11

3.2.1 StyleGAN 13

3.2.2 SMIT (Stochastic Multi-Label Image-to-Image Translation) . . . 14

3.3 SAGAN (Self Attention Generative Adversarial Network) 15

3.4 Partial Label Learning 18

3.4.1 PLLGAN (Partial Label Learning via GANs) 19

3.4.2 PMLGAN (Partial Multi-Label Learning via GANs) 19

4 MLGAN **22**

4.1 Measuring Imbalance 23

4.2 Generative Adversarial Network (GAN) Architecture 24

4.3 Proposed Approach 25

5 Experimental Setup and Results **28**

5.1 Experimental Setup 28

5.1.1 Datasets 28

5.1.2 Classifiers 30

5.1.3 Evaluation Metrics 32

5.2 Experimental Results 34

5.2.1 Effect of K in MLSMOTE 40

6 Conclusion and Future Works **41**

References **42**

Chapter 1

Introduction

1.1 Overview

In traditional classification, a dataset is composed of a set of input features and corresponding output label or class. In multilabel learning, an instance can belong to more than one label, and the total number of different labels can be huge. For example, In a movie recommendation system, a movie can belong to both horror and action at the same time. Multilabel data is helpful to process many practical problems effectively as it inherits rich semantics. Let $X = R^d$ denote the domain of input feature space and $Y = \{l_1, ..., l_q\}$ denote a finite set of labels. Let $D = \{(x_i, y_i), 1 \leq i \leq n\}$ be a multi-label training set containing n instances. Here, $x_i \in X$ is the feature vector and $y_i \in Y$ is the label vector. y_{ij} is the jth element of y_i and $y_{ij} = 0$ or 1 denoting the absence or presence of l_j with the ith instance. The goal of multilabel learning is to learn a mapping function $f : X \rightarrow Y_q$, which is able to correctly predict the label vector of unseen instances. Multilabel learning tasks are ubiquitous in real-world problems and have gained a lot of attention in a variety of domains such as images, text, audio [1]. The main focus of multilabel learning in its early stage was on text categorization [2]. Recently, multilabel learning has attracted a lot of attention from machine learning and related communities. Multilabel datasets are being widely applied to a lot of real-world problems like images[3], bioinformatics[4], tag recommendation[5], etc. Recently, much research has been done on multilabel learning, and class imbalance has been recognized as a critical challenge. To overcome this class imbalance problem, research has primarily focused on either adding or removing instances [6]. Oversampling approaches based on cloning have also been proposed in [6, 7]. Charte et al. [8] proposed MLSMOTE that generates new instances by picking up random samples among the nearest neighbors of one another.

Deep Learning models demand high-quality datasets to train on the algorithm. By high-quality, we not only mean that the data should be extensive, but it should cover each class almost equally. The real-world datasets suffer from several forms of imbalance which is a challenge in Deep Learning. The limited minority data may not be sufficient and often cause the networks to overfit. In such cases, it is possible to create new entries in the minority data, which increases the representation of the minority class and helps to avoid overfitting.

In this dissertation, Generative Adversarial Network (GAN) [9] is used to create synthetic samples for the minority class to overcome the class Imbalance problem. GAN employs two Neural Networks; Generator and the Discriminator. We will investigate whether incorporating GANs in multilabel datasets as a data augmentation technique can improve the performance of our model. Also, we will check whether we are able to resample the multilabel dataset in a classifier-independent way. We will use the architecture of CTGAN (Conditional Tabular Generative Adversarial Network) [10] in this study. CTGAN model employs 1) Conditional Generator, so that Generator can generate synthetic data conditioned on discrete columns. 2) Training-by-sample technique helps the model explore all possible values evenly by properly sampling the conditional vector and training data. We check imbalance in all the labels using IRLbl (Imbalance ratio per label) [11, 12]. The labels where there is an imbalance are called minority labels, and those labels with no imbalance are called majority labels. The main objective of this dissertation is to produce synthetic data associated with minority labels. The Generator generates new data for the minority labels, and the multilabel learning problem is tackled by a simple categorization of multilabel learning algorithms[13, 14, 15, 16, 17] discussed later in this dissertation. CVIR (Coefficient of variation of IRLbl)[12] will be used to know the overall imbalance in the multilabel dataset.

Multilabel dataset is partitioned into training set T_{train} and test set T_{test} using k-fold cross-validation technique, where the value of k is set to 5. Imbalance is checked on T_{train}, and the required samples are synthesized and concatenated with T_{train}. After training the classifier on T_{train}, we evaluate the efficacy of our model on T_{test}. We will compare the results obtained from our technique with the results obtained from MLSMOTE [8] method. Experiments over six real datasets using five different multilabel learning algorithms and seven evaluation measures[18, 19] show that our strategy of resampling the multilabel data constantly outperforms MLSMOTE.

1.2 Dissertation Outline

The rest of this dissertation is organized as follows. Related work on data augmentation in multilabel learning and generative adversarial network is reviewed in section **2**. In section **3**, DCGAN (Deep Convolutional Generative Adversarial Networks) will be discussed. In section **4**, we will discuss our approach; that is, MLGAN (Multilabel Generative Adversarial Networks), followed by experimental setup and experimental results in section **5**. Conclusion and future work regarding our study will be discussed in **6**.

Related work2

DCGAN (Deep Convolutional Generative Adversarial Network)3

Multilabel Generative Adversarial Networks (MLGAN) 4

Experimental setup and Experimental results 5

Conclusion and Future Work 6

Chapter 2

Related Work

2.1 Generative Adversarial Networks

Goodfellow et al. [9] introduced Generative Adversarial Networks (GANs). GANs are a type of generative model pitted against an adversary. In GANs, we simultaneously train two models: Generator and Discriminator. The Generator and the Discriminator are deep Neural Networks competing against each other. A generative model G to be trained on training data X sampled from true distribution D is the one which, given some standard random distribution z, produces a distribution D' close to D according to some closeness metric. Generator creates new instances $p_g(z)$ by capturing the data distribution. At the same time, the Discriminator estimates the probability that a sample came from the training data $p_{data}(X)$ rather than Generator. Both Generator and Discriminator are trained using the backpropagation. After several steps of training, Generator and Discriminator will reach the optimal point where $p_g(z) = p_{data}(X)$. At this optimal condition, the Discriminator will not differentiate whether the sample comes from the original or generated distribution.

GANs have gained a lot of attention over recent years. Romero et al. [20] introduced an image-to-image translation technique SMIT (Stochastic Multi-Label Image-to-Image Translation). In SMIT, using a single Generator, authors propose a joint framework of multimodality, unpaired datasets, and multiple attributes to conditionally produce countless and fake images, holding the underlying characteristic of the source image. Cao et al. [21] proposed Human Pose estimation using SAGAN (Self-Attention GAN). The Self-Attention module in SAGAN helps in capturing long-range dependencies. Park et al. [22] introduced table-GAN to synthesize tabular data using GANs. A classifier neural network was added apart from the Generator and Discriminator to increase the semantic integrity. Lei Xu et al. [23] introduced TGAN (Tabular Generative Adversarial Network). TGAN uses a Recurrent Neural network, while table-GAN uses a convolutional network. TGAN learns the marginal distribution of each column by minimizing KL divergence, while table-GAN minimizes the cross-entropy loss. Lei Xu et al. [10] introduced CTGAN (Conditional Tabular Generative Adversarial Network). In CTGAN, a mix of activation functions are used at the output of the network, tanh and gumble softmax, to generate discrete and continuous values. This study uses CTGAN as the base architecture to generate the samples in multilabel datasets. Y Zhang et al.[24] introduced PLLGAN (Partial Label Learning via Generative Adversarial Nets). PLLGAN comprises two components; 1) Generator and 2) Discriminator. The Generator adopts the idea of CGAN (Conditional GAN). The candidate label set (candidate to be true) is given to the Generator to generate samples similar to the real samples. In [25], the authors introduced PMLGAN (Partial Multilabel learning GAN). In PMLGAN, along with the Discriminator and the Generator, a disambiguation network to identify irrelevant labels and a predictive network to map the training instances to their disambiguated label vectors are also employed. Cao et al. in [26] introduced Human Pose estimation using self attended GANs. In this study, both the Discriminator and the Generator used Hourglass networks. The Hourglass networks are fully connected with residual blocks and convolution-deconvolution architecture. In the Generator, 4-stack hourglass networks are used. The Generator generates heatmaps that indicate the confidence score for all the body joint key points at every location. The Discriminator uses a 1-stack hourglass network. It reconstructs both the predicted heatmaps and the ground truth heatmaps and distinguishes real from fake.

2.2 Data Augmentation

Data augmentation was initially done for image classification tasks. Traditional data augmentation techniques for image classification were achieved by flipping, scaling, cropping, padding, rotation, or adding a small amount of noise to the original image. These small mutations were applied to the machine learning tasks to increase the training data by creating new examples. In [27], a simple yet effective data augmentation technique for the image classification task was proposed. In this paper, two randomly chosen examples (X_i, y_i) and (X_j, y_j) are picked. The new example is randomly decided among two choices; $(\frac{X_i, X_j}{2}, y_i)$ or $(\frac{X_i, X_j}{2}, y_j)$. Oversampling and undersampling are techniques used to re-sample imbalanced class distributions. An intelligent oversampling technique called SMOTE (Synthetic Minority Over-sampling Technique), discussed in [28], creates new instances by interpolating new points from existing instances via K-NN (K-Nearest Neighbors). Maayan et al. [29] introduced a GAN-based data augmentation technique for liver lesion classification. In this paper, the authors suggested a combination of standard image perturbation to create a larger dataset of CT images and synthetic labeled liver lesion generation using DCGAN (Deep Convolutional Generative Adversarial Network) from CT images. The combined standard and synthetic augmentation are finally used to train a lesion classifier. This improved classification performance by 7.1% specificity and 4% sensitivity when compared using classical augmentation techniques.

2.3 Imbalance in Multilabel Classification

Multilabel datasets and classification methods have rapidly become more common in recent years. Multilabel datasets suffer from an imbalance problem which prevents the model from predicting the minority class and suffers from overfitting. There are two objectives when producing synthetic data in multilabel datasets: High Data Utility and Low Disclosure Risk [30]. Reducing disclosure risk comes at a cost in utility. So while augmenting, a balance in trade-off between the two has to be maintained.

Much research is being done to overcome the imbalance in binary and multiclass datasets. Handling imbalance in multilabel cases can be an issue as each instance may be associated with multiple labels. Existing oversampling and undersampling methods assumed an instance to be associated with a single label. Two preprocessing measures aimed at reducing the imbalance in multilabel tasks were introduced by Charte et al. [6]. In the Undersampling method, 25% of random samples of majority labelsets were deleted, and in Oversampling method, 25% of random samples from minority label sets were cloned. The proposal in [31] is an undersampling approach used to improve the classification performance in text categorization. Du et al. [7] proposed random undersampling and random oversampling algorithms while using methods to divide the instances into minority and majority groups. Charte et al.[8] introduced MLSMOTE (Multilabel Synthetic Minority Over-Sampling Technique). MLSMOTE takes all the instances from minority labels and then picks random samples among the nearest neighbors of each one. After selecting the neighbors, interpolation techniques help obtain the set of features, and finally, a synthetic label set is generated for the new instance. In this dissertation, we will compare our model with MLSMOTE. We statistically evaluate both techniques on seven multilabel datasets.

Chapter 3

Deep Convolutional Generative Adversarial Networks (DCGAN)

GANs are comprised of two neural networks, Discriminator and Generator, competing for one against the other. A generative model that captures the data distribution and a discriminative model estimates the probability that a sample came from the training data rather than the generative model. Both the Generator and Discriminator are differential modules. The Discriminator is trained precisely in the same way as a primary classification task done by Convolution Neural Network (CNN), having two different output nodes. While training the Generator, the Discriminator weights and bias are kept fixed. Otherwise, the Generator would be trying to hit a moving target and might never converge. The training procedure for the Generator is to maximize the probability of the Discriminator making a mistake.

Discriminator in Generative Adversarial Networks has two outputs:

1. D(x) : Probability that x comes from original dataset.
2. $D(G(z))$: Probability that G(z) comes from random distribution dataset.

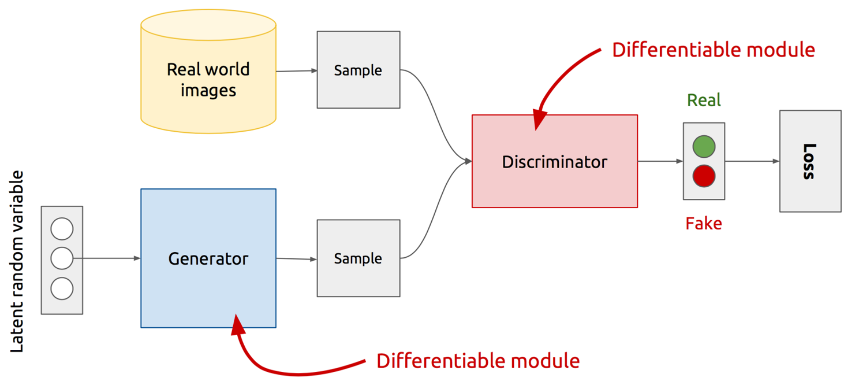


Figure 3.1: Both the Neural Networks; Generator and Discriminator are Differentiable modules.

Since GANs has two outputs, Binary Cross Entropy loss function is used.

$$Loss = -\frac{1}{outputsize} \sum_{i=1}^{outputsize} y_i log\hat{y}_i + (1 - y_i) log(1 - \hat{y}_i)$$

The label for the data is coming from either:

1. Original dataset: y=1 and $\hat{y} = D(x)$.
2. Generator: y=0 and $\hat{y} = D(G(z))$

Substituting the value of y and $\hat{y}$ in binary cross entropy loss function, we get the loss function of Generative Adversarial Networks.

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

The training criterion for the Discriminator, given any Generator, is to maximize the above loss function. The role of the Generator is reverse of that of the Discriminator, that is, to fool the Discriminator. Discriminator and Generator play a two-player min-max game which means both are adversarial in nature. AA lot of research by the machine learning community has been done on GANs in recent years. The introduction of condition in CGAN (Conditional Generative Adversarial

Network)[32] allowed to control the outputs of GANs. It involves the conditional generation of images by a generator model. Image generation can be conditioned on a class label. E.g., we want to show only the number 8 in the MNIST dataset. A condition of label or labels (y) is inserted into the objective function of GANs:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data(x)}}[log(D(x|y)] + E_{z\sim p_z(z)}[log(1 - D(G(z|y)))]$$

SSGAN (Semi-supervised GAN) [33] extends CGAN to the semi-supervised context by forcing the Discriminator to output class labels. SSGAN trains a generative model and Discriminator on the dataset with Y classes, with Discriminator made to predict Y+1 classes. Extra class is added to correspond to the outputs of G.
Deep Convolutional Generative Adversarial Networks (DCGAN) provided a way for a lot of applications of GANs. In the following sections, we will discuss a few of those applications. In section 3.1, we will discuss Improved Techniques for Training GANs[34]. In section 3.2, we will discuss DCGAN[35] and one of the most commonly used application of DCGAN i.e. Image-to-Image Translation [20, 36, 37]. Self Attention Generative Adversarial Networks (SAGAN)[38] will be discussed in section 3.3. In the last section, 3.4, we will discuss about Partial Label Learning in binary, multiclass and multilabel data[24, 25].

3.1 Improved Techniques for Training GANs

This section will discuss some problems faced by GANs and their solutions.

1. **Mode Collapse:** During the training, the Generator may collapse to a parameter setting where it always produces the same output. A good and easy strategy to avoid this type of failure is to allow the Discriminator to look at multiple data examples in combination and perform minibatch discrimination.

2. **Vanishing Gradient Descent:** Sometimes Generator fails to change weights during backpropagation. So Generator will always create fake samples as

Discriminator has already been trained, and it is very good at detecting fake vs. real samples, that is, D(G(z)) = 0. The vanishing Gradient Descent problem can be avoided by training the Generator to maximize the loss instead of minimizing it.

$$\max_G E_{z \sim p_z(z)}[log(D(G(z)))]$$

3. **Hard to Achieve Nash Equilibrium:** Training GANs require finding a Nash Equilibrium, which is a point where each player in a two-player non-cooperative game wishes to minimize its cost function. Finding Nash Equilibrium is a challenging problem. Gradient Descent fails to converge. So to encourage the convergence, gradient descent on each player's cost was applied simultaneously, despite the lack of guarantee that this procedure would converge. To encourage convergence, the following techniques were introduced.

    (a) **Feature Matching:** Generator is trained to match the expected value of the features on an intermediate layer of the Discriminator.

    (b) **Minibatch Discrimination:** Discriminator processes each example independently, and hence there is no coordination between its gradients, that is, there is no mechanism to tell the outputs of the Generator to become more dissimilar to each other. In Minibatch Discrimination, Discriminator is made to look at multiple data examples in combination.

    (c) **One-sided label smoothing:** In this strategy, targets 0 and 1 are replaced with smoothed values like 0.9 or 0.1. This reduces the vulnerability of Neural Networks to adversarial examples.

3.2 DCGAN (Deep Convolutional Generative Adversarial Network)

Convolutional Neural Networks, over the recent years, have seen massive adoption in computer vision applications. DCGAN is an extension of GAN where Convolutional neural networks were used in the Generator and Discriminator. Soon after the introduction of DCGAN in the machine learning community, GANs saw a variety of applications ranging from bioinformatics, image to image translation, super-resolution, Face Frontal View Generation, data augmentation, etc. The architecture guidelines for stable Deep Convolutional GANs are given below:

- Pooling layers are replaced with strided convolutions in the Discriminator and fractional-strided convolutions in the Generator.
- Batchnorm is used in both the Generator and the Discriminator.
- Fully connected hidden layers are removed for deeper architectures.
- ReLU activation in Generator is used for all layers except for the output, which uses Tanh.
- LeakyReLU activation in the discriminator is used for all layers.

DCGAN provided a way to a lot of applications in real world. Image-to-Image Translation techniques was one among them. Image-to-image translation (I2I) aims to transfer images from a source domain to a target domain while preserving the content representations. E.g., You can take a selfie as a source image and a cartoon as a target reference to âtranslateâ it into desired artistic style image. Image-to-Image Translation techniques can be of two types:

1. **Paired Data:** should have paired representation of an object. That is while translating from one domain x_i to another domain y_i, and both domains are from paired training data. E.g., Paired data is from changing a pencil sketch of an object to its real-life counterpart.
2. **Unpaired Data:** has no representation between x_i and y_i. The goal is to find the mapping between different image domains using unpaired training data. E.g., Unpaired data consists of images vs. paintings.

In the upcoming subsections 3.2.1and 3.2.2, we will discuss two common Image-to-Image Translation strategies; StyleGAN and SMIT (Stochastic Multi-Label Image-to-Image Translation).

3.2.1 StyleGAN

StyleGAN comprises four modifications from progressive growing GAN.

1. **Baseline Progressive GAN:** StyleGAN uses baseline progressive GAN architecture; that is, it starts from the very first resolution and gradually increases from low resolution to high resolution.

2. **Bilinear upsampling:** Usually, upsampling in Generator is done using the transposed convolution layer, but here in styleGAN, bilinear upsampling (all nearby pixels are used to calculate the pixel value) is used for upsampling.

3. **Removing Latent input z:** Latent vector z is not given directly to the Generator, rather, to start the image synthesis process, StyleGAN starts with the constant 4x4x512.

4. **Mapping Network:** In traditional GAN, the latent vector z is used as the input to Generator, but in StyleGAN, Mapping Network is used. Mapping Network comprises eight fully connected layers. It takes input from latent space z and generates a styled vector. The style vector is then transformed and incorporated into each block of the generator model via AdaIN (Adaptive Instance Normalization). The AdaIN layers involve first standardizing the output of the feature map to a standard Gaussian, then adding the style vector as a bias term.

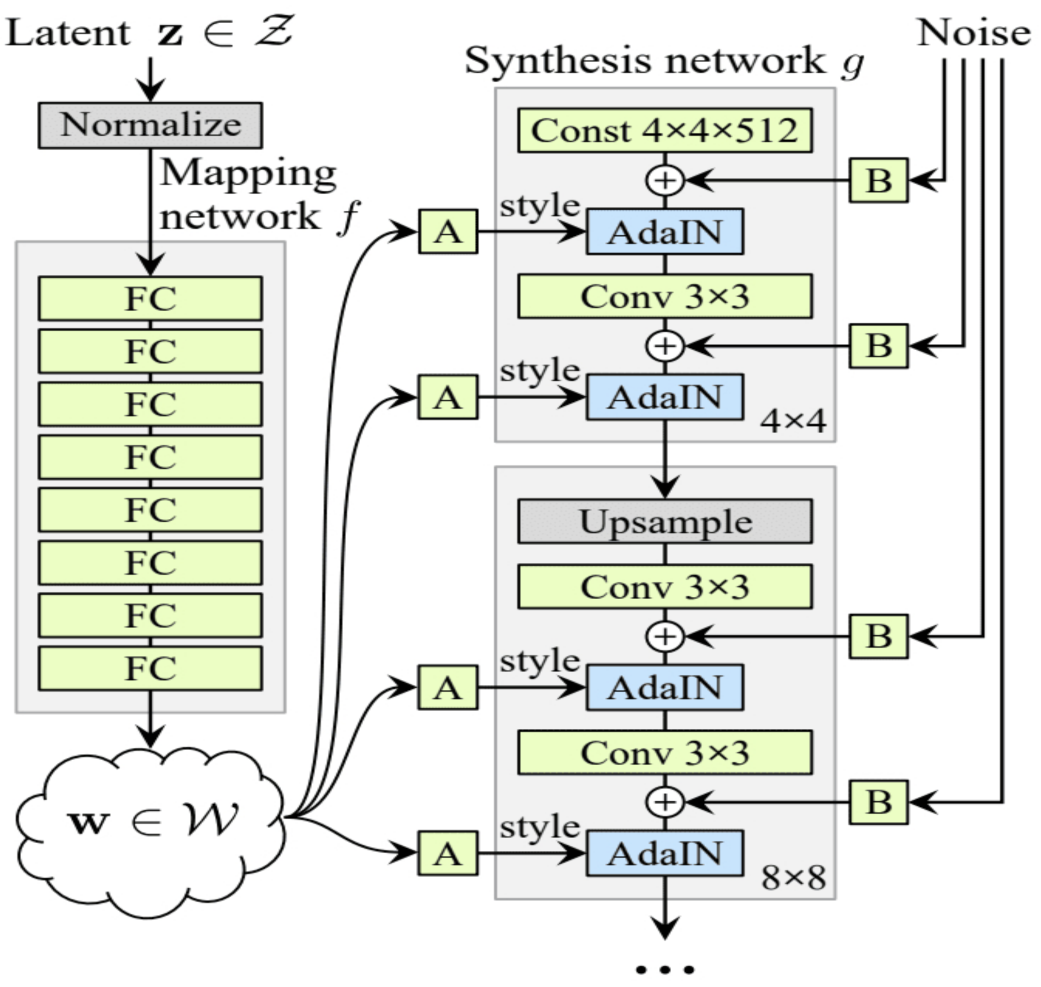


Figure 3.2: G has eighteen convolution layers with two for each resolution

3.2.2 SMIT (Stochastic Multi-Label Image-to-Image Translation)

Multilabelling in SMIT refers to the translation from one domain to the multiple desired domains. In SMIT, a joint framework of; 1) Unpaired datasets, 2) Multiple attributes, and 3) Multimodality is proposed. SMIT doesn't use style regularization; instead, Domain Embedding is used for both style and domain. Style and the target labels are individually injected through AdaIN (Adaptive Instance Normalization) layers in the Generator. A single Generator is used to conditionally produce countless fake images that hold the underlying characteristics of the source image. The Generator doesn't ignore the noise perturbation, i.e., for a different level of noise,

SMIT produces different styles with the underlying characteristics and structure of the target domain. The Discriminator not only differentiates between real and fake images but also approximates the output distribution of the actual target by means of an auxiliary classifier.
The goal of SMIT is to generate multi-attribute images with different styles using a single Generator. Given y_f and s_f, SMIT learns a mapping function G to produce X_f, without having to access to the joint distribution $p(X_r, X_f)$:

$$G(X_r, y_f, s_f) \rightarrow X_f$$

Where,
$X_r \rightarrow RealImage$
$y_r \rightarrow LabelsencodingX_r$
$s_r \rightarrow UnknownStyleDistribution$
$y_f \rightarrow TargetLabel$
$s_f \rightarrow TargetStyle$

Discriminator outputs source domain probability, i.e. real or fake and classification/regression estimator namely; $D(X_f) \rightarrow \{0, y_f\}$ and $D(X_r) \rightarrow \{0, y_r\}$.

3.3 SAGAN (Self Attention Generative Adversarial Network)

Convolution-based GANs could easily generate images with a simpler geometry like Ocean, Sky, etc. but failed on images that had some specific geometry. For example, CGAN was able to produce the texture of the furs of dogs but could not generate distinct legs. If we look at DCGAN, ConvNet (Convolutional Network) learns representation hierarchically. Complex geometry contours demand long-range details that the convolution, by itself, might not grasp. Long-range dependency might be hard to understand.

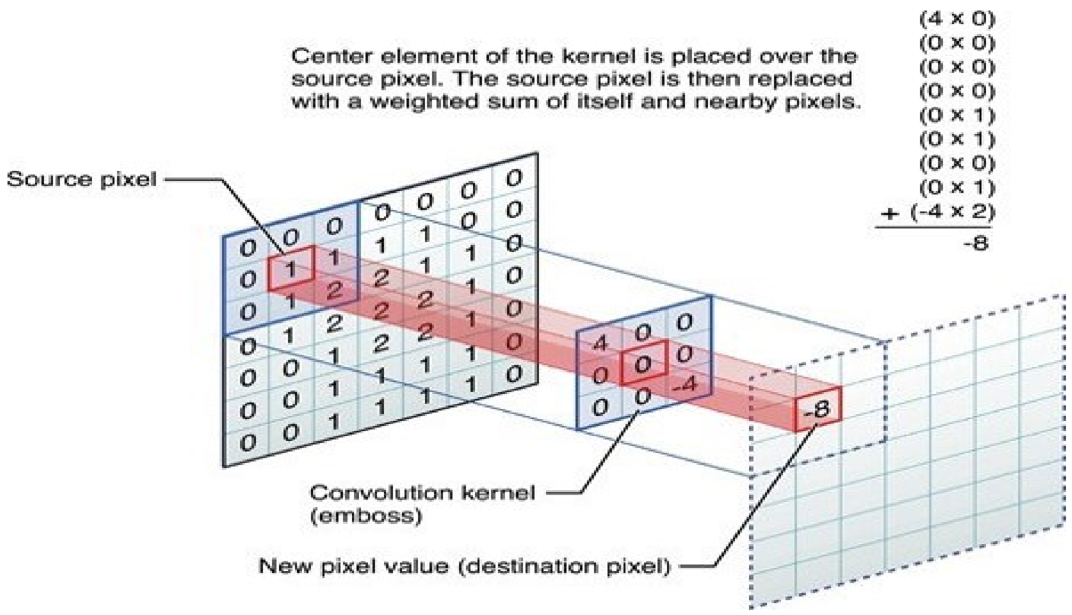

Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
Source pixel
Convolution kernel (emboss)
New pixel value (destination pixel)
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8

Figure 3.3: In the image below we can see that the output -8 is computed by the top-left Pixels of the image and it has no relation to any other part in the image

So for long-range modeling dependencies, the self-attention mechanism was introduced. The idea of attention is to give information for a broader feature space of G. Self-attention module guides the model to look at features in distant portions of the image.

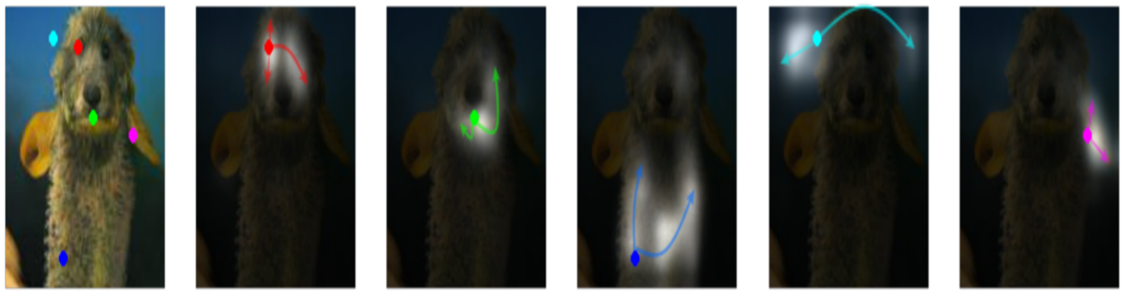


Figure 3.4: 5th image shows that when the model generates the left ear of the dog, it not only looks at the local region around the left ear but also looks at the right ear.

A self attention module takes n inputs and returns n outputs. It allows the inputs to interact with each other and find out who they should pay more attention to.

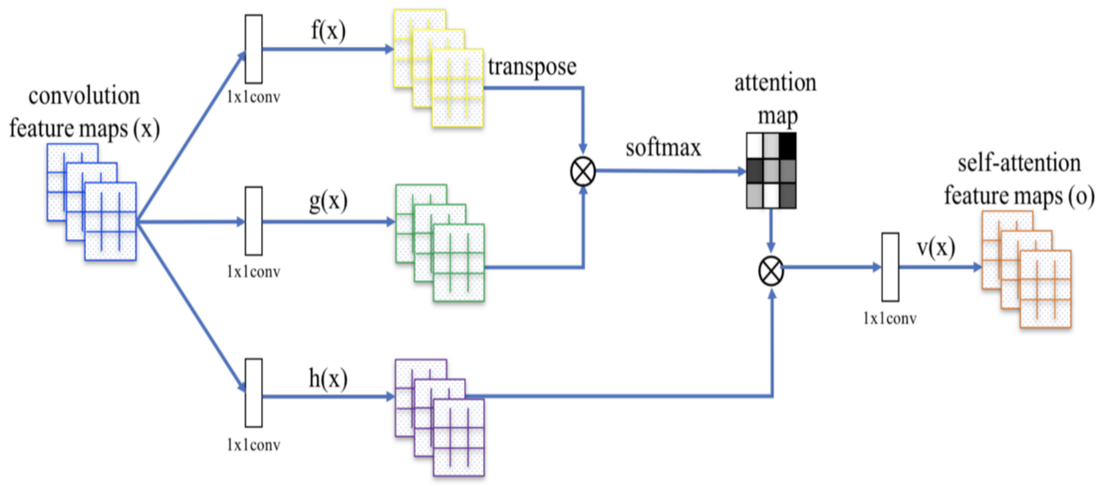


Figure 3.5: Self-Attention mechanism

The feature map obtained from the previous convolution layer is passed through three 1x1 convolutions separately. After passing through them, three feature maps are obtained f, g, and h. Now the self-attention is performed over it. Transpose of f is calculated and matrix-multiply by the g and takes the soft-max on all the rows. So we get an attention map as a result which is then multiplied by the h vector, and an output self-attention feature map is obtained. At last, multiply the final output by a learnable scale parameter and add back the input as a residual connection.

$$g(x_i) = w_g x \quad ; \quad x \in R^{c*n} \ , \ w_g \in R^{c'*c} \ , \ c' = \frac{c}{8}$$

$$f(x_i) = w_f x \quad ; \quad w_f \in R^{c'*c}$$

$$s_{ij} = f(x_i)^T g(x_i)$$

$$\beta_{j,i} = \frac{exp(x_{ij})}{\sum_{i=1}^{n} exp(s_{ij})} \quad ; \quad \beta \in R^{n*n}$$

β gives us the attention map which gives us the extend to which the model attends to the ith location when synthesizing the jth region.

$$f(h_i) = w_h x \quad ; \quad w_h \in R^{c*c}$$

$$o_j = \sum_{i=1}^{n} \beta_{j,i} h(x_{i)} \quad ; \quad o = (o_1, ..., o_n) \in R^{c*n}$$

Final output is multiplied by a learnable parameter γ and added back the input as a residual connection.

$$y_i = \gamma o_i + x_i$$

γ is initialized to 0, and it cancels out the attention layers at the beginning. As a result, the network only relies on the local representation from local convolution layers. Then γ receives gradient descent updates, and the network gradually allows the passage of signals from non-local fields.
Self Attention Module is used in both the Generator and the Discriminator. The attention Module in Generator creates images with fine details, and the attention module in Discriminator accurately enforces complicated geometric constraints on the global image structure.

3.4 Partial Label Learning

In multilabel learning, each instance can belong to more than one class or label. E.g., The weather can be sunny and cloudy at the same time, or a movie can be both comedy and action. The partial label learning problem is ubiquitous in multilabel data. In section 3.4.1, we will discuss PLLGAN (Partial Label Learning via Generative Adversarial Nets) [24]. PLLGAN deals with partial label learning problems in multiclass data. In section 3.4.2, we will discuss PMLGAN (Partial Multi-Label Learning via GANs) [25]. PMLGAN deals with partial label learning in multilabel data.

3.4.1 PLLGAN (Partial Label Learning via GANs)

In PLLGAN, each sample is provided with multiple candidate labels or labels (candidate to be true label or labels) while only one of them is correct. Using PLLGAN, partial label problems are solved by combining CGAN (Conditional GAN) and SSGAN (Semi-supervised GAN). The Generator adopts the idea of CGAN; candidate labels are given to the Generator as a condition to generate samples similar to the real samples. The Discriminator adopts the idea of SSGAN; that is, it not only distinguishes generated and actual samples but also predicts the ground-truth labels. Let $D = \{(x_i, s_i)\ ;\ 1 \leq i \leq n\}$ be a partial label training set of n samples where x_i is the feature vector, s_i is the corresponding candidate label set, and y_i is the ground truth label set. $y_i \in \{0, 1\}$; $y_{ij} = 1$ indicates jth label is among the candidate label set of the sample x_i, and $y_{ij} = 0$ means jth label is a non candidate label of x_i. The total loss function of PLLGAN is given by:

$$\max_G \min_D V(D, G) = E_{x_i|y_i^p \sim p_{data(x_i|y_i^p)}} ||D(x_i|y_i{}^p) - y_i^{rc}||_2^2 \\ + E_{z_i|y_i^p \sim p_{z(z_i|y_i^p)}} ||D(G(z_i|y_i{}^p)) - y_i^{fc}||_2^2$$

Here, p_{data} is the real training data distribution, z is the noise vector sampled from an Std. Normal Distribution, y_i^p is the partial label vector of real training data, y_i^{rc} is the reconstructed labels of the real samples, and y_i^{fc} is the reconstructed labels of the generated samples.

3.4.2 PMLGAN (Partial Multi-Label Learning via GANs)

IN PMLGAN, each instance is assigned multiple candidate labels that are partially relevant; some irrelevant noise labels are assigned with the ground-truth labels, or some labels are missing. There are four components in PMLGAN: 1) Generator, 2) Discriminator, 3) Disambiguation network and 4) Prediction network. The Generator generates samples in the feature space given latent vectors in the label space. The discriminator separates generated and real data. The disambiguation network predicts the probability of each candidate label being an additive noise for the training instance. The prediction network predicts disambiguated true labels of each instance from its input features.

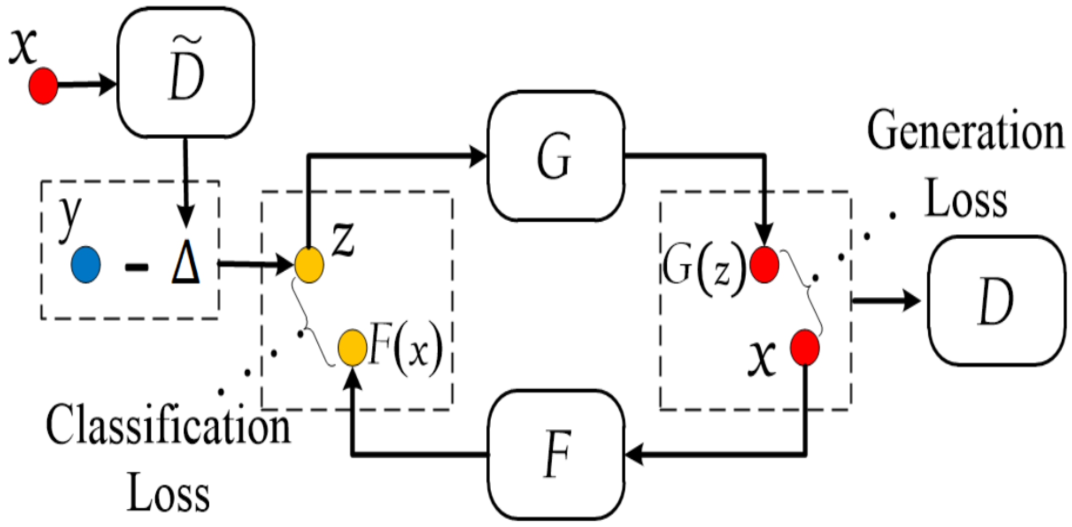


Figure 3.6: Four Components: Generator G, Discriminator D, Disambiguation network $\hat{D}$, and Prediction network F.

Let $S = (X, Y) = \{(x_i, y_i)\}_{i=1}^{n}$ be a training set with n samples. Here, x_i is the input feature vector for ith instance and $y_i \in \{0, 1\}$ is the corresponding annotated label indicator vector. The objective of PMLGAN is to obtain ground truth labels z_i from each annotated candidate label vector y_i, that is, we drop the additional 1's from each candidate label vector y_i. This is obtained using Disambiguation network $\hat{D}$:

$$\hat{D} \ : \ \sigma_x \rightarrow \sigma_\delta$$

Here, σ denotes the corresponding domain space. The above equation predicts the irrelevant labels for a given instance. The true label indicator can be recovered as:

$$z_i = ReLU(y_i, \delta_i)$$

Here, δ_i is the output of disambiguation network $\hat{D}(x_i)$. ReLu activation function ensures that disambiguation effort is only counted on the candidate labels. Then the prediction network can be learned $F \ : \ \sigma_x \rightarrow \sigma_z$, that is, a multilabel classifier to predict the disambiguated ground truth labels for each instance. With The prediction and disambiguation network, we can perform partial multilabel learning by minimizing the classification loss on training data S:

$$\min_{F,\hat{D}} L_c(X, Y; F, \hat{D}) = \textstyle\sum_{(x_i,y_i)\sim S} l_c(F(x_i), z_i)$$
$$\text{s.t.} z_i = ReLU(y_i - \delta_i) \ , \ \delta_i = \hat{D}(x_{i)\ , \forall (x_i,y_i) \sim S}$$

Here, $l_c(.,.)$ is the cross-entropy loss between the predicted probability of each label and its confidence being a ground truth label. Label indicator vectors are discrete values, and hence sigmoid activation function on the last layer of each network, $\hat{D}(x)$ and $F(x)$, can predict the probability of each class label being the additive irrelevant label and the ground truth label respectively.

Chapter 4

MLGAN

This section presents our approach MLGAN (Multilabel Generative Adversarial Networks). In MLGAN, instead of generating instances for only one class, we will generate instances for all labels with an imbalance. Features, as well as labels for multilabel datasets, are synthesized using adversarial training. Instead of only considering the neighboring minority samples as the seed to generate new synthetic samples, we will consider taking the whole distribution of minority samples in those classes where we want to overcome the imbalance problem. MLSMOTE creates data by looking at neighbourhood only, so there is a limitation on the number of samples that we can create. Synthesizing beyond that limit may make our model prone to overfitting as it would more or less behave like random oversampling. In MLGAN, there is no limit on the number of generated distinct samples. We can create infinite samples from n training samples. The first step is to split the data into train and test using k-fold cross-validation. The value of k is set to 5, keeping four folds for T_{train} and one fold for T_{test}. In our approach, we have three main aspects of solving:

1. First aspect will be to know which labels are minority ones.
2. Once minority labels are selected, we use representative class samples to represent each class distribution of those minority labels. The class representative samples will be given to GANs to generate synthetic samples.

3. Once synthetic samples are generated, multilabel learning algorithms are used to handle multilabel datasets.

In sections 3.1 and 3.2, a discussion on the first two aspects will be given in detail. The third aspect will be discussed as we move ahead in this dissertation.

4.1 Measuring Imbalance

In binary and multiclass, the imbalance ratio is used to measure the imbalance in a particular label. The imbalance ratio is the proportion of majority class instances to the number of minority class instances. In multilabel learning, the imbalance of a particular label is evaluated by IRLbl (Imbalance ratio per label) [11][12]. IRLbl is the ratio between the majority and the considered labels.

$$IRLbl_i = \frac{max_{j=1,...,q}\{m_j^1\}}{m_i^1}, i = 1, 2, ..., q$$

where, $m_i^b = \|\{(x_j, y_{ji}), y_{ji} = b, 1 <= j <= n\}\|$ be the number of instances whose ith label value is equal $b \in \{0, 1\}$.

MeanIR represents the average level of imbalance in multi-label data which is calculated by considering the average of IRLbl.

$$MeanIR = \frac{1}{\|q\|}\sum_{i=1}^{\|q\|} IRLbl_i$$

CVIR (Coefficient of variation of IRLbl) examines whether all labels suffer from a similar or different level of imbalance.

$$CVIR = \frac{1}{MeanIR}\sqrt{\sum_{i=1}^{q}\frac{(IRLbl_i - MeanIR)^2}{q-1}}$$

MeanIR sets the threshold, so we don't need to consider setting a particular number of labels as a minority. CVIR used in Equation **3** examines whether all labels suffer from a similar or different level of imbalance or, on the contrary, there are big differences in them. The higher the CVIR, the larger this difference will be. For instance, for 3 three classes C1 = [1,1,0,0], C2 = [0,1,0,0] and C3 = [1,1,1,0], we calculate the IRLbl of each class. $IRLbl_{C1}$ is 1.5, $IRLbl_{C2}$ is 3 and $IRLbl_{C3}$ is 1. MeanIR is 1.8333, and CVIR is 0.5677. Here, 1.8333 is the threshold; that is, any class having a value more than 1.8333 has an imbalance. In this case, C2 has an imbalance. CVIR value indicates that there is a 56% variation in the IRLbl values. The values of MeanIR and CVIR shown in section 4 are of the entire multilabel dataset. We will only be calculating these values on T_{train}, keeping T_{test} as it is for performance testing. Having CVIR greater than 1 implies a huge imbalance.

4.2 Generative Adversarial Network (GAN) Architecture

After finding the imbalance ratio of each label and determining the threshold MeanIR, we will start generating synthetic samples for all the classes with an imbalance. This process is done until $IRLbl_{label} = MeanIR$, that is, the Imbalance ratio of each label equals MeanIR. To synthesize new samples, We use CTGAN [10] as our base model. We will be using the architecture of CTGAN because the real-world tabular consists of mixed types of data (Discrete or Continuous). In CTGAN, both softmax and tanh are applied to the output. A tanh activation function is employed in the last layer of the network to normalize the discrete values in the range [-1,1]. Continuous values are usually non-Gaussian, so Mode-specific Normalization is used to normalize continuous values.

CTGAN uses Conditional-Generator so that it can be interpreted as the conditional distribution of rows given the particular value at that specific column, i.e., if j^* is the value in the i^*th discrete column D_{i*} that has to be matched by the generated samples $\hat{r}$ then

$$\hat{r} \sim P_g(row, D_{i*} = j^*)$$

CTGAN consists of three key elements:

Conditional Vector: Discrete columns $D_1, ..., D_N$ end up as one-hot vectors $d_1, ..., d_N$ such that ith one-hot vector is $d_i = [d_i^{(j)}]$, where $j = 1, ..., \|D_i\|$. The condition is expressed in term of mask vectors as

$$[m_i^{(j)}] = \begin{cases} 1 & \text{i = i}^* \text{ and } j = j^* \\ 0 & otherwise \end{cases}$$

where, $m_i = [m_i^{(j)}]$ be the ith mask vector associated with the ith one-hot vector d_i

Eg. D1=(1,2,3,4,5) and D2=(1,2,3) are two discrete columns, the condition (D2=2) is expressed by the mask vector m_1 = [0,0,0,0,0] and $m_2 = [0, 1, 0]$. So the condition = [0,0,0,0,0,0,1,0] .

Training by sampling: CTGAN model employs the training-by-sample technique; that is properly sample the conditional vector and training data, which helps the model explore all possible values evenly.

Generator Loss: Cross entropy between m_i and $\hat{d}_i$ is added to penalize the loss so we get $m_i = \hat{d}_i$,averaged over all the instances.

4.3 Proposed Approach

We consider finding the imbalance on the T_{train} by using two measures; IRLbl and MeanIR. After finding the classes with an imbalance, we will start generating synthetic samples using Generative Adversarial Networks, whose architecture is discussed in the above section. We will keep generating synthetic samples until $IRLbl_{label} = MeanIR$. Following that, we pass the training data concatenated with synthesized data into five different multilabel learning algorithms that we will be discussing in the next section. Test data will be used to evaluate the model performance. We calculate the efficiency of MLGAN by comparing it with MLSMOTE on six multilabel datasets. We will synthesize the same amount of synthetic samples in MLSMOTE as that synthesized in MLGAN. The number of synthetic samples to be generated is decided by the imbalance ratio IRLbl of each label, while MeanIR sets the threshold value. The implementation of both will be compared using many evaluation criteria discussed briefly in the next section. We will be taking only a representation, representing each class distribution, from the minority samples in T_{train}. This is done to improve the time performance of MLGAN. Algorithm 1 shows the pseudo-code of the proposed MLGAN algorithm.

Algorithm 1: MLGAN pseudo-code.

```
Require: $D \leftarrow MultilabelDataset$
 1: $Train, Test \leftarrow kfoldcrossvalidation$
 2: $L \leftarrow LabelsInTrainSet(Train)$
 3: $minsamples \leftarrow minsamples(Train)$          ▷ Bag of minority label samples
 4: $minsamples \leftarrow sample(minsamples, frac = 0.2)$
 5: $MeanIR \leftarrow CalcMeanIR(Train, L)$
 6: for each label in L do
 7: $IRLbl_{label} \leftarrow CalcIRLbl(Train, label)$
 8: if $IRLbl > MeanIR$ then
 9: while $IRLbl == MeanIR$ do
10: $synsamples \leftarrow GAN(minsamples, epochs = 80, condition)$
11: $NewTrain \leftarrow concatenate(Train, synsamples)$
12: end while
13: end if
14: end for
```

Unlike MLSMOTE, the input of algorithm 1 is Multilabel Dataset (D) only and not the number of neighbors to use (K). As suggested in MLSMOTE, the number of neighbors considered is five by default. While comparing MLGAN with MLSMOTE, we have also set the value of k to 5. In section 4, we will see how changing the value of the parameter k in MLSMOTE will impact the performance. MLGAN is free from this parameter which is an excellent advantage of this approach. Discrete columns are given as a condition to the GAN architecture. The output of algorithm 1 will be new T_{train} which will consist of samples from the original T_{train} and the newly generated samples. T_{test} is not changed in this process and will be called only while evaluating the performance using multilabel learning algorithms.

Chapter 5

Experimental Setup and Results

To assess the benefits of MLGAN, an extensive experimental study is conducted in this work. The results produced by MLGAN are compared against those obtained by MLSMOTE.

5.1 Experimental Setup

5.1.1 Datasets

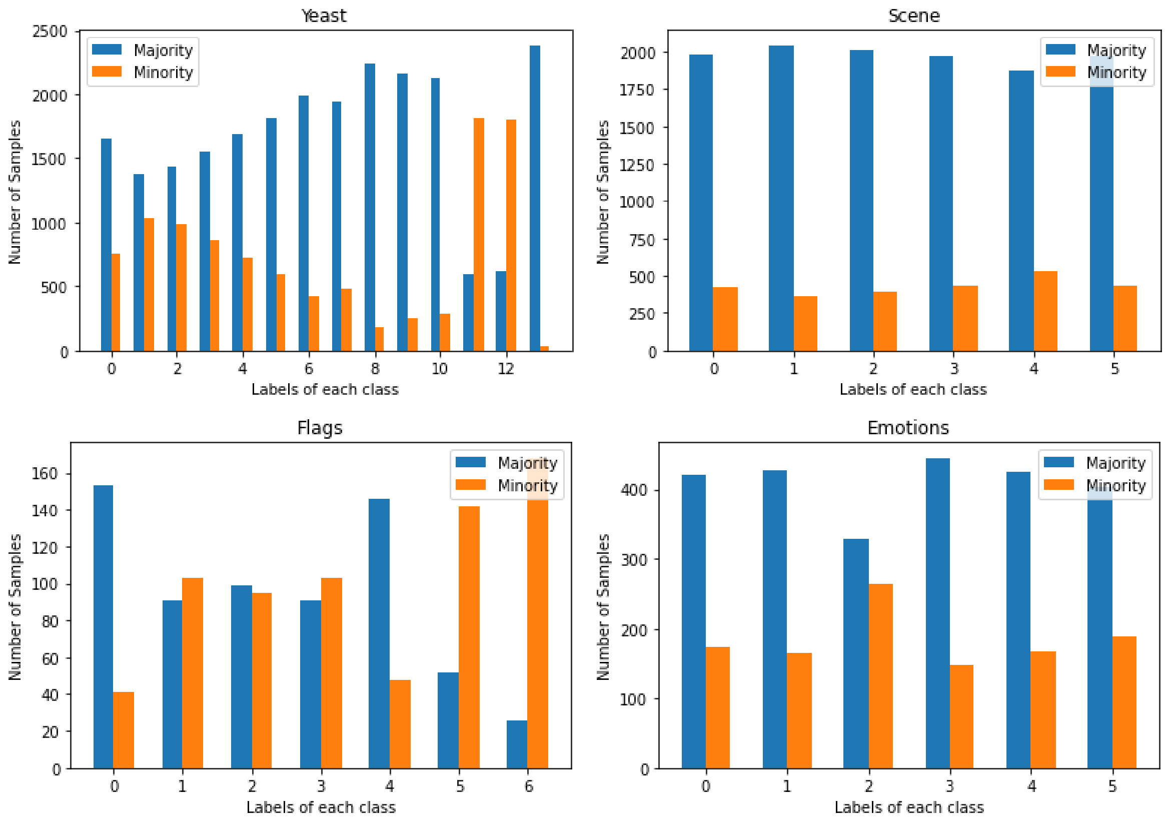
In order to get an effective performance evaluation, six real-world multi-label datasets are collected for experimental analysis. All the datasets can be downloaded from Kaggle[1], UCO[2] and MULAN[3]. All the multilabel datasets were preprocessed on which the effect of rebalancing produced by MLGAN was tested. Table 1 reports the detailed information of all the datasets. For each dataset, the k-fold cross-validation technique is used to split the data in T_{train} and T_{test}, where k is set to 5.

[1]`https://www.kaggle.com/c/planet-understanding-the-amazon-from-space/data/`
[2]`https://www.uco.es/kdis/mllresources/`
[3]`http://mulan.sourceforge.net/datasets-mlc.html/`

Table 5.1: Characterization measures of six datasets used in experimentation

Dataset	Domain	Samples	Features	Labels	Label Count
Scene	Image	2407	294	6	1.074
Yeast	Biology	2417	103	14	4.237
Amazon	Planet	40478	512	17	2.87
Emotions	Music	593	72	6	1.869
Mirflickr	Images	2500	150	24	3.716
Flags	Image	194	19	7	3.392

Figure 5.1 shows the number of minority and majority label samples belonging to a particular class. As it is evident from the graph that there is a considerable imbalance in almost all of the multilabel datasets.



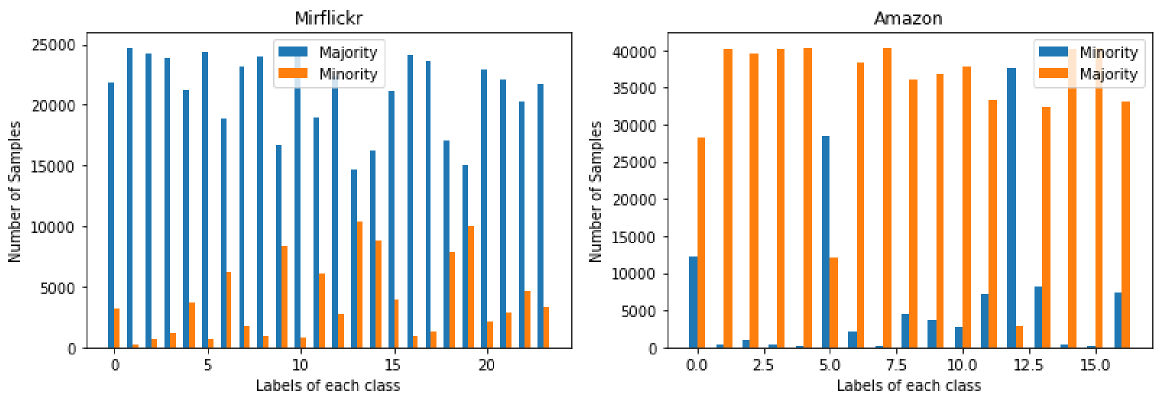


Figure 5.1: Number of samples of the minority and majority label in each class.

5.1.2 Classifiers

The multilabel learning problem is tackled by either transforming into other well-established learning scenarios or adapting popular learning techniques. In this dissertation, a simple categorization of five well-known Multilabel learning algorithms is adopted. A brief introduction of them is given in this section. The objective of using several algorithms is to check whether GANs can reduce the imbalance in multilabel datasets in a classifier-independent way; that is, the classifier should have a deplorable influence on the effectiveness of the resampling strategy.

- Binary Relevance[13] decomposes the multilabel learning problem into q independent binary learning problem and trains a classifier on each of these decomposed binary problems. Each decomposed binary learning problem contains the same number of instances as the original multilabel learning. For new instances, Binary Relevance outputs the union of the labels positively predicted by the q classifiers.

- Label Powerset[14] decomposes the multilabel learning problem into a multi-class classification problem. It considers each unique set of labels as one of the classes of a new single-label classification task.

- In RAkELd[15] (Random Label Space Partitioning with Label Powerset), Label space is divided into equal partitions of size k. Label Powerset classifier is trained on each partition, and prediction is made by summing the result of all trained classifiers.

- Classifier chains[16] transform a multilabel problem into a chain of binary classification problems, where subsequent binary classifiers are built upon the predictions of preceding ones.

- Majority Voting Classifier[17] partitions the label space into separate sublabel spaces using provided clustering class. A base multilabel subclassifier is trained on each subspace. A label is assigned to an instance if more than half of all classifiers (majority) trained on a label subspace that contains the label have assigned it to a given sample.

In all the multilabel learning approaches discussed above, the employed base model is Support Vector Machines (SVM). The objective of SVM is to find a hyperplane in an n-dimensional space that has the maximum margin such that it distinctly classifies the data points. If data is linearly separable, we can select the margin such that there are no points between them, and we try to maximize their distance. In such cases, a pair of (w,b) exists such that:

$$w.x_i + b \geq 1, \forall x_i \in P$$

$$w.x_i + b \leq 1, \forall x_i \in N$$

This can be written as:

$$y_i(w.x_i + b) \geq 1, 1 \leq i \geq n$$

where, $y_i = +1$ for P (Positive samples), and $y_i = -1$ for N (Negative samples). The optimization can be set up as a convex quadrating programming problem:

$$\min_{w} \quad \frac{\|w\|^2}{2}$$

s.t.

$$y_i(w.x_i + b) \geq 1, 1 \leq i \leq n$$

Data contains misclassified instances that can be addressed by soft margin in the real world. Soft margin is accomplished by introducing slack variables ξ_i, i=1,2,...,n in the constraints.
Soft margin condition becomes:

$$y_i(w.x_i + b) \geq 1 - \xi_i, 1 \leq i \leq n$$

$$\xi_i \geq 0$$

Our goal in soft margin SVM is to maximize the margin while also minimizing the sum of slacks. The optimization problem for soft margin becomes:

$$min\tfrac{\|w\|^2}{2} + c\textstyle\sum_{n=1}^{N}(\xi_i)$$

s.t.

$$y_i(w.x_i + b) \geq 1 - \xi_i, 1 \leq i \leq n \ ,$$
$$\xi_i \geq 0$$

Here c is a regularization parameter controlling the tradeoff between large margin and small hinge loss.

5.1.3 Evaluation Metrics

We have evaluated the performance of the compared algorithms in terms of nine common metrics.

1. Hamming Loss (HammingL): It reports that how many times on average the relevance of an example to a class label is incorrectly predicted.

$$HammingL = \frac{1}{nL}\sum_{i=1}^{n}\sum_{j=1}^{L} I(y_i^j \neq \hat{y_i^j})$$

   where, n is the number of samples y_i is the set of actual Labels, L denotes the length of label and $\hat{y}_i$ is the set of predicted labels.

2. Accuracy Score (Acc Score): Accuracy score for the sample is defined by the proportion of the number of correctly predicted labels and total number of labels. Final accuracy is defined by taking the average of accuracy over all test samples; that is, proportion of the sum of accuracy of all training samples and number of training samples

$$AccScore = \frac{1}{n}\sum_{i=1}^{n}\frac{\|y_i \cap \hat{y}_i\|}{\|y_i \cup \hat{y}_i\|}$$

3. Exact Match Ratio (EMR): EMR extends the concept of accuracy but does not account for partially correct labels.

$$EMR = \frac{1}{n}\sum_{i=1}^{n} I(y_i = \hat{y}_i)$$

   where, I denotes the Indicator Function.

4. 0/1 Loss: We calculate proportion of instances whose actual value is not equal to predicted value.

$$0/1Loss = \frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$$

5. Recall : Recall is defined as the proportion of predicted correct labels to the total number of predicted labels, averaged over all instances.

$$Recall = \frac{1}{n}\sum_{i=1}^{n}\frac{\|y_i \cap \hat{y}_i\|}{\|\hat{y}_i\|}$$

6. Precision: It is the proportion of predicted correct labels to the total number of actual labels, averaged over all instances.

$$Precision = \frac{1}{n}\sum_{i=1}^{n}\frac{\|y_i \cap \hat{y}_i\|}{\|y_i\|}$$

7. F1-Measure: It is the harmonic mean of recall and precision.

$$F1 - measure = \frac{1}{n}\sum_{i=1}^{n}\frac{\|y_i \cap \hat{y}_i\|}{\|y_i\| + \|\hat{y}_i\|}$$

8. Ranking Loss (RankL): This determines the percentage that the ranking of negative labels of the example are higher than that of positive labels.

$$RankL = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{\|y_i\|\|\hat{y}_i\|}$$

9. Average AUC (AUC): Calculates the average fraction of times a positive instance of all-class labels has a higher ranking than a negative instance.

5.2 Experimental Results

Firstly, we will measure imbalance using the concept discussed in section 3.1. Table 2 shows the MeanIR and CVIR of the six datasets used in this study. The more the CVIR of a multilabel dataset, the more is the overall imbalance.

Table 5.2: Imbalance ratio per label

Dataset	MeanIR	CVIR
Scene	1.252	0.121
Yeast	4.274	1.796
Amazon	27.01	1.501
Emotions	1.478	0.179
MirFlickr	6.756	1.244
Flags	1.341	0.222

The values of MeanIR and CVIR showed in Table 2 are of the entire multilabel dataset. It should be noted that the values are shown in Table 2 show for each multilabel dataset the MeanIR and CVIR values before applying MLGAN. Since we are splitting the multilabel datasets using k-fold cross-validation, where k is set to 5. Out of the five folds, only four folds are taken as T_{train}, so the values of MeanIR and CVIR are calculated only on T_{train} as MLGAN is applied only to training partitions.

After applying MLGAN to the multilabel datasets, we reassess the CVIR for each multilabel dataset. Figure **2** compared CVIR of the original dataset and rebalanced dataset using the two strategies; MLGAN and MLSMOTE. From these results, it can be drawn that MLGAN does produce the change in imbalance level.

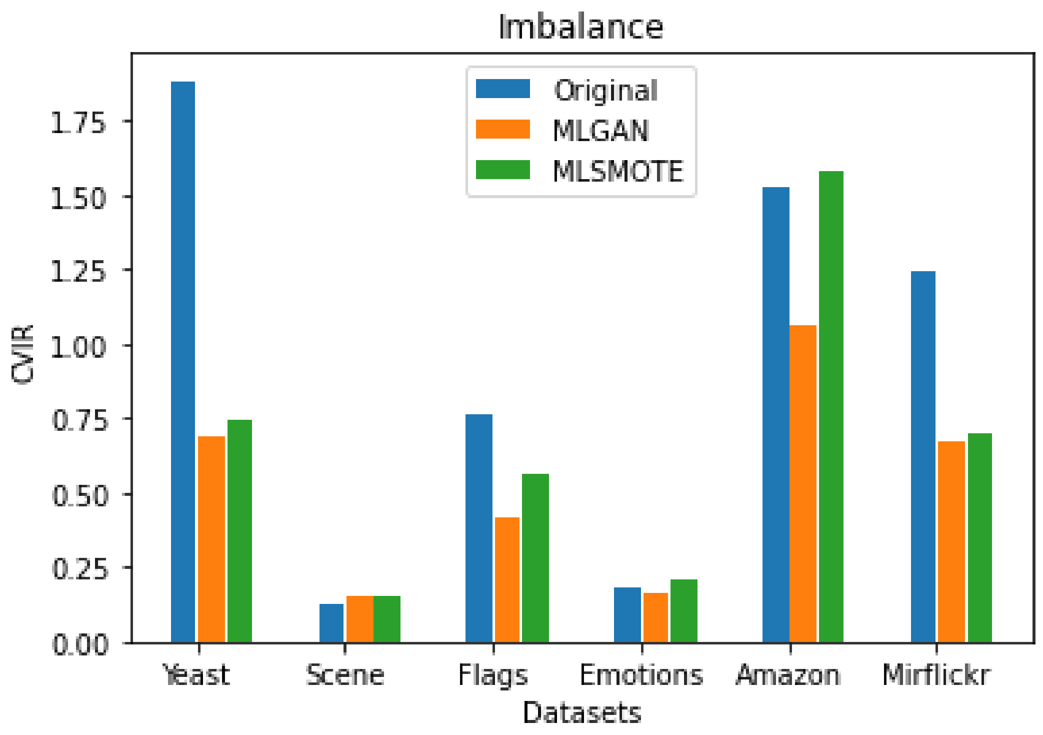


Figure 5.2: CVIR for each case: Higher CVIR means huge Imbalance in the dataset.

The next step is the analysis of results produced by the two strategies, MLGAN and MLSMOTE. We will also be comparing the results of both MLGAN and MLSMOTE with the Original dataset; that is, we will compare the results obtained by MLGAN and MLSMOTE with the multilabel dataset in which no augmentation of samples has been done. Table 3 shows the results obtained when multilabel learning approaches discussed in section 4.1.2 are used. All the results are obtained by keeping the number of epochs fixed to 80 in the case of MLGAN. In the case of MLSMOTE, the value of k (nearest neighbors) is kept fixed at 5. For the ease of comparing the results produced by the two strategies, Table 3 only shows the F1-Measure, which is nothing but the harmonic mean of recall and precision. 0/1 Loss is also not shown in the results as $EMR + 0/1 Loss = 1(100\%)$

Table 5.3: Classification results with MLGAN and MLSMOTE when five Multilabel learning algorithms are used.

Datasets		**Binary Relevance**				
		HammingL	**EMR**	**F1-Measure**	**RankL**	**AUC**
Scene	MLGAN	0.0557	0.7255	0.7713	0.2363	0.7540
	Original	0.0547	0.7318	0.7733	0.2338	0.7524
	MLSMOTE	0.0602	0.7151	0.7595	0.2471	0.7571
Emotions	MLGAN	0.3011	0.0168	0.0672	0.9481	0.5075
	Original	0.3039	0.0084	0.0686	0.9495	0.5156
	MLSMOTE	0.3008	0.0169	0.0790	0.9378	0.5114
Flags	MLGAN	0.3105	0.0263	0.6876	0.4699	0.5095
	Original	0.3195	0.0526	0.6590	0.4776	0.4974
	MLSMOTE	0.3150	0.0256	0.6849	0.4752	0.5075
MirFlickr	MLGAN	0.1507	0.0212	0.0862	0.9231	0.5077
	Original	0.1520	0.0204	0.0920	0.9207	0.5078
	MLSMOTE	0.1511	0.0170	0.0897	0.9231	0.5089
Amazon	MLGAN	0.0741	0.4019	0.7557	0.3062	0.5656
	Original	0.0745	0.3995	0.7527	0.3099	0.5647
	MLSMOTE	0.0747	0.3970	0.7518	0.3113	0.5667
Yeast	MLGAN	0.1839	0.1987	0.6315	0.4254	0.5844
	Original	0.1888	0.1887	0.6239	0.4451	0.5809
	MLSMOTE	0.1861	0.1884	0.6258	0.4324	0.5815

Datasets		**Label PowerSet**				
		HammingL	**EMR**	**F1-Measure**	**RankL**	**AUC**
Scene	MLGAN	0.0575	0.7900	0.8427	0.1705	0.7562
	Original	0.0554	0.7962	0.8489	0.1642	0.7545
	MLSMOTE	0.0609	0.7796	0.8322	0.1808	0.7575
Emotions	MLGAN	0.3445	0.1681	0.465	0.5229	0.6190
	Original	0.3599	0.1512	0.4103	0.5746	0.5351
	MLSMOTE	0.3571	0.1680	0.4439	0.5384	0.5980
Flags	MLGAN	0.2894	0.2308	0.6799	0.4244	0.5419
	Original	0.3003	0.2051	0.6695	0.4311	0.5000
	MLSMOTE	0.3076	0.1282	0.6496	0.4658	0.5000
MirFlickr	MLGAN	0.1689	0.071	0.2174	0.8019	0.5066
	Original	0.1739	0.0494	0.1855	0.8301	0.5002
	MLSMOTE	0.1734	0.0504	0.1845	0.8316	0.5002
Amazon	MLGAN	0.0752	0.4770	0.7677	0.2815	0.6128
	Original	0.0766	0.4673	0.7628	0.2872	0.6109
	MLSMOTE	0.0759	0.4655	0.7647	0.2848	0.6170
Yeast	MLGAN	0.2048	0.2561	0.6292	0.4176	0.5804
	Original	0.2062	0.2877	0.6282	0.4207	0.5865
	MLSMOTE	0.1968	0.2727	0.6481	0.4012	0.5841

Datasets		**RAkELd**				
		HammingL	**EMR**	**F1-Measure**	**RankL**	**AUC**
Scene	MLGAN	0.0575	0.7475	0.8232	0.1837	0.7508
	Original	0.0705	0.7385	0.7773	0.2292	0.8646
	MLSMOTE	0.0592	0.7380	0.8073	0.1987	0.7647
Emotions	MLGAN	0.3417	0.1008	0.3114	0.7397	0.5132
	Original	0.3474	0.1779	0.3093	0.7220	0.5082
	MLSMOTE	0.3305	0.0504	0.2647	0.7913	0.5231
Flags	MLGAN	0.2932	0.1578	0.6137	0.4864	0.5146
	Original	0.3120	0.1578	0.6488	0.4666	0.5085
	MLSMOTE	0.3479	0.1025	0.6223	0.5106	0.4984
MirFlickr	MLGAN	0.1554	0.0218	0.1009	0.9129	0.5083
	Original	0.1547	0.0266	0.0771	0.9280	0.5048
	MLSMOTE	0.1528	0.0314	0.1340	0.8888	0.5098
Amazon	MLGAN	0.0746	0.4082	0.7598	0.2997	0.5762
	Original	0.0750	0.3974	0.7565	0.3054	0.5750
	MLSMOTE	0.0751	0.3998	0.7464	0.3184	0.5704
Yeast	MLGAN	0.1871	0.2024	0.6427	0.4108	0.5844
	Original	0.1918	0.2107	0.6269	0.4255	0.5872
	MLSMOTE	0.1903	0.2086	0.6344	0.4162	0.5830

Datasets	**Method**	**Classifier Chains**				
		HammingL	**EMR**	**F1-Measure**	**RankL**	**AUC**
Scene	MLGAN	0.0643	0.7596	0.8108	0.2001	0.7954
	Original	0.0661	0.7588	0.8087	0.2018	0.7916
	MLSMOTE	0.0703	0.7484	0.8011	0.2095	0.7533
Emotions	MLGAN	0.2689	0.0000	0.0644	0.9537	0.5164
	Original	0.2717	0.0000	0.0644	0.9537	0.5142
	MLSMOTE	0.2703	0.0000	0.0644	0.9537	0.5153
Flags	MLGAN	0.3040	0.2051	0.6589	0.4487	0.5152
	Original	0.3040	0.2051	0.6788	0.4226	0.5134
	MLSMOTE	0.3186	0.1025	0.6930	0.4538	0.5256
MirFlickr	MLGAN	0.1588	0.0364	0.1797	0.8397	0.5113
	Original	0.1574	0.0296	0.1572	0.8689	0.5041
	MLSMOTE	0.1550	0.0246	0.1294	0.8842	0.5070
Amazon	MLGAN	0.0742	0.4016	0.7555	0.3065	0.5654
	Original	0.0721	0.4094	0.7617	0.3011	0.5682
	MLSMOTE	0.0734	0.4001	0.7560	0.3059	0.5679
Yeast	MLGAN	0.2057	0.2401	0.6199	0.4308	0.5955
	Original	0.2151	0.2215	0.6140	0.4350	0.5911
	MLSMOTE	0.2131	0.2252	0.6106	0.4397	0.5869

Datasets	**Method**	**Majority Voting Classifier**				
		HammingL	**EMR**	**F1-Measure**	**RankL**	**AUC**
Scene	MLGAN	0.0696	0.6819	0.7151	0.2881	0.8489
	Original	0.0696	0.6715	0.7144	0.2943	0.8432
	MLSMOTE	0.0727	0.6777	0.7079	0.2978	0.8416
Emotions	MLGAN	0.2857	0.0168	0.1316	0.9005	0.5308
	Original	0.2937	0.0254	0.1002	0.9209	0.5264
	MLSMOTE	0.2997	0.0168	0.0868	0.9327	0.5164
Flags	MLGAN	0.3646	0.0789	0.521	0.6232	0.6199
	Original	0.3809	0.0512	0.6020	0.5405	0.4878
	MLSMOTE	0.3759	0.0263	0.6284	0.5618	0.5088
MirFlickr	MLGAN	0.1524	0.0186	0.0000	0.9814	0.5000
	Original	0.1552	0.0162	0.0000	0.9838	0.5000
	MLSMOTE	0.1541	0.0166	0.0000	0.9834	0.5000
Amazon	MLGAN	0.1321	0.0009	0.4023	0.7094	0.5300
	Original	0.1332	0.0004	0.4028	0.7089	0.5243
	MLSMOTE	0.1329	0.0008	0.4048	0.7076	0.5258
Yeast	MLGAN	0.2533	0.0310	0.3190	0.7584	0.5749
	Original	0.2559	0.0351	0.3250	0.7543	0.5754
	MLSMOTE	0.2546	0.0455	0.3172	0.7580	0.5743

These results determine that MLGAN can accomplish a general improvement in classification results when compared with MLSMOTE. MLGAN, when applied to the flags dataset, shows a very good performance in almost all the multilabel learning approaches. When applied to Yeast and Emotions datasets, improvement in classification results is accomplished compared with MLSMOTE in almost all the multilabel learning approaches. Mirflickr and amazon multilabel dataset shows almost similar classification results on all the multilabel methods. Amazon and Mirflickr are large datasets, so augmenting a particular amount of data doesn't have that much impact on the classification results. The scene dataset shows either good or similar classification results on all the strategies strategies, if not worse. The reason for the scene dataset not showing results as good as demonstrated by other multilabel datasets could be the lack of imbalance in the scene dataset, as is evident from the value of CVIR in Table 2. These results in combination indicate that the classifier has a deplorable influence on the effectiveness of our strategy. So, our strategy MLGAN has successfully reduced the imbalance in a classifier-independent way.

5.2.1 Effect of K in MLSMOTE

MLSMOTE works on the principle of Smoothness assumption; it is assumed that nearby points have the same label. In this study, the value of K (parameter in MLSMOTE) is set to 5. Our proposed approach, MLGAN, doesn't require any parameter k. This is a considerable advantage of MLGAN over MLSMOTE.

Table 5.4: Classification results when k = 3, 5 and 7

Datasets		HammingL	EMR	F1-Measure	RankL	AUC
	k=3	0.0582	0.7879	0.8406	0.1725	0.7564
Scene	k=5	0.0609	0.7796	0.8322	0.1808	0.7575
	k=7	0.0589	0.7858	0.8385	0.1746	0.7509
	k=3	0.3912	0.1610	0.4511	0.5798	0.5927
Emotions	k=5	0.3571	0.1680	0.4439	0.5384	0.5980
	k=7	0.3940	0.1610	0.3940	0.6302	0.5324
	k=3	0.3040	0.2051	0.6619	0.4414	0.4989
Flags	k=5	0.3076	0.1282	0.6496	0.4658	0.5000
	k=7	0.3150	0.2051	0.6713	0.4645	0.4990
	k=3	0.1634	0.0410	0.1152	0.8867	0.4998
MirFlickr	k=5	0.1734	0.0504	0.1845	0.8316	0.5002
	k=7	0.1736	0.0608	0.2190	0.8008	0.4997
	k=3	0.0763	0.4629	0.7626	0.2878	0.6134
Amazon	k=5	0.0759	0.4655	0.7647	0.2848	0.6170
	k=7	0.0759	0.4667	0.7646	0.2847	0.6166
	k=3	0.1992	0.2670	0.6368	0.4058	0.5897
Yeast	k=5	0.1968	0.2727	0.6481	0.4012	0.5841
	k=7	0.1913	0.2608	0.6560	0.3820	0.5976

Table 4 shows the classification results of MLSMOTE when the value of k is set to 3, 5, and 7. The multilabel algorithm used in Table 4 is Label PowerSet. The results are proof that k has a pretty good impact on the classification results in MLSMOTE. Setting the value of K in MLSMOTE would require extra effort. This is a clear advantage of using MLGAN over MLSMOTE for augmenting data in multilabel datasets.

Chapter 6

Conclusion and Future Works

In this dissertation, GAN based solution for the imbalance problem in the multilabel classification has been presented. A review of the previous attempts to handle class imbalance has been given, and how our approach, MLGAN, differs from other techniques. To our knowledge, our method is the first attempt to overcome the imbalance in multilabel datasets using GANs. We take a bag of all the samples belonging to the minority label and synthesize data via GANs using the same bag. In this way, our newly synthesized data helps us overcome the class imbalance problem in such a way that the freshly synthesized sample has similar characteristics with that of all the minority labeled samples and not only with the samples in a particular neighborhood. We performed experiments on six multilabel datasets using five multilabel algorithms. It is worth noting that MLGAN is able to solve the problem of class imbalance in a classifier-independent way. We achieve better classification results when compared with MLSMOTE. Also, in MLGAN, we do not require any parameter K as we do in MLSMOTE. All these facts encourage us to recommend using MLGAN to overcome the class imbalance problem. Multilabel Learning suffers from a common problem of partial label learning. In the future, we will take the partial label learning problem into account while solving the imbalance problem using MLGAN. We will solve the partial multilabel learning problem generated while synthesizing from our approach MLGAN via the Generative Adversarial Network (PMLGAN).

References

[1] M.-L. Zhang and Z.-H. Zhou, "Multilabel neural networks with applications to functional genomics and text categorization," *IEEE transactions on Knowledge and Data Engineering*, vol. 18, no. 10, pp. 1338–1351, 2006. 1

[2] A. K. McCallum, "Multi-label text classification with a mixture model trained by em," in *AAAI 99 workshop on text learning*, Citeseer, 1999. 1

[3] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown, "Learning multi-label scene classification," *Pattern recognition*, vol. 37, no. 9, pp. 1757–1771, 2004. 1

[4] A. Clare and R. D. King, "Knowledge discovery in multi-label phenotype data," in *European conference on principles of data mining and knowledge discovery*, pp. 42–53, Springer, 2001. 1

[5] I. Katakis, G. Tsoumakas, and I. Vlahavas, "Multilabel text classification for automated tag suggestion," in *Proceedings of the ECML/PKDD*, vol. 18, p. 5, Citeseer, 2008. 2

[6] F. Charte, A. Rivera, M. J. d. Jesus, and F. Herrera, "A first approach to deal with imbalance in multi-label datasets," in *International conference on hybrid artificial intelligence systems*, pp. 150–160, Springer, 2013. 2, 7

[7] Y. Du, W. He, Q. Xia, W. Zhou, C. Yao, and X. Li, "Thioether

phosphatidylcholine liposomes: a novel ros-responsive platform for drug delivery," *ACS applied materials & interfaces*, vol. 11, no. 41, pp. 37411–37420, 2019. 2, 7

[8] F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera, "Mlsmote: Approaching imbalanced multilabel learning through synthetic instance generation," *Knowledge-Based Systems*, vol. 89, pp. 385–397, 2015. 2, 3, 7

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014. 2, 4

[10] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional gan," *Advances in Neural Information Processing Systems*, vol. 32, 2019. 2, 5, 24

[11] F. Charte, A. Rivera, M. J. d. Jesus, and F. Herrera, "A first approach to deal with imbalance in multi-label datasets," in *International conference on hybrid artificial intelligence systems*, pp. 150–160, Springer, 2013. 2, 23

[12] B. Liu, K. Blekas, and G. Tsoumakas, "Multi-label sampling based on local label imbalance," *Pattern Recognition*, vol. 122, p. 108294, 2022. 2, 3, 23

[13] M.-L. Zhang, Y.-K. Li, X.-Y. Liu, and X. Geng, "Binary relevance for multi-label learning: an overview," *Frontiers of Computer Science*, vol. 12, no. 2, pp. 191–202, 2018. 3, 30

[14] E. A. Cherman, M. C. Monard, and J. Metz, "Multi-label problem transformation methods: a case study," *CLEI Electronic Journal*, vol. 14, no. 1, pp. 4–4, 2011. 3, 30

[15] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Random k-labelsets for multilabel classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, pp. 1079–1089, July 2011. 3, 31

[16] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 254–269, Springer, 2009. 3, 31

[17] G. Madjarov, D. Gjorgjevikj, and S. Džeroski, "Two stage architecture for multi-label learning," *Pattern Recognition*, vol. 45, no. 3, pp. 1019–1034, 2012. 3, 31

[18] M. S. Sorower, "A literature survey on algorithms for multi-label learning," *Oregon State University, Corvallis*, vol. 18, pp. 1–25, 2010. 3

[19] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining multi-label data," *Data mining and knowledge discovery handbook*, pp. 667–685, 2009. 3

[20] A. Romero, P. Arbeláez, L. Van Gool, and R. Timofte, "Smit: Stochastic multi-label image-to-image translation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0–0, 2019. 4, 10

[21] Z. Cao, R. Wang, X. Wang, Z. Liu, and X. Zhu, "Improving human pose estimation with self-attention generative adversarial networks," in *2019 IEEE international conference on Multimedia & Expo Workshops (ICMEW)*, pp. 567–572, IEEE, 2019. 5

[22] N. Park, M. Mohammadi, K. Gorde, S. Jajodia, H. Park, and Y. Kim, "Data synthesis based on generative adversarial networks," *arXiv preprint arXiv:1806.03384*, 2018. 5

[23] L. Xu and K. Veeramachaneni, "Synthesizing tabular data using generative adversarial networks," *arXiv preprint arXiv:1811.11264*, 2018. 5

[24] G. Y. S. Z. P. N. H. L. H. C. Zhang, Yabin and C. Li, "Partial label learning via generative adversarial nets," pp. pp. 1674–1681, 2020. 5, 10, 18

[25] Y. Yan and Y. Guo, "Adversarial partial multi-label learning with label disambiguation," in *AAAI*, pp. 10568–10576, 2021. 5, 10, 18

[26] Z. Cao, R. Wang, X. Wang, Z. Liu, and X. Zhu, "Improving human pose estimation with self-attention generative adversarial networks," in *2019 IEEE international conference on Multimedia & Expo Workshops (ICMEW)*, pp. 567–572, IEEE, 2019. 5

[27] H. Inoue, "Data augmentation by pairing samples for images classification," *arXiv preprint arXiv:1801.02929*, 2018. 6

[28] 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. 6

[29] F. Maayan, K. Eyal, G. Jacob, and G. Hayit, "Gan-based data augmentation for improved liver lesion classification," *arXiv preprint*, 2018. 6

[30] C. Little, M. Elliot, R. Allmendinger, and S. S. Samani, "Generative adversarial networks for synthetic data generation: A comparative study," *arXiv preprint arXiv:2112.01925*, 2021. 6

[31] S. Dendamrongvit and M. Kubat, "Undersampling approach for imbalanced training sets and induction from multi-label text-categorization domains," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 40–52, Springer, 2009. 7

[32] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014. 10

[33] G. Zhang, Y. Pan, and L. Zhang, "Semi-supervised learning with gan for automatic defect detection from images," *Automation in Construction*, vol. 128, p. 103764, 2021. 10

[34] 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. 10

[35] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *arXiv preprint arXiv:1511.06434*, 2015. 10

[36] Y. Pang, J. Lin, T. Qin, and Z. Chen, “Image-to-image translation: Methods and applications,” *IEEE Transactions on Multimedia*, 2021. 10

[37] T. Karras, S. Laine, and T. Aila, “A style-based generator architecture for generative adversarial networks,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4401–4410, 2019. 10

[38] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, “Self-attention generative adversarial networks,” in *International conference on machine learning*, pp. 7354–7363, PMLR, 2019. 10