Imbalanced Image Classification Using Adaptive Dynamic Oversampling Framework in Deep Feature Space

Sourav Karmakar

Imbalanced Image Classification Using Adaptive Dynamic Oversampling Framework in Deep Feature Space

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
Computer Science

by

Sourav Karmakar
[Roll No: CS-1721]

under the guidance of

Dr. Swagatam Das
Associate Professor
Electronics and Communication Sciences Unit

Indian Statistical Institute
Kolkata-700108, India

July 2019

To my family, friends and my guide

CERTIFICATE

This is to certify that the dissertation entitled **"Imbalanced Image Classification Using Adaptive Dynamic Oversampling Framework in Deep Feature Space"** submitted by **Sourav Karmakar** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Swagatam Das
Associate Professor,
Electronics and Communication Sciences Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA.

Acknowledgments

I would like to show my highest gratitude to my advisor, *Dr. Swagatam Das*, Associate Professor, Electronics and Communication Sciences Unit, Indian Statistical Institute, Kolkata, for his guidance and continuous support and encouragement. He has literally taught me how to do good research, and motivated me with great insights and innovative ideas.

I would also like to thank *Mr. Sankha Subhra Mullick*, Senior Research Fellow, Indian Statistical Institute, Kolkata, for his valuable suggestions and insightful discussions.

My deepest thanks to all the teachers of Indian Statistical Institute, for their valuable suggestions and discussions which added an important dimension to my research work.

Finally, I express my profound gratitude to my mother for her everlasting support and continuous encouragement throughout my years of study.

Last but not the least, I would like to thank all of my friends for their help and support. I thank all those, whom I have missed out from the above list.

Sourav Karmakar
Indian Statistical Institute
Kolkata - 700108 , India.

Abstract

In real world applications, it is very common to encounter data with high class imbalance. Imbalanced dataset is a challenging issue in practical classification problem, as the classifier gets biased towards the majority classes.

The traditional techniques like synthetic minority oversampling have great success in traditional machine learning problems with class imbalance, however these techniques fail to perform well in the field of complex, structured and very high dimensional data like images.

In our work we propose a *novel dynamic oversampling framework*, which is broadly subdivided into three parts. The first step is the representation learning of the dataset, where a Convolutional Neural Network is used to map the raw input training data into a new feature space. In the second step a modified minority oversampling technique is implemented with adaptive k-NN based search between in-class samples in deep feature space. Finally a dense neural classifier is trained on the augmented dataset. To increase the discriminating power of the final classifier we have trained it with modified sample weights.

We have also supplemented our work with empirical studies on publicly available benchmark image datasets and have shown that our technique provides a good countermeasure to handle imbalanced image datasets and provides superior performance than existing techniques.

Keywords: *Imbalanced Classification, Representation Learning, Adaptive-kNN, Minority Oversampling*

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Problem Statement

In general the pattern classification problem can be mathematically viewed as learning a function $g(.)$ from a set of patterns X to a set of class labels $\mathcal{C} = \{1, 2, 3, ..., C\}$ [1, 2]. Each data points in set X is a D-dimensional vector (i.e. $X \subseteq \mathbb{R}^D$). If there are only two classes (i.e. $|\mathcal{C}| = 2$) then it is called *binary classification problem.* If there are more than two classes (i.e.$|\mathcal{C}| > 2$ then it is called *multi-class classification problem.*

A classifier is designed to perform the task of classification [1]. During training a subset of the original dataset $X_{train} \subset X$, called the training dataset are fed to classifier in order to estimate the property of the function $g : X \rightarrow \mathcal{C}$. Let there are total n number of training samples (i.e. $|X_{train}| = n$). At this stage for each training sample $x_i \in X_{train}$, where $i = 1, 2, ..., n$ the true value of $g(x_i)$ are known to the classifier. During testing phase the classifier is expected to predict correct class-label of unseen test samples. i.e. for a new unseen data point $x_j \in X_{test}$, where $j = 1, 2, ..., m$ the classifier is expected to provide correct values of $g(x_j)$. Here, $X_{test} \subset X$ called the test dataset and $|X_{test}| = m$.

Definition 1.1. Imbalanced Classification: Consider a classification problem. If one or few of the classes have significantly less number of training samples as compared to the other classes then it is called a Imbalanced Classification problem. The under-represented classes are called *minority classes* and rest are called *majority classes.*

Class imbalance is *intrinsic* in many classification problems, e.g. medical diagnoses problem where the majority of patients are healthy. *Extrinsic* imbalance, on the other hand, is introduced through external factors, e.g. collection or storage procedures [3].

Let n_c denotes the number of training sample pertaining to class-c ($c \in \mathcal{C}$) and $\sum_{c=1}^{|\mathcal{C}|} n_c = n$. We define *Class Imbalance Ratio* as following.

Definition 1.2. Class Imbalance Ratio (ρ)

$$\rho = \frac{max\ \{n_c \mid c = 1, 2, ..., |\mathcal{C}|\}}{min\ \{n_c \mid c = 1, 2, ..., |\mathcal{C}|\}}$$

Definition 1.3. Let $n_{max} = max\ \{n_c \mid c = 1, 2, ..., |\mathcal{C}|\}$. Then, we define *Minority Classes* ($\mathcal{C}_{min}$) and *Majority Classes* ($\mathcal{C}_{mjr}$) as following:

$$\mathcal{C}_{min} = \{c \mid c \in \mathcal{C}\ and\ n_c \leq \frac{1}{\rho'} \times n_{max}\} \quad and \quad \mathcal{C}_{maj} = \mathcal{C} \setminus \mathcal{C}_{min}\ ,$$

where, ρ' ($\leq \rho$) usually depends on user's discretion and specific to the dataset under consideration.

Images are highly structured, spatially coherent, and complex high dimensional datapoints. Consider a tiny RGB image of spatial dimension 32×32. If we consider each pixel to be a different attribute, then the overall dimensionality of such images is 3072. Convolutional Neural Network (CNN) (sec: 2.1) has proven its excellence in classifying structured data like Images than traditional classifiers [4]. Image dataset also contains intrinsic *long-tail* or *skewed* distribution of classes, e.g. *S*cene *UN*derstanding dataset SUN-397 [5].

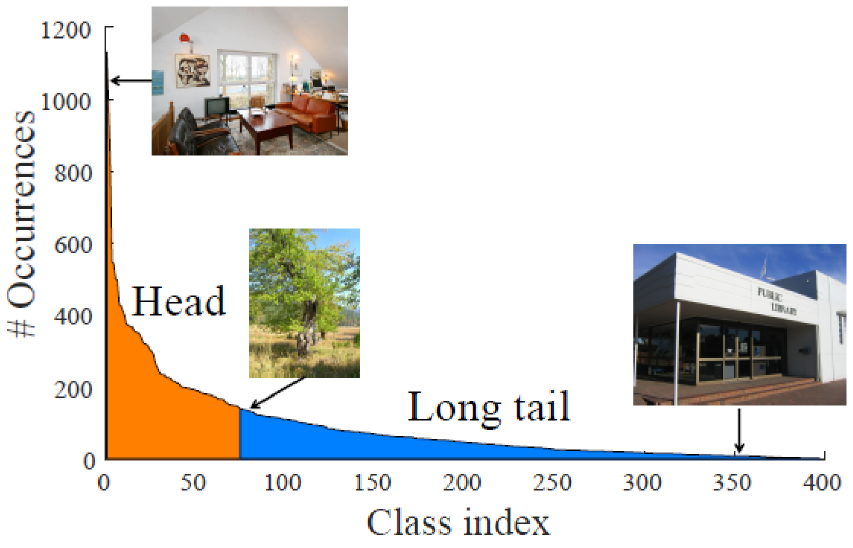


Figure 1.1: Long-tail distribution of SUN-397 dataset
Figure taken from reference: [6]

Despite its success in image classification problem, CNN suffers from the class imbalance issue and like traditional classifiers it is also prone to get biased towards the majority classes, resulting in poor performance in minority classes [7]. Our task is to develop an algorithm that will boost the performance of CNN based image classifier in the presence of class imbalance in the dataset. Let's first see an example to understand the effect of class imbalance in deep learning.

1.2 A Motivational Example

In this section we shall see an example where the performance of CNN gets deteriorated with class imbalance.

1.2.1 Dataset and Imbalanced Settings

MNIST handwritten digit recognition dataset is a very popular benchmarking dataset in machine learning [8]. There are total 10 classes pertaining to 10 digits. The dataset contains 60000 training images and 10000 test images.

The dataset is not inherently imbalanced. We have randomly chosen 4 classes as minority classes (digit-0, 2, 5 & 8). The data from the minority classes are dropped randomly to achieve the desired class imbalance ratio ($\rho = 10, 20, 40$ &100).

Digits	Recall with Balanced Data	Recall with Imbalanced Data			
	$\rho = 1$	$\rho = 10$	$\rho = 20$	$\rho = 40$	$\rho = 100$
0*	0.993	**0.978**	**0.981**	**0.933**	**0.861**
1	0.998	0.998	0.999	0.997	0.993
2*	0.997	**0.933**	**0.930**	**0.823**	**0.804**
3	0.995	0.998	0.991	0.993	0.994
4	0.994	0.999	0.993	0.986	0.994
5*	0.987	**0.950**	**0.946**	**0.879**	**0.826**
6	0.990	0.995	0.992	0.996	0.991
7	0.992	0.988	0.982	0.993	0.998
8*	0.982	**0.927**	**0.899**	**0.899**	**0.846**
9	0.981	0.970	0.991	0.991	0.991

* marks digits belong to minority classes

Table 1.1: Effect of imbalance in handwritten digit recognition using MNIST dataset

1.2.2 Performance and Observation

We have tested the performance of the CNN (sec: 6.1) on imbalanced MNIST data of different imbalance ratio ($\rho = 10, 20, 40$ &100) and compared the performance with the balanced data. The performance metric being *Class Specific Recall* (sec: 2.6). It has been observed (Table 1.1) the steady deterioration of the performance of CNN pertaining to the test samples in minority classes with the increase in class imbalance ratio (ρ).

This example helps us to understand the effect of class imbalance on the performance of CNN.

1.3 Our Contribution

Our contributions are summarized as follows:

- We have proposed a new synthetic oversampling technique for minority classes in *deep feature space* obtained by the CNN's *representation learning.*
- We have analyzed the complexity of our algorithm.
- We have also proposed a sample wise re-weighting scheme to improve the discriminating power of the final classifier.
- We have tested our proposed method for some bench-marking dataset as well as for a new synthetic dataset which was obtained by sub-sampling original ImageNet dataset of ILSVRC'12.

1.4 Thesis Outline

The rest of the thesis is organized as follows:

- In **Chapter-2**, we briefly discussed about the preliminaries, like: CNN, VGG-16 architecture, k-NN and Ada-k-NN algorithm and different performance metrics.
- In **Chapter-3**, we have discussed about the other related work that has been already done in this field.
- In **Chapter-4**, we have described our proposed framework with details.
- In **Chapter-5**, the datasets being used are briefly described along with detailed testing procedure.
- In **Chapter-6**, different CNN architectures used for representation learning are thoroughly discussed.
- In **Chapter-7**, the performance evaluation of our proposed framework is carried out on different datasets.
- Finally, we have concluded our discussion in **Chapter-8** after discussing the scope of future work.

Chapter 2

Preliminaries

2.1 Convolutional Neural Network (CNN)

Convolutional Neural Network (**CNN** or **ConvNet**) [4] is a class of deep neural network mostly applied to visual recognition task, e.g. classification of images. They are also known as *Shift Invariant* or *Space Invariant Artificial Neural Networks* (**SIANN**), based on their shared-weights architecture and translation invariance characteristics. It has shown superior performance over *Multi-Layered Perceptron* [9] in visual recognition task [10]. A Convolutional Neural Network comprises of following building blocks [11].

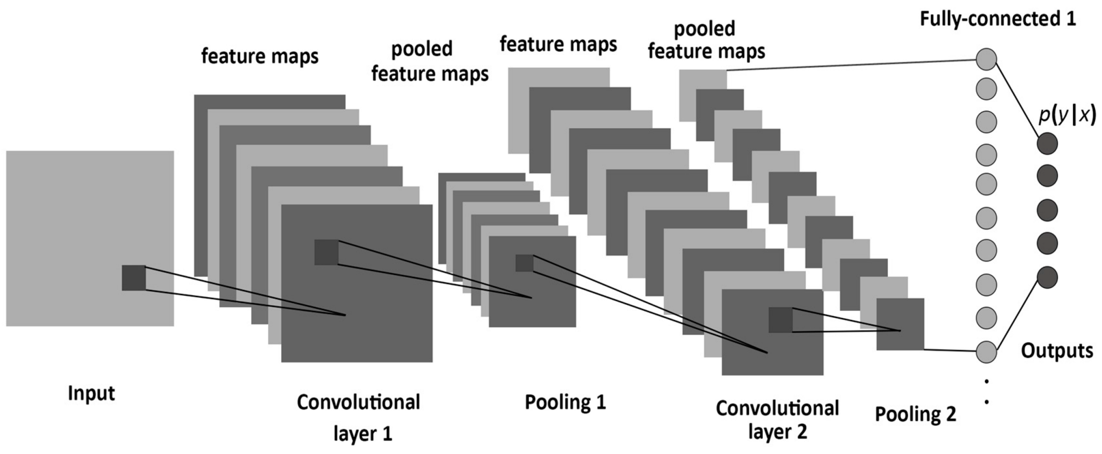


Figure 2.1: Block Diagram of Convolutional Neural Network (CNN)
Source: https://res.mdpi.com/entropy/entropy-19-00242/article_deploy/

- **Convolutional Layer:**
  Convolutional layer is the basic building block of CNN. Each convolutional layer should have the following attributes:
  - Input is a tensor with shape (number of images) × (image width) × (image height) × (image depth).

  - A set of *learnable* Convolutional Kernels whose width and height are hyper-parameters, and whose depth must be equal to that of the image. Convolutional layers convolve across the width and height of the input volume and pass its result to the next layer.

  Each convolution operation is usually succeeded by **ReLU** activation [10].

- **Pooling Layer:**
  Another important concept of CNNs is pooling, which is a form of *non-linear down-sampling*. There are several non-linear functions to implement pooling among which max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum.

- **Fully Connected Layer**
  After several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers (also known as *Dense* layers).

- **Classification Layer**
  This is the final output layer of the CNN where the actual output is compared with the predicted output and the loss is calculated in supervied learning framework. The loss is optimized by updating the learnable parameters through *backpropagation* learning algorithm [9]

2.2 VGG-16 Architecture

VGG-16 [12] is a deep CNN architecture developed by Visual Geometry Group (VGG) at Oxford University in the year 2014 for ILSVRC'14 competition [13].

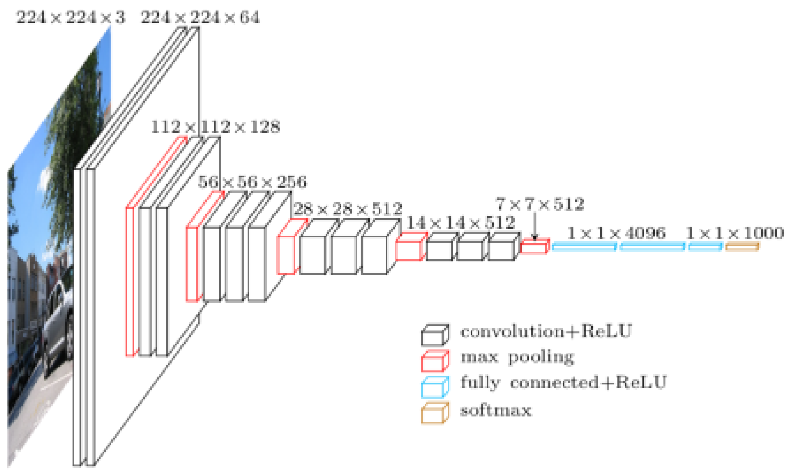


Figure 2.2: How VGG-16 works
Source: http://www.cs.toronto.edu/ frossard/post/vgg16/

It comprises of 16 number of intermediate weight layers and 1 Classification (or *Softmax* layer).

- 13 convolutional layers
- 3 fully connected layers

The model achieves 92.7% top-5 test accuracy in ILSVRC'14 competition. The number of trainable parameters of VGG-16 is approxiately 138.36 million.

2.3 Representation Learning: Supervised Deep Feature Extraction using Deep Neural Network (DNN)

Definition 2.1. *Representation learning* is the process of using machine learning to map raw input data features into a new representation, i.e. a new feature space, for the purpose of improving detection and classification tasks. This mapping from raw input data to new representations is achieved through non-linear transformations of the input data. [14]

Let the original data points belong to D dimensional space. Through representation learning we are learning a function $f(.)$ which shall map (*embed*) the original data points into d dimensional space (also known as *feature space*). Usually $d \ll D$, thus representation learning is a technique used for *dimensionality reduction* and *feature extraction*

The mapping $f : \mathbb{R}^D \to \mathbb{R}^d$ should preserve the neighbourhood information of the original datapoints as much as possible.

Thus with representation learning classification task $g : X \to \mathcal{C}$ (as described in sec: 1.1) becomes a two step process.

- Find the representation of the original datapoints (X) in feature space using function $f(.)$.
$$f : X \to \tilde{X}, \quad (X \subseteq \mathbb{R}^D \, and \, \tilde{X} \subseteq \mathbb{R}^d)$$
- Use these representation of the original dataset ($\tilde{X}$) to learn another function $h(.)$ which shall map these feature space to the set of classes ($\mathcal{C}$).
$$h : \tilde{X} \to \mathcal{C}$$

As data passes through the hidden layers of a *Deep Neural Network* (DNN), it is transformed by each layer into a new representation. Given sufficient data, DNNs are able to learn high-level feature representations of inputs through the composition of multiple hidden layers.[15]

We have employed basic CNN to perform the task of representation learning as shown above. The CNN is trained on labelled training dataset to learn the embedding function $f(.)$ from input feature space to deep feature space. Thus this procedure is also known as *Supervised Feature Extraction.*

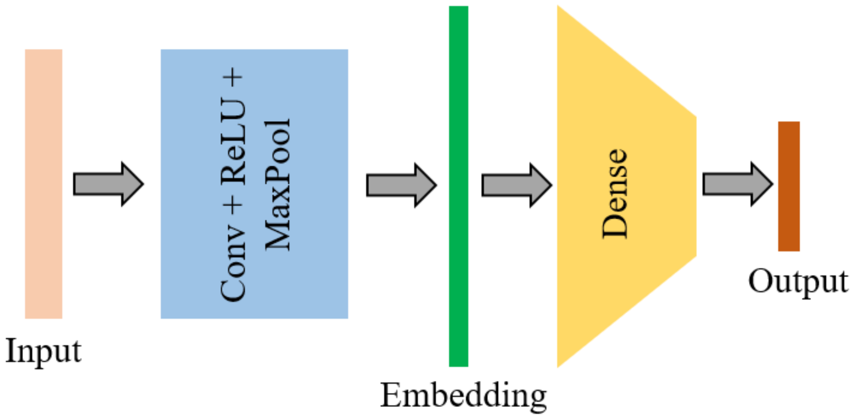


Figure 2.3: Representation Learning by CNN

2.4 The k-Nearest Neighbour Algorithm

k-Nearest Neighbour (k-NN) algorithm is a very well known non-parametric pattern classification algorithm [16]. Because of its simplicity it has found its applications almost in every domain of pattern classification. Following is the algorithm to find the k nearest neighbours of a datapoint x from a dataset S.

Algorithm 1 Get_k_Nearest_Neighbours (x, S, k)

Require: The set of Data S and the point x and number of neighbours k

```
1: Let $|S \setminus \{x\}| = n - 1$
2: $dist = [\,]$                                        ▷ Initialize list of distances
3: for $i = 1$ to $n - 1$ do
4:     $d \leftarrow$ Euclidean Distance between $x$ and $i^{th}$ point of $S \setminus \{x\}$
5:     $dist.append(d)$                  ▷ Adding the distance computed to the end of list
6: end for
7: $neigh \leftarrow argsort(list, k)$   ▷ Sort w.r.t. index and return indices of k-smallest
   values
8: return $neigh$
```

2.5 Class Weighting Scheme

One of the effective way of handling class imbalance is to use class specific weights. Usually in this method the learning algorithm is highly penalized in the event of misclassification of minority class samples. A *folklore* technique is to assign the class weight of a particular class to be inversely proportional to the prior probability of the class. Following *Class Weighting Scheme* (Algorithm: 4) shall be used to compute class weights of some labelled dataset X.

Algorithm 2 Class_Weight_Calculation (X)

Require: The labeled dataset X

```
 1: Let $|X| = n$
 2: Let the set of class labels $\mathcal{C} = \{1, 2, ..., C\}$
 3: Let $g(x)$, $\forall x \in X$ denotes the class label of datapoint $x$, i.e. $g(x) \in \mathcal{C}$
 4: $X_c \leftarrow \{x \mid x \in X \; and \; g(x) = c\} \; \forall c \in \mathcal{C}$
 5: $n_c \leftarrow |X_c|$
 6: Initialize $ClassWeights \in \mathbb{R}^{|\mathcal{C}|}$          ▷ Initialize Global Class Weight vector
 7: $Z \leftarrow 0$                                       ▷ Initialize normalization constant
 8: for $c = 1 \; to \; |\mathcal{C}|$ do
 9:     $p_c \leftarrow (n_c/n)$                         ▷ $p_c$ is the prior probability of class $c \in \mathcal{C}$
10:     $ClassWeights[c] \leftarrow (1/p_c)$
11:     $Z \leftarrow Z + ClassWeights[c]$
12: end for
13: $ClassWeights \leftarrow ClassWeights \; /Z$   ▷ This step normalizes the class weight
    vector
14: return $ClassWeights$
```

2.6 Performance Metrics

Performance of a classifier on a C-class classification problem can be expressed in the form of a matrix called the *Confusion Matrix*, which is defined as follows:

Definition 2.2. A *Confusion Matrix* over a test set X_{test} for a C-class classification problem can be defined as $M_C = [m_{ij}]_{C \times C}$, where m_{ij} represents the number of points which actually belong to i^{th} class but are predicted as a member of class j, $\forall i, j \in \mathcal{C}$. Thus, the diagonal elements (i.e.m_{ii}) are those instances of class i which are correctly classified while the rest are different misclassifications. Each entry in the confusion matrix must be a non-negative integer i.e. $m_{ij} \in \mathbb{Z}^+ \cup \{0\}$; $\forall i, j \in \mathcal{C}$

There are some important properties of the Confusion Matrix as discussed following:

1. The sum of the entries of the confusion matrix is denoted by n_i (i.e. $\sum_{j=1}^{C} m_{ij} = n_i$; $\forall i \in \mathcal{C}$), which is the number of test points belong to the i^{th} class. We assume $n_i > 0$.

2. The sum of the entries in the j^{th} column of the confusion matrix denoted by l_j (i.e. $\sum_{i=1}^{C} m_{ij} = l_j$; $\forall j \in \mathcal{C}$), which is the number of test points predicted as j^{th} class by the classifier.

3. The total number of test points $= \sum_{i=1}^{C} \sum_{j=1}^{C} m_{ij}$; $\forall i, j \in \mathcal{C}$.

To summarize the performance of a classifier we define following performance metrics, whose values are easily obtainable from the Confusion Matrix.

- **Precision:** Precision (denoted by α) for i^{th} class is defined as:

$$\alpha_i = \frac{m_{ii}}{l_i} \; ; \quad \forall i \in \mathcal{C} \, . \tag{2.1}$$

Average Precision is calculated as:

$$\bar{\alpha} = \frac{1}{|\mathcal{C}|} \sum_{i=1}^{|\mathcal{C}|} \alpha_i \, . \tag{2.2}$$

Average Precision for majority and minority classes are calculated as:

$$\alpha_{maj} = \frac{1}{|\mathcal{C}_{maj}|} \sum_{c \, \in \, \mathcal{C}_{maj}} \alpha_c \quad and \quad \alpha_{min} = \frac{1}{|\mathcal{C}_{min}|} \sum_{c \, \in \, \mathcal{C}_{min}} \alpha_c \, . \tag{2.3}$$

- **Recall:** Recall (denoted by β) for i^{th} class is defined as:

$$\beta_i = \frac{m_{ii}}{n_i} \; ; \quad \forall i \in \mathcal{C}. \tag{2.4}$$

Average Recall (also known as *Average Class Specific Accuracy*, ACSA) is calculated as:

$$\bar{\beta} = \frac{1}{|\mathcal{C}|} \sum_{i=1}^{|\mathcal{C}|} \beta_i \, . \tag{2.5}$$

Average Recall for majority and minority classes are calculated as:

$$\beta_{maj} = \frac{1}{|\mathcal{C}_{maj}|} \sum_{c \, \in \, \mathcal{C}_{maj}} \beta_c \quad and \quad \beta_{min} = \frac{1}{|\mathcal{C}_{min}|} \sum_{c \, \in \, \mathcal{C}_{min}} \beta_c \, . \tag{2.6}$$

- **F1 Score:** F1 score for i^{th} class is defined as:

$$F_i = \frac{2\alpha_i \beta_i}{\alpha_i + \beta_i} \, . \tag{2.7}$$

Average F1 score is calculated as:

$$\bar{F} = \frac{1}{|\mathcal{C}|} \sum_{i=1}^{|\mathcal{C}|} F_i \, . \tag{2.8}$$

Average F1 score for majority and minority classes are calculated as:

$$F_{maj} = \frac{1}{|\mathcal{C}_{maj}|} \sum_{c \, \in \, \mathcal{C}_{maj}} F_c \quad and \quad F_{min} = \frac{1}{|\mathcal{C}_{min}|} \sum_{c \, \in \, \mathcal{C}_{min}} F_c \, . \tag{2.9}$$

- **GMean:** GMean (denoted by γ) is calculated as:

$$\gamma = \left(\prod_{i=1}^{|\mathcal{C}|} \beta_i \right)^{\frac{1}{|\mathcal{C}|}} = \left(\prod_{i=1}^{|\mathcal{C}|} \frac{m_{ii}}{n_i} \right)^{\frac{1}{|\mathcal{C}|}} . \tag{2.10}$$

GMean for majority and minority classes are calculated as:

$$\gamma_{maj} = \left(\prod_{c \,\in\, \mathcal{C}_{maj}} \beta_c \right)^{\frac{1}{|\mathcal{C}_{maj}|}} \quad and \quad \gamma_{min} = \left(\prod_{c \,\in\, \mathcal{C}_{min}} \beta_c \right)^{\frac{1}{|\mathcal{C}_{min}|}} . \tag{2.11}$$

- **Accuracy:** Accuracy (denoted by η) is calculated as:

$$\eta = \frac{trace(M_c)}{sum(M_c)} = \frac{\sum_{i=1}^{|\mathcal{C}|} m_{ii}}{\sum_{i=1}^{|\mathcal{C}|} \sum_{j=1}^{|\mathcal{C}|} m_{ij}} \tag{2.12}$$

Chapter 3

Related Works on Deep Learning with Class Imbalance

Addressing class imbalance has been extensively studied over past few decades. Johnson and Khoshgoftaar [14] published a comprehensive review of different techniques used in the context of class imbalance both in the fields of traditional machine learning and deep learning. Researcher have shown that in class imbalanced scenarios, the length of the minority class's gradient component is much smaller than the length of the majority class's gradient component. In other words, the majority class is essentially dominating the net gradient that is responsible for updating the model's weights[17].

All the techniques used in order to increase the classifier's performance in the presence of imbalanced data can be grouped into *Data Level Methods*, *Algorithmic Level Methods* and *Hybrid Methods*.

In this section we shall briefly discuss the different techniques used for handling class imbalance in Deep Learning.

3.1 Data Level Methods

These kind of techniques mainly deals with different methods of *re-sampling* the training data to make it to-some-extent balanced.Various Data Level methods in the context of deep learning are following:

3.1.1 Random Over Sampling (ROS)

In this method all minority classes were over-sampled until class balance was achieved, where any class smaller than the largest class size is considered a minority class. Masko and Hensman [18] explored the effects of class imbalance and ROS using deep CNNs.

3.1.2 Random Under Sampling (RUS)

All majority classes were under-sampled until class balance was achieved, where any class larger than the smallest class size is considered a majority class. RUS performs poorly when compared to baseline as because of under-sampling the classifier looses much of the information, which may be crucial for pattern classification. Buda et al. [19] performed a two phase training procedure where in the first phase the CNN is trained with balanced data produced by RUS and in the second phase the CNN is fine tuned with the original data.

3.1.3 ROS + RUS

Let, ρ_a is the *actual class imbalance ratio* in the dataset and ρ_d is the *desired class imbalance ratio* ($\rho_a > \rho_d$). Then in this method the majority classes were under-sampled and minority classes are over-sampled to achieve the desired imbalance ratio ρ_d. Buda et al. [19] compared ROS and RUS using three multi-class image data sets and deep CNNs.

3.1.4 Synthetic Minority Oversampling TEchnique (SMOTE)

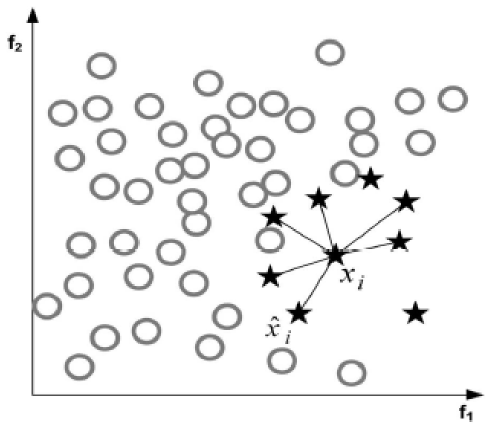
Synthetic Minority Oversampling Technique is a powerful method that has shown a great deal of success in various applications [20]. The SMOTE algorithm creates artificial data based on the feature space similarities between existing minority examples. The SMOTE algorithm in short is described below.

- Let $x_i^{(c)}$ denotes i^{th} sample of a minority class c ($c \in C_{min}$).
- Let $\{x_{i1}^{(c)}, x_{i2}^{(c)}, ..., x_{ik}^{(c)}\}$ denotes a set of k nearest neighbours of $x_i^{(c)}$ for some predefined value k ($k \in \mathbb{N}$).
- Choose one of the k nearest neighbours. Let's call it $\hat{x}_i^{(c)}$.
- For some random number $\delta \in [0, 1]$, a new synthetic sample $\overset{*}{x}_i^{(c)}$ is generated in class c using the following formula:

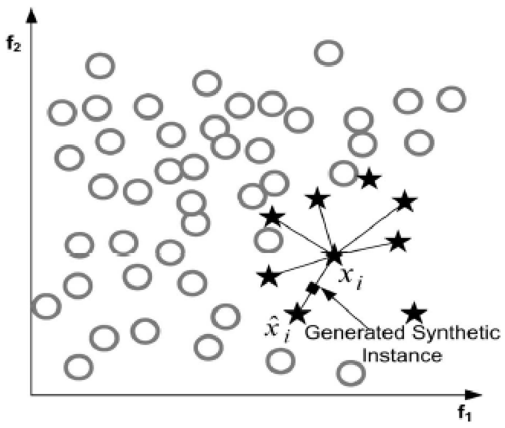
$$\overset{*}{x}_i^{(c)} = x_i^{(c)} + \left(\hat{x}_i^{(c)} - x_i^{(c)}\right) \times \delta. \tag{3.1}$$

The working principle of SMOTE algorithm has been pictorially depicted in figure: 3.1. There are other variants of SMOTE like *borderline*-SMOTE [21] and *safe-level*-SMOTE [22] which improve upon original algorithm.

Khan et al. [23] used SMOTE to compare with the results obtained by their method. We have also used SMOTE to compare with the results produced by our algorithm on different dataset.



(a) Example of k-NN of x_i ($k = 6$).



(b) Synthetic sample generated.

Figure 3.1: How SMOTE works
Figure taken from reference: [3]

3.2 Algorithmic Level Methods

These type of techniques mainly focuses on modifying the model's underlying learning or decision process to increase sensitivity towards the minority group.Various Algorithmic Level methods in the context of Deep Learning are discussed below.

3.2.1 Focal Loss

Lin et al. [24] proposed a model that effectively addresses the extreme class imbalance commonly encountered in object detection problems, where positive foreground samples are heavily outnumbered by negative background samples.

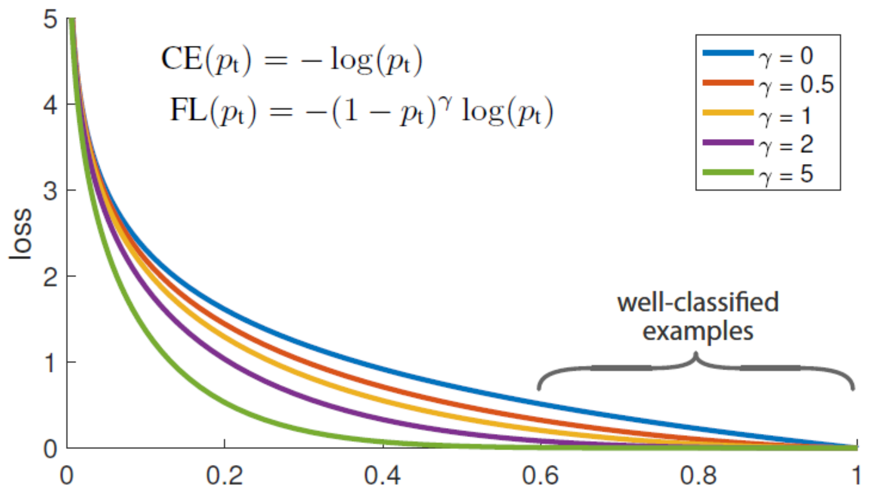


Figure 3.2: Focal Loss for different values of hyper-parameter γ
Figure taken from reference: [24]

To combat the extreme imbalances, *focal loss* re-shapes the cross entropy (CE) loss. This is achieved by multiplying the CE loss by a modulating factor, $\alpha_t(1-p_t)^\gamma$.

$$FL(p_t) = -\alpha_t(1-p_t)^\gamma \log(p_t). \tag{3.2}$$

Hyper parameter $\gamma \geq 0$ adjusts the rate at which majority classes are down weighted, and $\alpha_t \geq 0$ is a class-wise weight that is used to increase the importance of the minority class. Easily classified examples (i.e. the majority class samples), where $p_t \to 1$, cause the modulating factor to approach 0 and reduce the sample's impact on the loss. Though the idea was originally proposed for binary classification problem, the same can be extended for multi-class classification as well.

3.2.2 Cost Sensitive Deep Neural Network

In Cost Sensitive Deep Neural Network (CSDNN) the loss function was modified to incorporate a pre-defined cost matrix, forcing the network to minimize misclassification cost. Cost sensitive method has outperformed many other existing methods of deep learning in case of imbalanced classification. Wang et al. [25] has employed a CSDNN method to detect hospital re-admissions, a class imbalanced problem where a small percentage of patients are readmitted to a hospital shortly after their original visit.

3.3 Hybrid Level Methods

Data level and algorithm level methods are combined in several ways and applied to the problem of imbalance learning. These type of composite methods are labeled as Hybrid Methods. Among various existing Hybrid Level methods following two methods are very useful in the context of deep learning for imblanced classification.

3.3.1 Large Margin Local Embedding (LMLE)

Huang et al. [26] proposed the Large Margin Local Embedding (LMLE) method for learning more discriminative deep representations of imbalanced image data. The method is motivated by the observation that minority groups are sparse and typically contain high variability, allowing the local neighborhood of these minority samples to be easily invaded by samples of another class. By combining a new informed *quintuplet sampling* method with a new *triple-header hinge loss* function, deep feature representations that preserve same class locality and increase inter-class discrimination are learned from imbalanced image data. These deep feature representations, which form well-defined clusters, are then used to label new samples with a fast cluster-wise k-NN classification method. The proposed LMLE method is shown to achieve state-of-the-art results on the CelebA [27] data set, which contains high imbalance levels up to $\rho = 49$.

3.3.2 Deep Over-Sampling Framework (DOS)

Ando and Huang [28] introduced over-sampling to the deep feature space produced by CNNs in their DOS framework. The DOS framework consists of two simultaneous learning procedures, optimizing the lower layer and upper layer parameters separately. The lower layers are responsible for acquiring the embedding function, while the upper layers learn to discriminate between classes using the generated *embeddings*. In order to learn the embedding features, the CNN's input is presented with both a class label and a set of deep feature targets, an in-class nearest neighbor cluster from deep feature space. Then the micro-cluster loss computes the distances between each of the deep feature targets and their mean, constraining the optimization of the lower layers to shift deep feature embeddings towards the class mean.

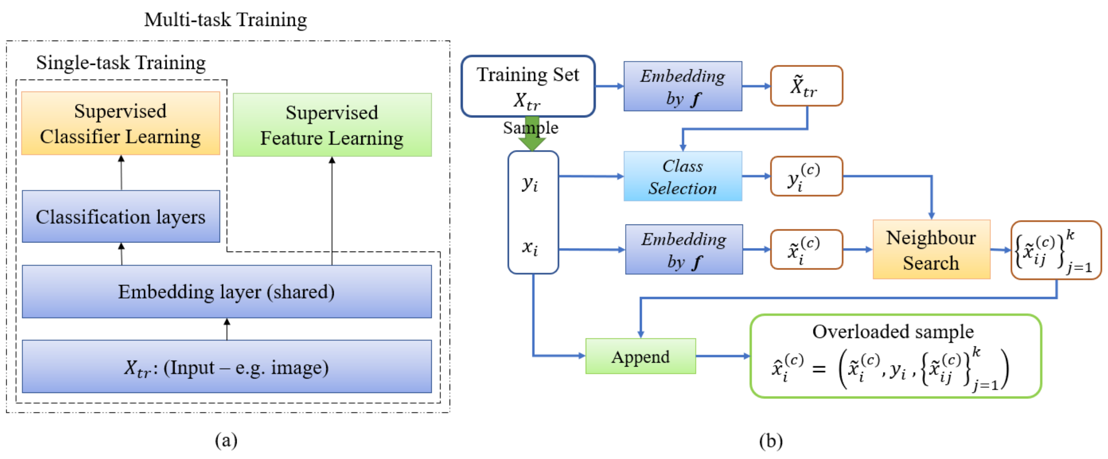


Figure 3.3: The DOS Framework: (a) Basic CNN Architecture for supervised feature learning and (b) The Deep Feature Overloading Framework
Figure adopted from reference: [28]

The deep over-sampling component is the process of selecting k in-class neighbours from deep feature space. In order to address class imbalance, the number of in-class neighbours to select should vary between classes. For example, using $k = 3$ for the minority class, and $k = 0$ for the majority class, will supplement the minority class with additional embeddings while leaving the majority class as is.

Chapter 4

Our Proposed Technique

In this chapter we shall discuss about our proposed framework. We call it ***Adaptive Dynamic Over-Sampling*** **(ADOS)** Framework.

4.1 Mathematical Notations

Following are the list of mathematical notations used to describe our work.

$\mathcal{C}$	Set of Classes
$\mathcal{C}_{maj}$	Set of Majority Classes
$\mathcal{C}_{min}$	Set of Minority Classes
$X_{tr} \in \mathbb{R}^{n \times D}$	Training Dataset
$X_{te} \in \mathbb{R}^{m \times D}$	Test Dataset
n	Number of training samples
m	Number of test samples
n_c	Number of training sample pertaining to class-c, $\forall c \in \mathcal{C}$
n_{max}	$max\,\{n_c : c \in \{1,\ 2, ..., \lvert\mathcal{C}\rvert\}\}$
n_{min}	$min\,\{n_c : c \in \{1,\ 2, ..., \lvert\mathcal{C}\rvert\}\}$
$y_{tr} = \{y_i\}_{i=1}^{n}$	A set of *one-hot-encoded* class labels for training dataset. $y_i \in [0:1]^{\lvert\mathcal{C}\rvert}$, $\forall i \in \{1,\ 2, ..., n\}$
$y_{te} = \{y_i\}_{i=1}^{m}$	A set of *one-hot-encoded* true class labels of test dataset
$\hat{y}_{te} = \{\hat{y}_i\}_{i=1}^{m}$	A set of class labels of test dataset predicted by the classifier. $\hat{y}_i \in (0,1)^{\lvert\mathcal{C}\rvert}$, $\forall i \in \{1,\ 2, ..., m\}$ gives empirical posterior probability distribution $P(c\lvert x_i)$, $where,\ x_i \in X_{te}\ and\,\forall c \in \mathcal{C}$
$f : \mathbb{R}^D \rightarrow \mathbb{R}^d$	Embedding function which maps input feature space of dimension D to extracted deep feature space of dimension d
$h : \mathbb{R}^d \rightarrow (0,1)^{\lvert\mathcal{C}\rvert}$	Final classification function which maps deep feature space to class conditional probability distribution
$g : \mathbb{R}^D \rightarrow (0,1)^{\lvert\mathcal{C}\rvert}$	Overall classifier which maps input feature space to class conditional probability distribution. $g(x) \approx h(f(x))$
$\tilde{X}_{tr} \in \mathbb{R}^{n \times d}$	Embedding of training dataset in deep feature space. This embedding is obtained from embedding function $f(.)$

$\tilde{X}_{te} \in \mathbb{R}^{m\times d}$	Embedding of test dataset in deep feature space This embedding is obtained from embedding function $f(.)$
$k_i^{(c)}$	number of nearest neighbours of i^{th} sample of class-c obtained by *ada*-kNN algorithm
$k_{max}^{(c)}$	Maximum number of nearest neighbours pertaining to any point in class-c
$k_{min}^{(c)}$	Minimum number of nearest neighbours pertaining to any point in class-c
ν_{max}	$max\ \{k_{max}^{(c)} : c \in \{1,\ 2, ..., \lvert\mathcal{C}\rvert\}\}$ (*User Specified*)
ν_{min}	$min\ \{k_{max}^{(c)} : c \in \{1,\ 2, ..., \lvert\mathcal{C}\rvert\}\}$ (*User Specified*)
$\tilde{x}_{ij}^{(c)}$	j^{th} nearest neighbour of i^{th} training sample of class-c, in deep feature space. $j = 1,\ 2, ...,\ k_i^{(c)}$ and $i = 1,\ 2, ...,\ n_c$
$w_{ij}^{(c)}$	Weight of j^{th} nearest neighbour of i^{th} training sample of class-c
$\hat{x}_i^{(c)}$	Generated new neighbour of $x_i^{(c)}$ as combination of other neighbours in deep feature space.
$\overset{*}{x}{}_i^{(c)}$	Generated new sample by $x_i^{(c)}$ in deep feature space
$\tilde{X}_{aug}$	Augmented dataset in deep feature space. This is generated by our proposed framework.

Table 4.1: Mathematical Notations

4.2 Step by Step walk through the Technique

Following is the step by step procedures of our novel oversampling framework.

4.2.1 Training of Supervised Feature Extraction Model

We have already described the architecture of the CNN used for Supervised feature extraction in Chapter 6. To train the CNN we used *weighted categorical cross-entropy* [29] loss function. The weights are calculated using the Class Weighting Scheme as described by the Algorithm 4.

$\lambda = Class\_Weight\_Calculation(X_{tr}),\quad where\ X_{tr}\ is\ the\ original\ training\ dataset$

$\lambda = \{\lambda^{(c)}\}_{c=1}^{\lvert\mathcal{C}\rvert}$ is the l_1-normalized weight vector for $\lvert\mathcal{C}\rvert$ classes. The *weighted categorical cross-entropy* loss function is defined as following:

$$\mathcal{L}_\lambda = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{\lvert\mathcal{C}\rvert}\lambda^{(c)}\ y_i^{(c)}\ \log(\hat{y}_i^{(c)}) \qquad (4.1)$$

The feature extracting CNN learns the *embedding* function $f(.)$ by optimizing the loss through *Back-propagation.* This is also called *Representation Learning.*

4.2.2 Extraction of features

After the feature extracting CNN learns the embedding function $f(.)$ we obtain the embedded representation of training and test datasets.

$$\tilde{X}_{tr} = \{\tilde{x} \mid \tilde{x} = f(x), \, \forall \, x \in X_{tr}\} \quad \mathit{and} \quad \tilde{X}_{te} = \{\tilde{x} \mid \tilde{x} = f(x), \, \forall \, x \in X_{te}\}.$$

CNN as feature extractor

Convolutional neural network is very useful in extracting features from the structured, high-dimensional and spatially coherent data like images. Wiatowski and Bölcskei gave a rigorous mathematical treatise on how complex features embedded in higher dimensional manifold are getting extracted by Deep Convolutional Neural Networks (DCNN) [30]. Wang and Raj also described how complex features are getting classified by non-linear mapping produced by different layers of deep neural network [31].

Motivation behind Feature Extraction

Our proposed algorithm is a k-NN based algorithm. We know that k-NN performs poorly in higher dimensional space (*curse of dimensionality* [32]). It is expected that, k-NN shall perform better in the deep feature space rather than the original input space, as the dimensionality of deep feature space is much less as compared the original input space. Hence, this representation learning followed by deep feature extraction from both training and test samples are necessary.

4.2.3 Adaptive Dynamic Over-Sampling Algorithm

In a nutshell our proposed algorithm does following:

– Find the k numbers of in-class nearest neighbours of a minority class sample. The number of neighbours (k) to consider is point specific and obtained from a linear heuristic inspired from *Ada*-kNN2 method [33].

– Take a weighted combination of these neighbours to create a *synthetic neighbour.*

– New synthetic sample pertaining to that minority class is a random convex combination of the original sample and that synthetic sample.

– Add this new synthetic sample to the augmented dataset and repeat the procedure until certain number of minority samples are generated (this is determined by the desired class imbalance ratio).

Our algorithm *adaptively* chooses the point specific number of nearest neighbours for each minority class samples, generates synthetic samples and also *dynamically* updates the dataset. Hence the name **A**daptive **D**ynamic **O**ver-**S**ampling (ADOS).

Now we shall touch upon the critical aspects of our algorithm.

Motivation for adaptively choosing the number of nearest neighbours

Bhattacharyya and Chakrabarti [34] and Mullick et al. [33] showed that the local density around a point bears a inverse relation to the expected nearest neighbour distance. That means for a particular point lesser value of nearest neighbour distance indicates a *dense locality* around the point, whereas a large nearest neighbour distance indicates *sparse locality*.

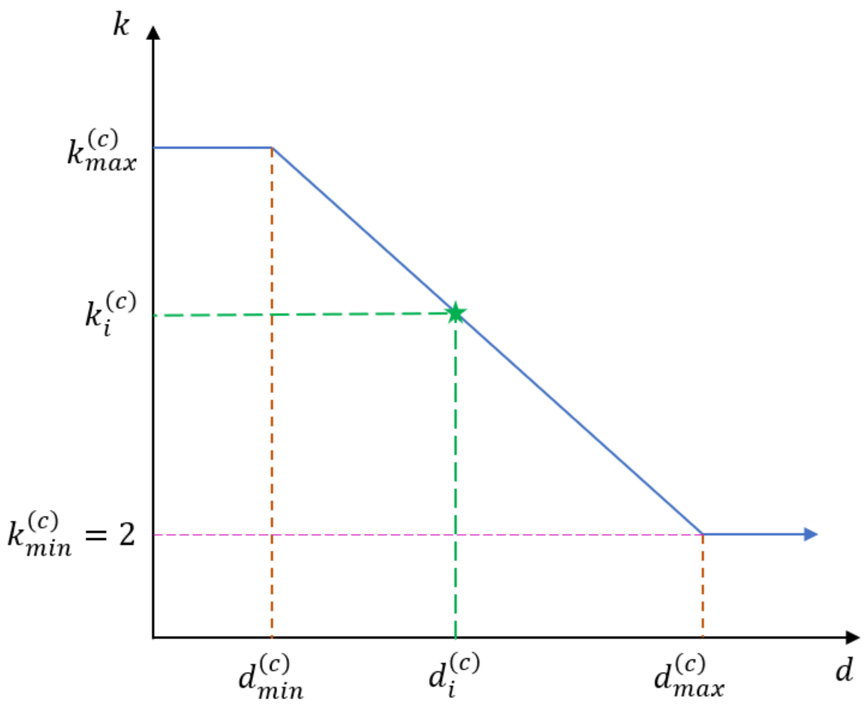


Figure 4.1: Choosing the value of k adaptively using linear heuristic

As the distribution of classes are not known *apriori*, choosing a global k for all points in the dataset stands a risk of ignoring the local distribution of the neighbourhood of each point. Moreover, finding a good global value of k using exhaustive search is almost an impossible task. Hence, a heuristic approach is taken to find-out the more appropriate value of k for each point. In our work, we have considered a linear heuristic, originally proposed by Mullick et al. [33] to find the *locality sensitive* number of nearest neighbours to consider for each point. The heuristic is duly modified to suit the need of our present work.

Once we get the value of parameter k for a specific point we can find out its k nearest neighbours using Algorithm 1. Another supplementary algorithm to compute pairwise distances between the points of a given dataset is shown also below (Algorithm 3).

Algorithm 3 Compute_Pairwise_Distance (S)

Require: The dataset S

```
1:  Let $|S| = n$
2:  Initialize $D \in \mathbb{R}^{n \times n}$                ▷ $D$ is the pairwise distance matrix
3:  **for** $i = 1$ to $n$ **do**
4:      **for** $j = 1$ to $n$ **do**
5:          **if** $i == j$ **then**
6:              $D[i, i] \leftarrow (-1)$
7:          **else**
8:              $D[i, j] \leftarrow$ **Euclidean Distance** *between* $i^{th}$ *and* $j^{th}$ *datapoints*
9:          **end if**
10:     **end for**
11: **end for**
12: **return** $D$
```

Algorithm 4 Get_Number_of_Nearest_Neighbours (x, S, k_{max}, k_{min})

Require: The dataset S, the test-point x, maximum number of neighbours k_{max} and minimum number of neighbours k_{min}

```
1:  Let $D \leftarrow$ Compute-Pairwise-Distance($S$)          ▷ Using Algorithm 2
2:  $d_{min} \leftarrow minimum\{d \mid d \in D \, and \, d > 0\}$
3:  $d_{max} \leftarrow maximum\{d \mid d \in D \, and \, d > 0\}$
4:  $dist = [\,]$                                        ▷ Initialize list of distances
5:  Let $|S \setminus \{x\}| = n - 1$
6:  **for** $i = 1$ to $n - 1$ **do**
7:      $d \leftarrow$ **Euclidean Distance** *between* $x$ *and* $i^{th}$ *point of* $S \setminus \{x\}$
8:      $dist.append(d)$          ▷ Adding the distance computed to the end of list
9:  **end for**
10: Let $d_x = minimum(dist)$
11: **if** $d \leq d_{min}$ **then**
12:     $k = k_{min}$
13: **else if** $d \geq d_{max}$ **then**
14:     $k = k_{max}$
15: **else**
16:     $k = \left\lfloor k_{max} - \frac{(k_{max} - k_{min})}{(d_{max} - d_{min})}(d_x - d_{min}) \right\rfloor$
17: **end if**
18: **return** $k$
```

On choosing the value of $k_{max}^{(c)}$

In our work we have considered that, the maximum number of neighbours we shall look upon for a particular data-point depends on which class it belongs. This approach is particularly important when we have different number of training images in different classes. It is kind of intuitive that we shall look upon more number of neighbours in the class with more number of training samples.

We assume a relationship of the form:

$$k^{(c)}_{max} = a \, . \, n^b_c \, ,$$

where, a & b are some positive constants particular to the dataset.
Now, the following conditions have to satisfied:

$$if \, n_c = n_{max} \; then, \; k^{(c)}_{max} = \nu_{max} \;\; and \;\; if \, n_c = n_{min} \; then, \; k^{(c)}_{max} = \nu_{min}$$

Solving for a & b we get the following values:

$$b = \log\left(\frac{\nu_{max}}{\nu_{min}}\right) \Big/ \log\left(\frac{n_{max}}{n_{min}}\right) \quad and \quad a = \left(\frac{\nu_{max}}{n^b_{max}}\right) \tag{4.2}$$

Putting the values of a and b we get the following formula:

$$k^{(c)}_{max} = \left\lfloor \nu_{max} \times \left(\frac{n_c}{n_{max}}\right)^b \right\rfloor \tag{4.3}$$

In our study we have assumed $\nu_{min} = 10$ and $\nu_{max} = 20$

Example 4.1. Let, there are four classes (viz. 1, 2, 3, 4). Let, the number of training samples per classes are 200, 400, 600 and 1000 respectively. Let, $\nu_{max} = 20$ and $\nu_{min} = 10$.
Thus from equation 6.3, $b = \log(2)/\log(5) \approx 0.431$.
Hence, $k^{(2)}_{max} = \left\lfloor 20 \times \left(\frac{400}{1000}\right)^{0.431} \right\rfloor = 13$. Similarly, $k^{(3)}_{max} = 16$.

Generating Synthetic Sample

$\tilde{x}^{(c)}_i$ is i^{th} sample of class-c in deep feature space. We find its $k^{(c)}_i$ neighbours.
Let, $d^{(c)}_{ij}$ is the euclidean distance between $\tilde{x}^{(c)}_i$ and its j^{th} neighbour $\tilde{x}^{(c)}_{ij}$.

$$i.e. \; d^{(c)}_{ij} = \left\| \tilde{x}^{(c)}_i - \tilde{x}^{(c)}_{ij} \right\|_2 , \quad for \, j = 1, \, 2, \, ..., \, k^{(c)}_i . \tag{4.4}$$

Let, $d^{(c)}_{i_{max}} = max \left\{ d^{(c)}_{ij} \mid j = 1, 2, ...k^{(c)}_i \right\}$. Then, $w^{(c)}_{ij}$, the weight associated with j^{th} neighbour of i^{th} sample of class-c is:

$$w^{(c)}_{ij} = \frac{1}{Z} exp\left(- d^{(c)}_{ij} / d^{(c)}_{i_{max}} \right) , \quad for \, j = 1, \, 2, \, ..., \, k^{(c)}_i . \tag{4.5}$$

Z is a l_1-normalization constant. i.e. $Z = \sum_{j=1}^{k^{(c)}_i} exp\left(- d^{(c)}_{ij} / d^{(c)}_{i_{max}} \right)$.

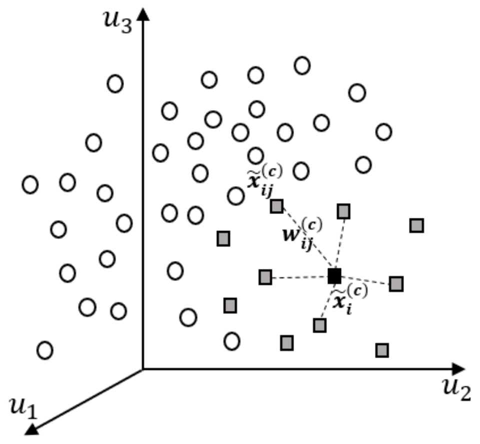
We now generate a *synthetic neighbour* in deep feature space pertaining to i^{th} sample of class-c as weighted combination of its neighbours:

$$\hat{x}_i^{(c)} = \sum_{j=1}^{k_i^{(c)}} w_{ij}^{(c)} \, \tilde{x}_{ij}^{(c)}. \tag{4.6}$$

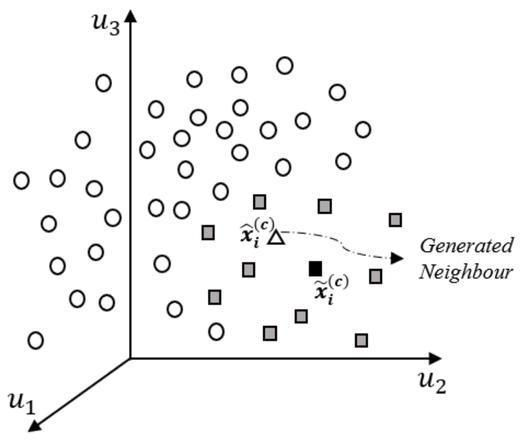
Hence further the neighbour, lesser will be its contribution to generate this *synthetic neighbour* because of the weight associated with it.

Now for some random value $\delta \in [0, 1]$ the new generated synthetic sample $\left(\overset{*}{x}_i^{(c)}\right)$ in deep feature space corresponding to i^{th} sample of class-c is:

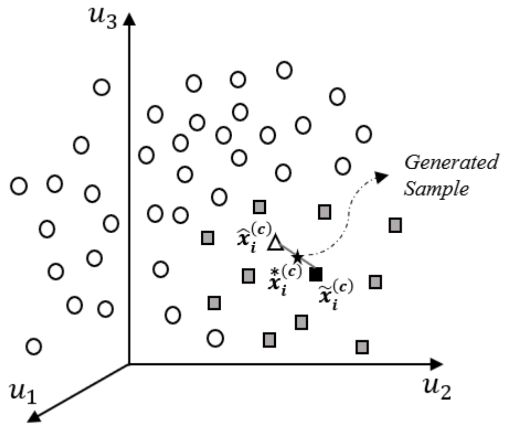
$$\overset{*}{x}_i^{(c)} = \delta \, \tilde{x}_i^{(c)} + (1 - \delta) \, \hat{x}_i^{(c)}. \tag{4.7}$$



(a) Example of $k_i^{(c)}$-NN of $\tilde{x}_i^{(c)}$.



(b) Synthetic neighbour generated.



(c) Synthetic sample generated.

Figure 4.2: The working principle of ADOS

This synthetic sample is augmented to the dataset. Generation of synthetic sample in our proposed framework is pictorially depicted in Figure 4.2. The complete algorithm is shown below.

Algorithm 5 Adaptive_Dynamic_Over_Sampling

Require: $\tilde{X}_{tr}$, y_{tr}, $\mathcal{C}$, Number of samples in each class $\{n_c\}_{c=1}^{|\mathcal{C}|}$, *Actual Imbalance Ratio* (ρ_a), *Desired Imbalance Ratio* (ρ_d)

```
1:  $\tilde{X}_{aug} \leftarrow \tilde{X}_{tr}$                                  ▷ Initialize the augmented dataset
2:  $y_{aug} \leftarrow y_{tr}$                                                   ▷ Initialize the augmented labels
3:  $n_{max} \leftarrow max\ \{n_c\}$
4:  $\mathcal{C}_{min} \leftarrow \left\{c \mid n_c \leq \frac{n_{max}}{\rho_d}\right\}$        ▷ The set of minority classes
5:  for $c$ in $\mathcal{C}_{min}$ do
6:      $n_{gen}^{(c)} = \left\lceil \frac{n_{max}}{\rho_d} \right\rceil - n_c$        ▷ No. of samples to be generated for class-c
7:      $\tilde{X}_{tr}^{(c)} \leftarrow \{\tilde{x} \mid \tilde{x} \in \tilde{X}_{tr} \land \tilde{x}$ belongs to class-$c\}$
8:      $\tilde{X}_{gen}^{(c)} \leftarrow \emptyset$                             ▷ Initialize generated set of samples of class-c
9:      $y_{gen}^{(c)} \leftarrow \emptyset$                                     ▷ Initialize generated labels of class-c
10:     Calculate $k_{max}^{(c)}$ using the the Equation 6.3
11:     for $i = 1$ to $n_{gen}^{(c)}$ do
12:         $k_i^{(c)} \leftarrow$ Get_Number_of_Nearest_Neighbours $(x_i^{(c)}, \tilde{X}_{tr}^{(c)} \bigcup \tilde{X}_{gen}^{(c)}, k_{max}^{(c)}, 2)$
13:         $\left\{\tilde{x}_{ij}^{(c)}\right\}_{j=1}^{k_i^{(c)}} \leftarrow$ Get_$k$_Nearest_Neighbours$(\tilde{x}_i^{(c)}, \tilde{X}_{tr}^{(c)} \bigcup \tilde{X}_{gen}^{(c)}, k_i^{(c)})$
14:         Calculate $\{w_{ij}\}_{j=1}^{k_i^{(c)}}$ using Equation 6.5
15:         Calculate $\hat{x}_i^{(c)}$ using Equation 6.6              ▷ Generate synthetic neighbour
16:         $\delta \leftarrow Random[0, 1]$                             ▷ A random number between 0 to 1
17:         Calculate $\overset{*}{x}_i^{(c)}$ using Equation 6.7              ▷ Generate synthetic sample
18:         $\tilde{X}_{gen}^{(c)} \leftarrow \tilde{X}_{gen}^{(c)} \bigcup \left\{\overset{*}{x}_i^{(c)}\right\}$
19:         $y_{gen}^{(c)} \leftarrow y_{gen}^{(c)} \bigcup one\_hot\_encode(c)$
20:     end for
21:     $\tilde{X}_{aug} \leftarrow \tilde{X}_{aug} \bigcup \tilde{X}_{gen}^{(c)}$
22:     $y_{aug} \leftarrow y_{aug} \bigcup y_{gen}^{(c)}$
23: end for
24: return $\tilde{X}_{aug}$, $y_{aug}$
```

4.2.4 Training of final classifier

The final classifier is a neural classifier with input dimension same as the dimension of the feature space and the final dimension is number of classes.

We train the neural classifier with a proposed *sample specific re-weighting scheme* which is explained below.

Sample Specific re-weighting scheme

Let, the number of training samples in a minority class-$c (c \in \mathcal{C})$ before oversampling is n_c and after oversampling is n'_c. Let, $\sum_{c \in \mathcal{C}} n_c = n$ and $\sum_{c \in \mathcal{C}} n'_c = n'$

Then prior probability of class-c before oversampling is $p_c = n_c/n$ and after oversampling the same is $p'_c = n'_c/n'$

Now, we have already discussed that a *folklore* technique is to assign the class weight of a particular class to be inversely proportional to the prior probability of the class (sec: 2.5). Hence before oversampling the weight of particular minority class-c is $w_c \propto p_c$ and after oversampling the same is $w'_c \propto p'_c$.

We propose that all the original sample of minority class-c shall get sample specific weight equal to w_c (i.e. the class weight before oversampling) and all the new synthetically generated sample shall have sample specific weight equal to w'_c (i.e. the class weight after oversampling). Following is an example to better understand the proposed sample weighting scheme.

Example 4.2. Consider a binary classification problem with one class having 2000 samples and other class contains 100 samples producing an actual class imbalance ratio $(\rho_a) = 20$. Let, our desired class imbalance ratio (ρ_d) is 4.

Then we have to synthetically generate additional 400 samples for class-2.

Now *prior probabilities* of the classes before oversampling are: $p_1 = \frac{20}{21}$ *and* $p_2 = \frac{1}{21}$

Hence the class-weights *(normalized)* prior to oversampling are:

$$w_1 = \frac{21/20}{(21/20) + (21)} = \frac{1}{21} \quad and \quad w_2 = \frac{21}{(21/20) + (21)} = \frac{20}{21}$$

Prior probabilities of the classes after oversampling is: $p'_1 = \frac{4}{5}$ *and* $p'_2 = \frac{1}{5}$

Hence the class-weights *(normalized)* after oversampling are:

$$w'_1 = \frac{5/4}{(5/4) + (5)} = \frac{1}{5} \quad and \quad w'_2 = \frac{5}{(5/4) + (5)} = \frac{4}{5}$$

According to our *sample specific re-weighting scheme* all the newly generated samples of class-2 shall have weights w'_2 while the original samples of class-2 will have the same weights w_2. The weights of the samples of class-1 remain unchanged.

The *intuition* behind this technique is we are assigning lesser importance to the synthetically generated samples as compared to the original one in minority classes. This type of weighting scheme is particularly important when the synthetic samples in the minority class has high variance.

4.3 Complexity Analysis of Our Proposed Method

Here we shall discuss about the *time complexity* of our proposed algorithm Adaptive Dynamic Over-Sampling technique.

Assumptions

We have made following assumptions while analysing the complexity of our proposed oversampling technique.

– Let us consider a binary classification problem.

– The positive class is denoted as $\mathcal{C}_+$ and the negative class is denoted as $\mathcal{C}_-$.

– $|\mathcal{C}_+| = n$ and $|\mathcal{C}_-| = N$.

– In the context of class imbalance let us further assume, $N > n$. Class imbalance ratio $\rho = \frac{N}{n}$.

– The dimensionality of deep feature space is d.

– In our algorithm we find the number of nearest neighbours specific to a point using the *linear heuristic* described in our algorithm. The exact value of k is difficult to determine analytically without knowing the distribution of the data points in deep feature space. Hence we consider $\hat{k}$ as *expected* value of the number of nearest neighbours for all points of minority class.

$$\hat{k} = \mathbb{E}[k].$$

The value of $\hat{k}$ shall lie between k_{max} and k_{min} which are user defined parameters.

– We shall consider the *desired imbalance ratio* = 1. That means after oversampling both the classes shall contain same number of samples.

Complexity of the generation of first synthetic sample

To generate first synthetic sample of minority class, we have to go through the following steps.

– Calculation of distance matrix. Complexity of this step is $\mathcal{O}(n^2d)$, as there are n number of d -dimensional data-points in the minority class (*in deep feature space*). We have assumed that claculating distance between two points in d-dimensional space takes $\mathcal{O}(d)$ time.

– To find $\hat{k}$ number of nearest neighbours from the distance matrix will take $\mathcal{O}(n\hat{k})$ time.

– Weight calculation of each individual neighbours shall take $\mathcal{O}(\hat{k})$ time.

– New synthetic point generation. Complexity of this step is $\mathcal{O}(\hat{k}d)$.

Complexity of the generation of subsequent synthetic samples

Now after we augment the dataset with this newly created sample we need not to recompute the distance matrix again. We can simply append a *row* and a *column* to the already calculated distance matrix. Hence, assuming that there are already t many samples in the minority class at some point $(n < t < N)$ following steps are required to create a new synthetic minority sample.

– Appending a row and column to the already calculated distance matrix, takes $\mathcal{O}(td)$ time.

– To find $\hat{k}$ number of nearest neighbours from the distance matrix will take $\mathcal{O}(t\hat{k})$ time.

– Weight calculation of each individual neighbours shall take $\mathcal{O}(\hat{k})$ time.

– New synthetic point generation. Complexity of this step is $\mathcal{O}(\hat{k}d)$.

Note that, the last two operations are same for all points.

Complexity of total algorithm

Hence, the total time complexity of our proposed algorithm is:

$$\mathcal{O}(n^2d) + \sum_{t\,=\,n+1}^{N-1} \mathcal{O}(td) + \sum_{t\,=\,n}^{N-1} \mathcal{O}(t\hat{k}) + \sum_{t\,=\,n}^{N-1} \Big(\mathcal{O}(\hat{k}) + \mathcal{O}(\hat{k}d)\Big)$$

Now, $\mathcal{O}(n^2d) + \sum_{t\,=\,n+1}^{N-1} \mathcal{O}(td) = \mathcal{O}(N^2d)$.
Now if we assume $N \gg n$, which is very reasonable in the context of high class imbalance, we have

$$\sum_{t\,=\,n}^{N-1} \mathcal{O}(t\hat{k}) \approx \mathcal{O}(N^2\hat{k}) \quad and \quad \sum_{t\,=\,n}^{N-1} (\mathcal{O}(\hat{k}) + \mathcal{O}(\hat{k}d)) \approx \mathcal{O}(N\hat{k}d)$$

.

Finally, the overall complexity reduces to: $\mathcal{O}(N^2d) + \mathcal{O}(N^2\hat{k}) + \mathcal{O}(N\hat{k}d)$. With further reasonable assumptions like, $\hat{k} \ll d \ll N$, the complexity of the algorithm further reduces to:

$$\mathcal{O}(N^2d),$$

where, N is the number of data-points in the majority class ($\approx$ total number of training data-points, assuming high class imbalance) and d is the dimension of the feature space.

Chapter 5

Description of the Datasets

To test our models we have used various datasets. Following are the description of the datasets.

5.1 MNIST-back-rotation Images (MNIST*rb*)

The MNIST-back-rotation-image (MNIST*rb*) [35] is an extension of the MNIST dataset contains 28×28 gray-scale images of rotated digits over randomly inserted backgrounds. The default training and test set consist of 12000 and 50000 images, respectively.

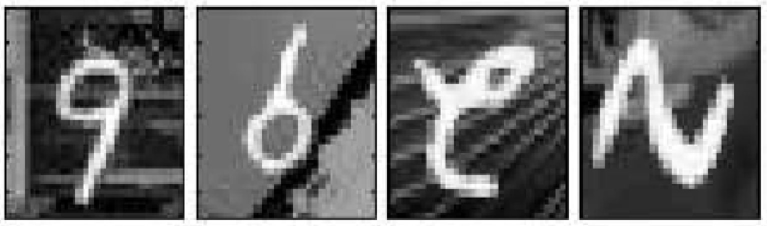


Figure 5.1: A glimpse of MNIST*rb* dataset
Figure taken from reference: [35]

5.1.1 Data Augmentation

As the number of training samples are less, we have augmented each of the images of MNIST*rb* training dataset with additional 9 images produced by different transformations (like: rotation, flipping etc.) of the original image. After augmentation the training dataset contains 120000 images.

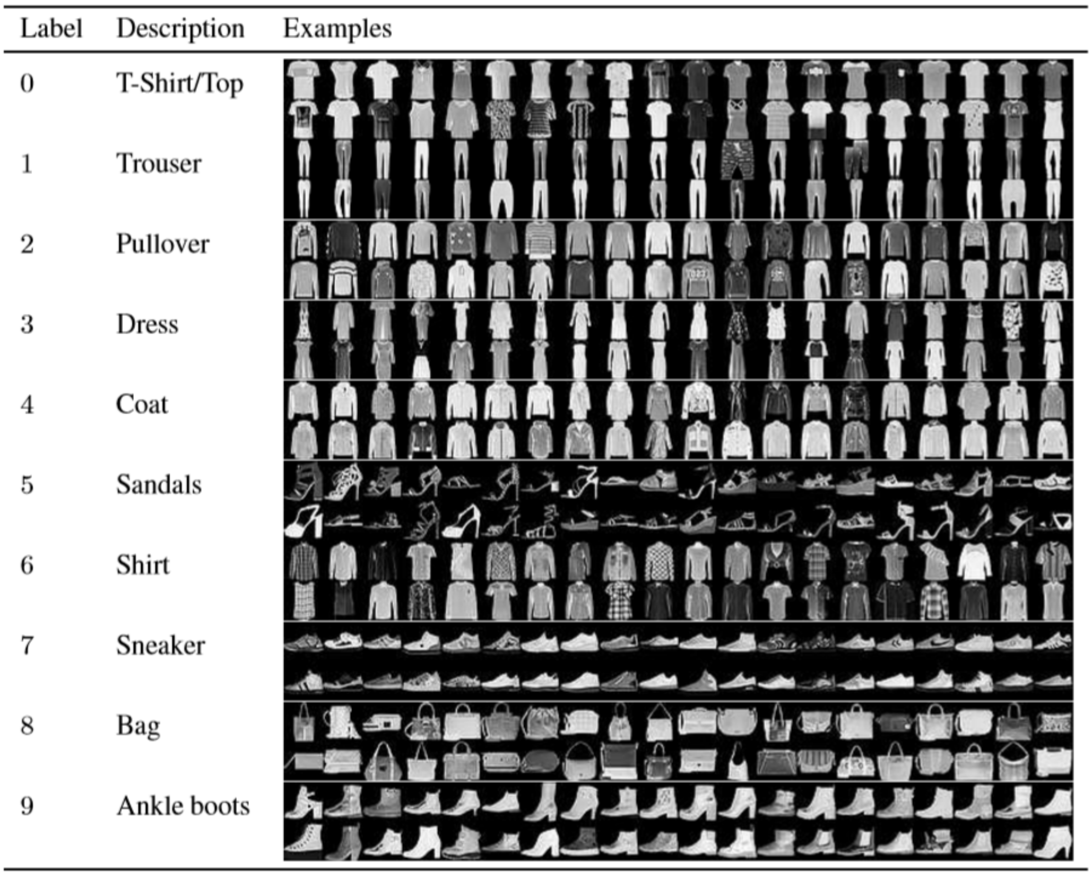
5.1.2 Imbalanced Settings

This augmented dataset as well as the original MNIST*rb* dataset is not inherently imbalanced. To make it imbalanced we have randomly chosen 4 classes out of

10 as minority class. We have randomly dropped the training samples from minority classes to make achieve desired imbalance ratio (ρ). Thus we have made imbalanced dataset of $\rho = 10,\ 20,\ 40\ \mathit{and}\ 100$ from augmented MNISTrb dataset.

5.2 Fashion-MNIST Dataset

Fashion MNIST [36] is a MNIST like dataset of gray-scale images of dimension 28×28. Like MNIST, it also consists of 10 classes, 60000 training and 10000 test images. However, as the name suggests the classes in Fashion-MNIST dataset are fashion items.



Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Figure 5.2: Class names and example images of Fashion MNIST dataset
Figure taken from reference: [36]

5.2.1 Imbalanced Settings

Fashion-MNIST dataset is also not inherently imbalanced. We make it imbalanced by first selecting 4 minority classes, followed by removing random samples from the minority classes to achieve predefined imbalance ratio (ρ). Thus we have prepared imbalanced dataset of $\rho = 10,\ 20,\ 40\ \mathit{and}\ 100$ from Fashion-MNIST dataset.

5.3 CIFAR-10 Dataset

The CIFAR-10 dataset [37] consists of 32 × 32 RGB images. A total of 60,000 images in 10 categories are split into 50,000 training and 10,000 testing images.

5.3.1 Imbalanced Settings

Like the previous two datasets described CIFAR-10 dataset is also originally a balanced dataset. We make it imbalance by first choosing 4 minority classes and randomly removing images from the minority classes until we achieve certain imabalnce ratio (ρ). We have prepared imbalanced CIFAR-10 dataset of $\rho = 10,\ 20,\ 40\ and\ 100$.

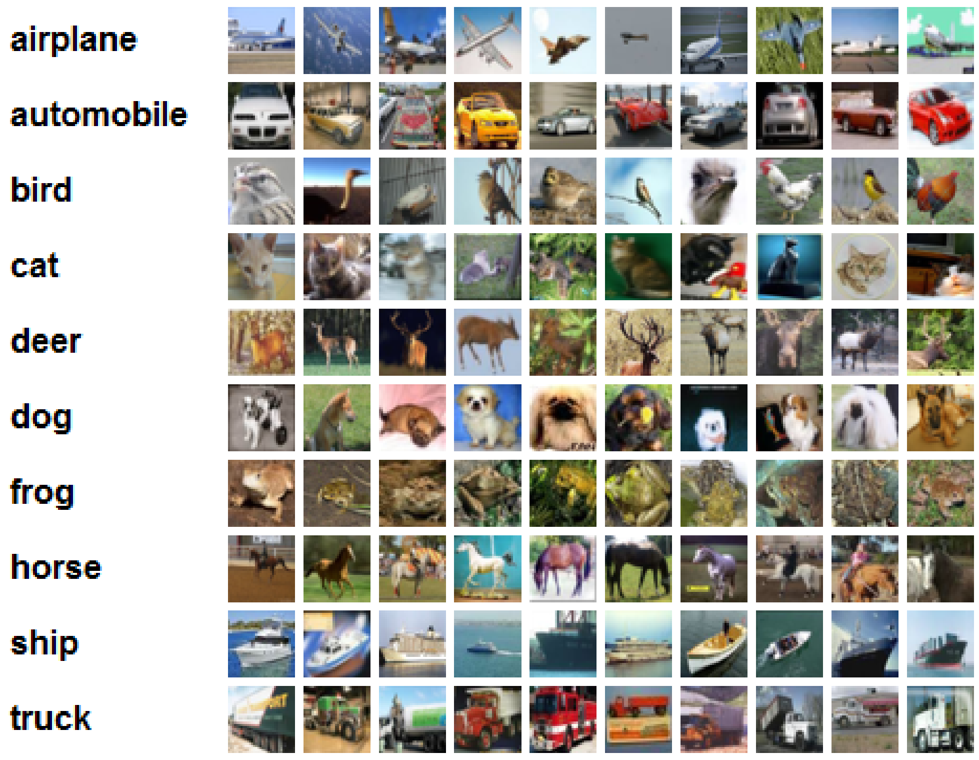


Figure 5.3: Class names and example images of CIFAR-10 Dataset
Source: http://karpathy.github.io/2011/04/27/manually-classifying-cifar10/

5.4 Street View House Number (SVHN) Dataset

The Street View House Numbers (SVHN) dataset [38] consists of 73,257 images for training and 26,032 images for testing. The images are in 32 × 32 RGB format. There are 10 classes, 1 for each digit. Digit '1' has label 1, '9' has label 9 and '0' has label 10.

Figure 5.4: A glimpse of SVHN dataset
Source: https://sigopt.com/blog/unsupervised-learning-with-even-less-supervision-using-bayesian-optimization/

5.4.1 Imbalanced Settings

The class distribution of SVHN dataset shows that it is not inherently balanced. The highest number of images belong to class-1 (13861) and the lowest number of images belong to class-9 (4659), producing an inherent class imbalance ratio $\rho = \frac{13861}{4659} \approx 3$.
However for our work we have taken 4 number of randomly chosen classes as minority classes and reduced images from those classes randomly until we achieve specified imbalance ratio (like $\rho = 10,\ 20,\ 40\ and\ 100$)

Remark. For our convenience we have re-labelled digit '0' to 0.

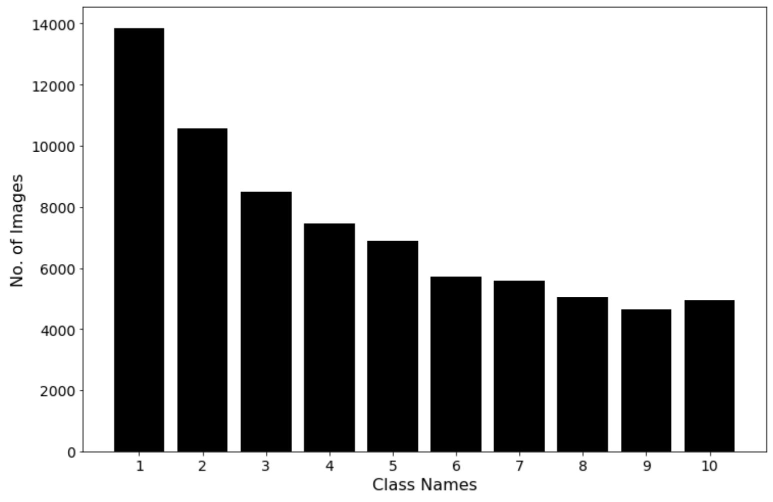


Figure 5.5: Class Distribution of SVHN Dataset

5.5 Re-sampled ImageNet Dataset

This dataset is created as an artificial benchmark data to demonstrate the scalability of our algorithm. Original ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [13] consists of around 14 million images distributed into 1000 classes.

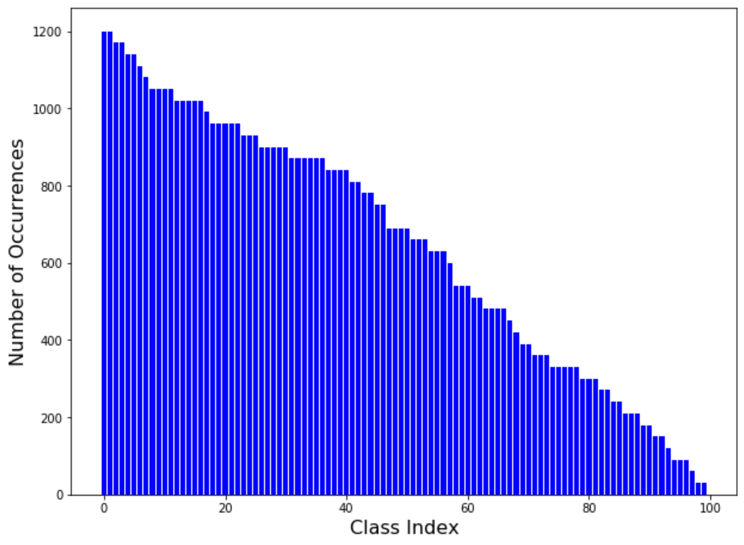


Figure 5.6: Class distribution of Re-sampled ImageNet dataset

We have randomly sampled a 100 classes out of those 1000 classes. To create imbalance we have randomly selected from 30 to 1200 samples in each classes in the multiple of 30. Fig: 5.6 shows the class distribution.

There are total 64,740 number of training samples in this dataset. We have *manually selected* 2000 test samples (20 samples per class). The inherent class imbalance ratio of this dataset is $\frac{1200}{30} = 40$.

Chapter 6

Neural Network Architectures and Training

In this chapter we shall discuss about the deep neural network *Architectures* that we are going to use for our study. Our whole process involves three steps as discussed following.

- First step is ***Supervised Feature Learning***. A CNN is used for this purpose (fig: 6.1). After learning of features we have to extract the features from *training* and *test* dataset using the trained CNN.
- Second step is ***populate more synthetic samples*** using our proposed algorithm (Chapter: 4)
- The final step being ***to build a classifier*** and train it with our populated dataset.

As discussed in the following sections the models used as feature extraction are different for different datasets.

6.1 Feature Extracting Architecture for MNIST*rb* and Fashion-MNIST Dataset

The feature extraction models for these datasets are taken from the Deep Over-Sampling (DOS) paper and we used the same setting as discussed in that paper[28]. The architecture of the CNN that used as feature extracting model for these datasets is pictorially described in the figure: 6.2.

The input layer of this CNN accepts a batch of gray-scale (single-channel) images of spatial dimension 28×28. Followed by input layer there is a **Convolutional Block** which consists of a *Convolutional Layer* with number of output filters = 6 and filter dimension: 5×5, followed by a *Batch Normalization*, *Activation (ReLU)*, and *Max Pool Layer* (with *stride:* 2).

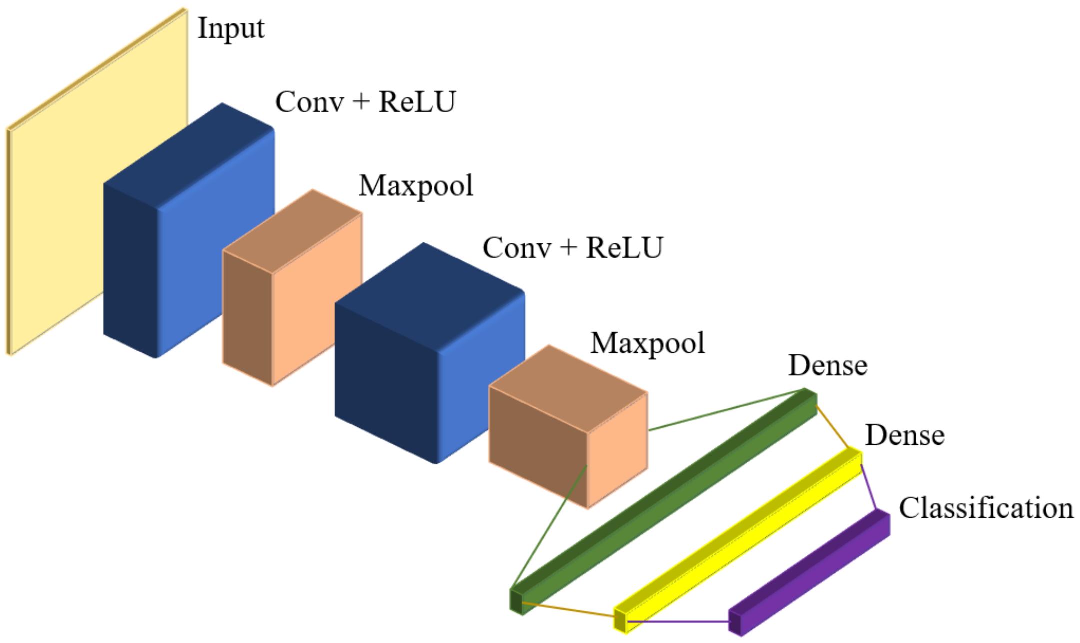


Figure 6.1: CNN used as Supervised Feature Extractor

After this, there is another **Convolutional Block** with number of output filters = 16. Then output of final Max-Pool layer is flattened and connected to a *Dense* (or *Fully Connected*) layer of dimension 120. This layer is our **Embedding Layer**. This layer is finally connected to *Classification Layer* of dimension 10, as there are 10 number of classes.

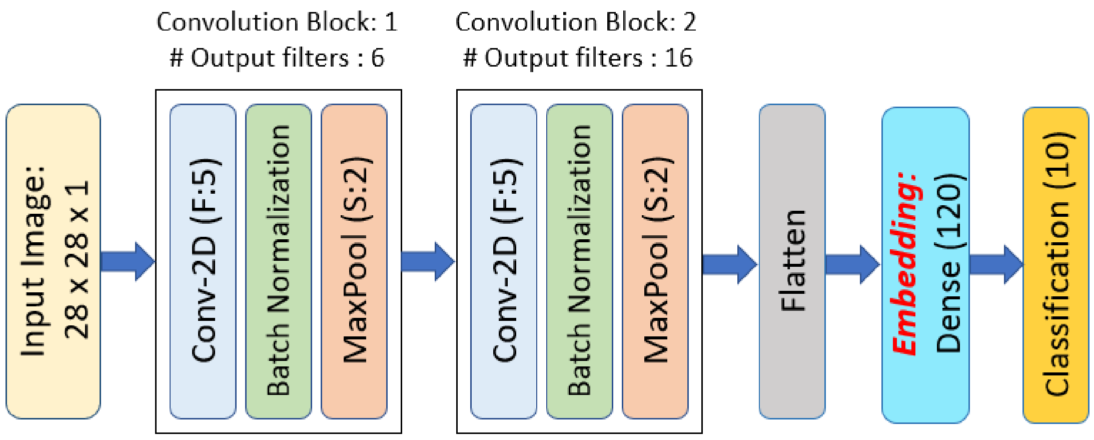


Figure 6.2: Supervised Feature Extracting CNN for MNIST*rb* and Fashion-MNIST dataset

The loss function is computed at the final classification layer. We shall brief about the loss function at the chapter: 4. The loss function is optimized using *Back-propagation* [9] learning algorithm. There are total 52186 number of trainable parameters in this model.

6.2 Feature Extracting Architecture for CIFAR-10 and SVHN Dataset

The architecture of the CNN that used as feature extracting model for these dataset is pictorially described in the figure: 6.3.

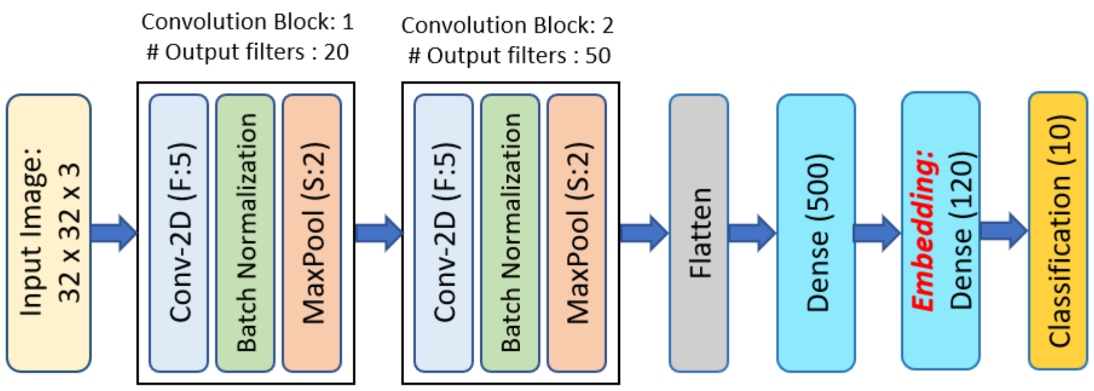


Figure 6.3: Supervised Feature Extracting CNN for CIFAR-10 and SVHN dataset

The input layer of this CNN accepts a batch of RGB (3-channels) images of spatial dimension 32×32. Followed by input layer there is a **Convolutional Block** which consists of a *Convolutional Layer* with number of output filters = 20 and filter dimension: 5×5, followed by a *Batch Normalization*, *Activation (ReLU)*, and *Max Pool Layer* (with *stride:* 2).

After this, there is another **Convolutional Block** with number of output filters = 50. Then output of final Max-Pool layer is flattened and connected to a *Dense* (or *Fully Connected*) layer of dimension 500. Which is then connected to another *Dense* layer of dimension 120. This layer is our **Embedding Layer**. This layer is finally connected to *Classification Layer* of dimension 10, as there are 10 number of classes.

There are total $7, 14, 780$ no. of trainable parameters in the model.

6.3 Feature Extracting Architecture for Re-sampled ImageNet Dataset

We have used fine-tuned model of VGG16 architecture (sec: 2.2) to extract the features from the images of this dataset. We have modified the top layer of the network. Most of the previous convolutional blocks are remained frozen except few at the last.

There are total $14, 473, 516$ number of trainable parameters in this model.

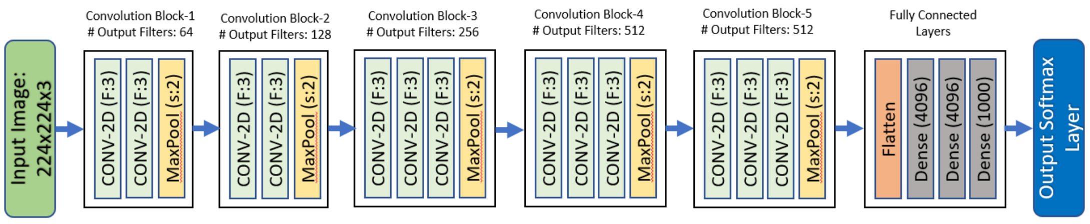

Input Image: 224x224x3
Convolution Block-1 # Output Filters: 64
CONV-2D (F:3)
CONV-2D (F:3)
MaxPool (s:2)
Convolution Block-2 # Output Filters: 128
Convolution Block-3 # Output Filters: 256
Convolution Block-4 # Output Filters: 512
Convolution Block-5 # Output Filters: 512
Fully Connected Layers
Flatten
Dense (4096)
Dense (4096)
Dense (1000)
Output Softmax Layer

Figure 6.4: Flattened VGG-16 architecture

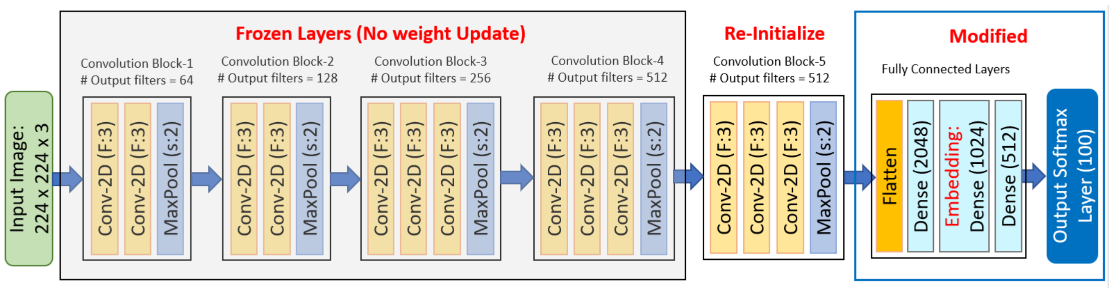

Frozen Layers (No weight Update)
Input Image: 224 x 224 x 3
Convolution Block-1 # Output filters = 64
Conv-2D (F:3)
Conv-2D (F:3)
MaxPool (s:2)
Convolution Block-2 # Output filters = 128
Convolution Block-3 # Output filters = 256
Convolution Block-4 # Output filters = 512
Re-Initialize
Convolution Block-5 # Output filters = 512
Modified
Fully Connected Layers
Flatten
Dense (2048)
Embedding: Dense (1024)
Dense (512)
Output Softmax Layer (100)

Figure 6.5: Fine-tuned VGG-16 architecture for feature extraction of Re-sampled ImageNet dataset

6.4 Training Schemes

Following table (6.1) illustrates the training scheme of the feature extracting CNN for various datasets.

Dataset	$\lvert\mathcal{C}\rvert$	n	m	Input Image Dimension	Encoded Dimension	Batch Size	Epochs
MNISTrb	10	12000	50000	28 × 28 × 1	120	40	5
Fashion MNIST	10	60000	10000	28 × 28 × 1	120	40	5
CIFAR-10	10	50000	10000	32 × 32 × 3	120	60	5
SVHN	10	73257	26032	32 × 32 × 3	120	60	5
Resampled ImageNet	100	64740	2000	224 × 224 × 3	1024	128	30

Table 6.1: Dataset-specific Feature Extracting CNN training scheme

All the networks were build in *python* programming language using *Keras* [39] package with *tensorflow* [40] back-end. The models were trained with NVIDIA TITAN-XP GPU.

Chapter 7

Experimental Results

This chapter is the summary of the results produced by experimental studies on different dataset. We have considered following metrics for evaluating our model's performance.

– Recall averaged over minority classes

– Recall averaged over all the classes, also known as Average Class Specific Accuracy (ACSA)

– F1 score averaged over all the classes

– GMean

– Accuracy

We have compared following methods for each dataset over varying class imblance ratio (ρ) = 10, 20, 40, and 100:

- **BL:** Baseline model
- **BL+CW:** Baseline model with class-weighting scheme
- **BL+ROS-RUS:** Baseline model along with Random Over-Sampling and Random Under-Sampling techniques.
- **SMOTE:** SMOTE algorithm applied in feature-space
- **DOS:** Deep Over-Sampling technique
- **ADOS:** Adaptive Dynamic Over-Sampling (Our proposed method)

7.1 Results on MNIST*rb* dataset

The results obtained for different methods in different imbalanced settings on MNIST*rb* dataset is shown below.

		Baseline			Baseline + Class Weight			Baseline + ROS - ROS			FS-SMOTE			DOS		**ADOS** *(Proposed Method)*		
Imb Ratio	Class	Re	F1	GMean	Re	F1	GMean	Re	F1	GMean	Re	F1	GMean	Re	F1	Re	F1	GMean
10	Maj	0.831	0.713	0.828	0.701	0.718	0.689	0.751	0.717	0.743	0.832	0.751	0.831	0.7	0.73	0.813	0.765	0.810
	Min	0.386	0.522	0.347	0.715	0.687	0.706	0.666	0.710	0.653	0.572	0.692	0.557	0.73	0.66	**0.730**	0.732	0.722
	Average	0.653	0.637	0.585	0.706	0.706	0.695	0.717	0.715	0.706	0.728	0.727	0.708	0.712	0.702	**0.780**	**0.752**	**0.774**
20	Maj	0.837	0.682	0.835	0.710	0.685	0.702	0.742	0.672	0.736	0.827	0.733	0.825	0.687	0.697	0.806	0.744	0.802
	Min	0.193	0.261	0.091	0.570	0.635	0.555	0.520	0.616	0.497	0.495	0.619	0.467	0.615	0.630	**0.626**	0.683	0.612
	Average	0.579	0.513	0.344	0.654	0.665	0.639	0.653	0.649	0.629	0.694	0.687	0.657	0.669	0.670	**0.734**	**0.719**	**0.720**
40	Maj	0.832	0.667	0.830	0.677	0.651	0.666	0.753	0.642	0.748	0.805	0.669	0.803	0.64	0.69	0.793	0.683	0.790
	Min	0.114	0.165	0.022	0.381	0.544	0.327	0.327	0.435	0.269	0.264	0.380	0.209	**0.532**	0.63	0.521	0.491	0.486
	Average	0.545	0.466	0.193	0.559	0.608	0.501	0.582	0.559	0.497	0.588	0.554	0.468	0.597	**0.666**	**0.684**	0.606	**0.650**
100	Maj	0.829	0.664	0.827	0.604	0.564	0.591	0.749	0.614	0.744	0.759	0.614	0.757	0.728	0.634	0.713	0.608	0.708
	Min	0.117	0.159	0.000	0.359	0.403	0.318	0.213	0.283	0.116	0.148	0.215	0.069	0.449	0.521	**0.469**	0.408	0.453
	Average	0.544	0.462	0.000	0.506	0.499	0.461	0.534	0.481	0.354	0.515	0.455	0.291	0.614	**0.537**	**0.615**	0.528	**0.592**

Table 7.1: Performance comparison of different methods on the imbalanced MNIST*rb* dataset of different imbalance ratio

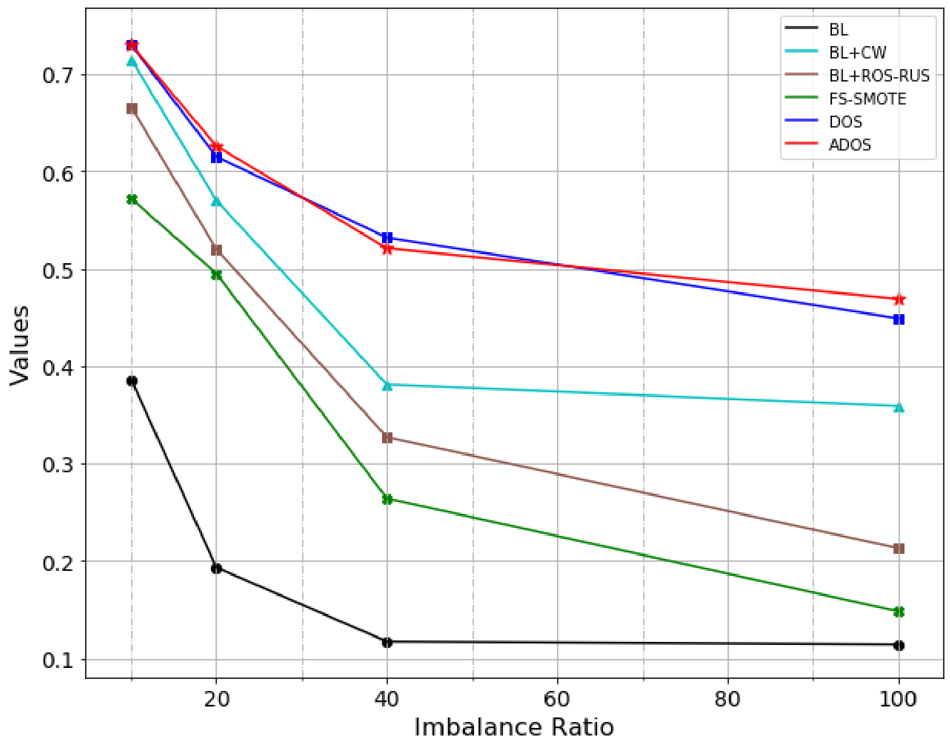


Figure 7.1: Average recall of minority classes on MNISTrb dataset

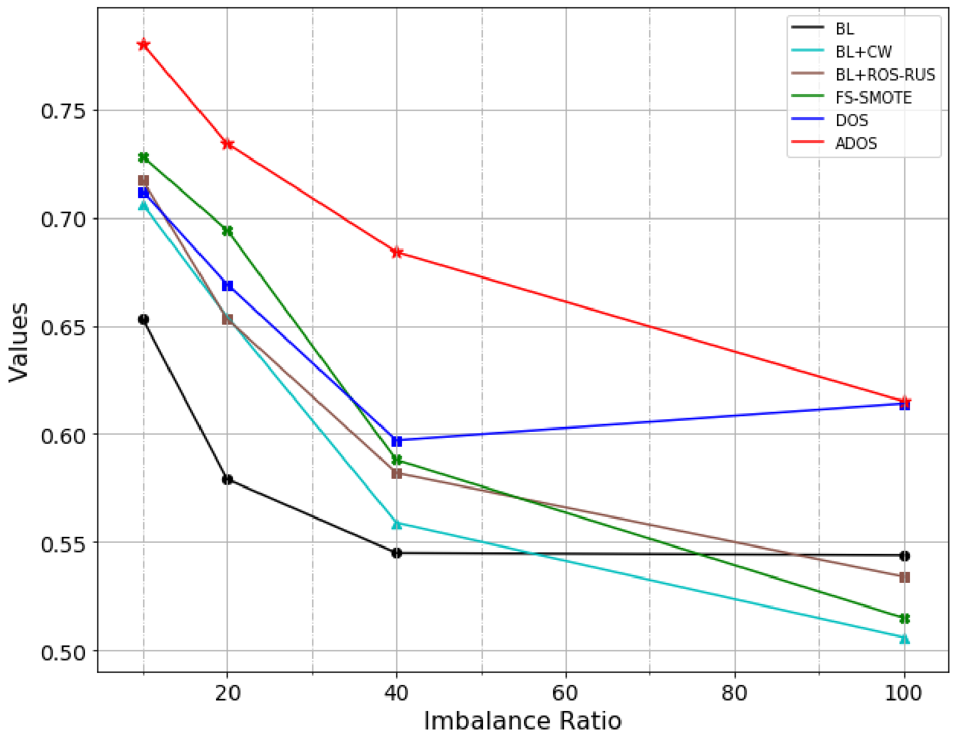


Figure 7.2: Average recall of all classes (ACSA) on MNISTrb dataset

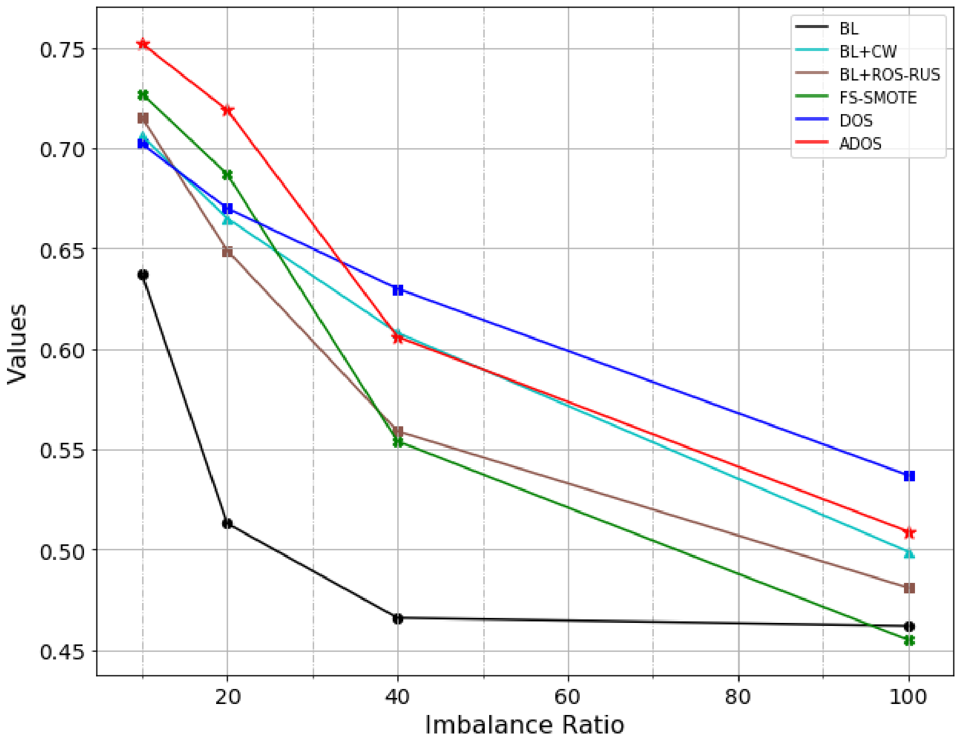


Figure 7.3: F1 score averaged over all classes on MNISTrb dataset

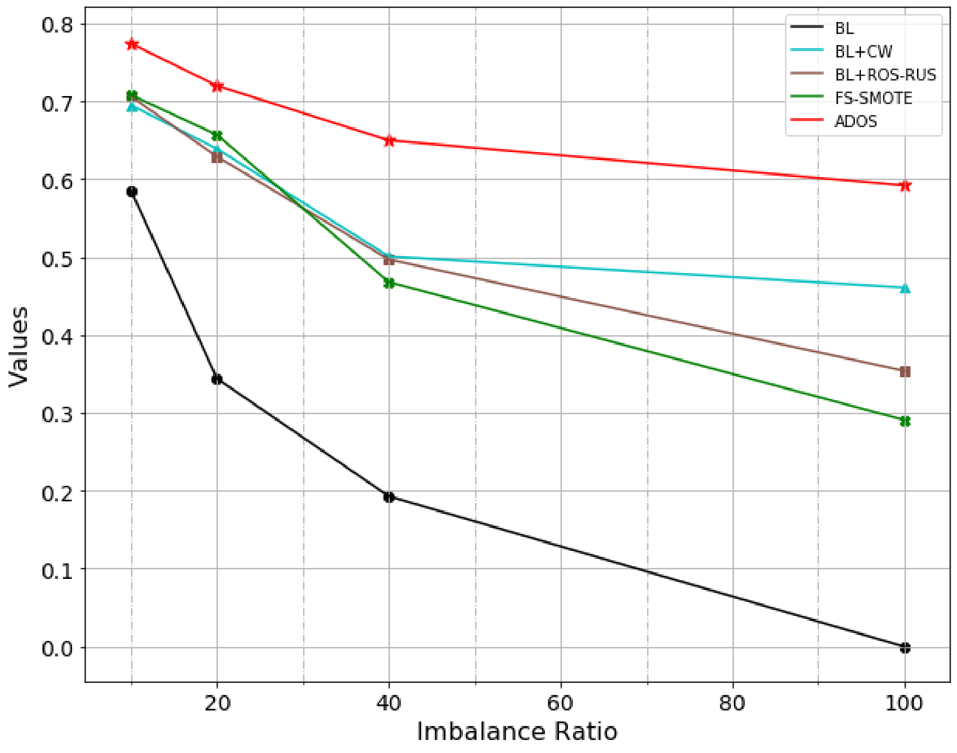


Figure 7.4: GMean on MNISTrb dataset

7.2 Results on Fashion-MNIST dataset

The results obtained for different methods in different imbalanced settings on Fashion-MNIST dataset is shown below.

		Baseline			Baseline + Class Weight			Baseline + ROS - ROS			FS-SMOTE			**ADOS** *(Proposed Method)*		
Imb Ratio	Class	Re	F1	GMean	Re	F1	GMean	Re	F1	GMean	Re	F1	GMean	Re	F1	GMean
10	Maj	0.914	0.838	0.912	0.844	0.837	0.832	0.858	0.847	0.848	0.919	0.876	0.917	0.908	0.880	0.905
	Min	0.620	0.719	0.554	0.832	0.842	0.825	**0.838**	0.853	0.831	0.796	0.864	0.788	0.836	0.880	0.830
	Average	0.796	0.790	0.747	0.839	0.839	0.829	0.850	0.850	0.841	0.870	0.871	0.863	**0.879**	**0.880**	**0.874**
20	Maj	0.927	0.833	0.925	0.851	0.836	0.835	0.871	0.836	0.864	0.917	0.856	0.915	0.903	0.860	0.900
	Min	0.556	0.668	0.482	0.768	0.841	0.756	0.773	0.831	0.764	0.714	0.807	0.691	**0.826**	0.841	0.819
	Average	0.778	0.767	0.713	0.818	0.838	0.802	0.832	0.834	0.822	0.835	0.836	0.817	**0.872**	**0.853**	**0.867**
40	Maj	0.912	0.811	0.910	0.821	0.794	0.798	0.892	0.816	0.888	0.926	0.842	0.924	0.910	0.858	0.907
	Min	0.515	0.634	0.428	0.740	0.800	0.763	0.635	0.744	0.609	0.633	0.750	0.601	**0.764**	0.817	0.718
	Average	0.753	0.740	0.673	0.798	0.796	0.783	0.789	0.787	0.764	0.809	0.805	0.778	**0.842**	**0.841**	**0.826**
100	Maj	0.923	0.793	0.922	0.747	0.737	0.683	0.891	0.809	0.888	0.925	0.822	0.924	0.908	0.827	0.906
	Min	0.330	0.419	0.155	0.689	0.752	0.737	0.567	0.689	0.539	0.513	0.643	0.460	**0.746**	0.728	0.670
	Average	0.686	0.644	0.451	0.747	0.743	0.705	0.761	0.761	0.727	0.760	0.750	0.699	**0.820**	**0.787**	**0.803**

Table 7.2: Performance comparison of different methods on the imbalanced Fashion-MNIST dataset of different imbalance ratio

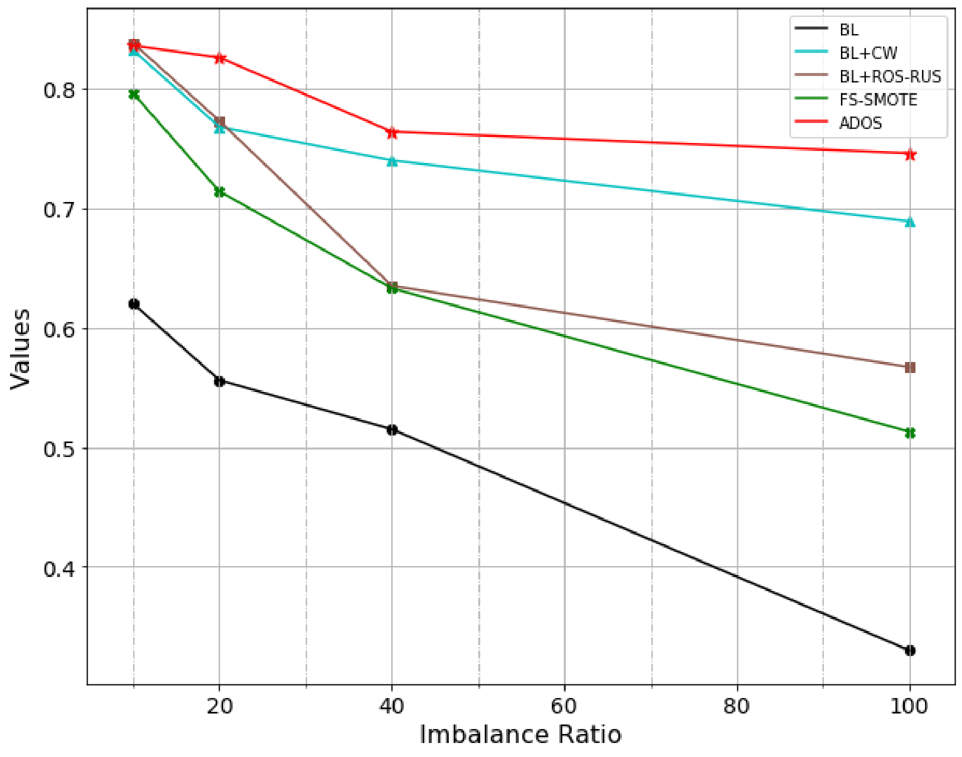


Figure 7.5: Average recall of minority classes on Fashion-MNIST dataset

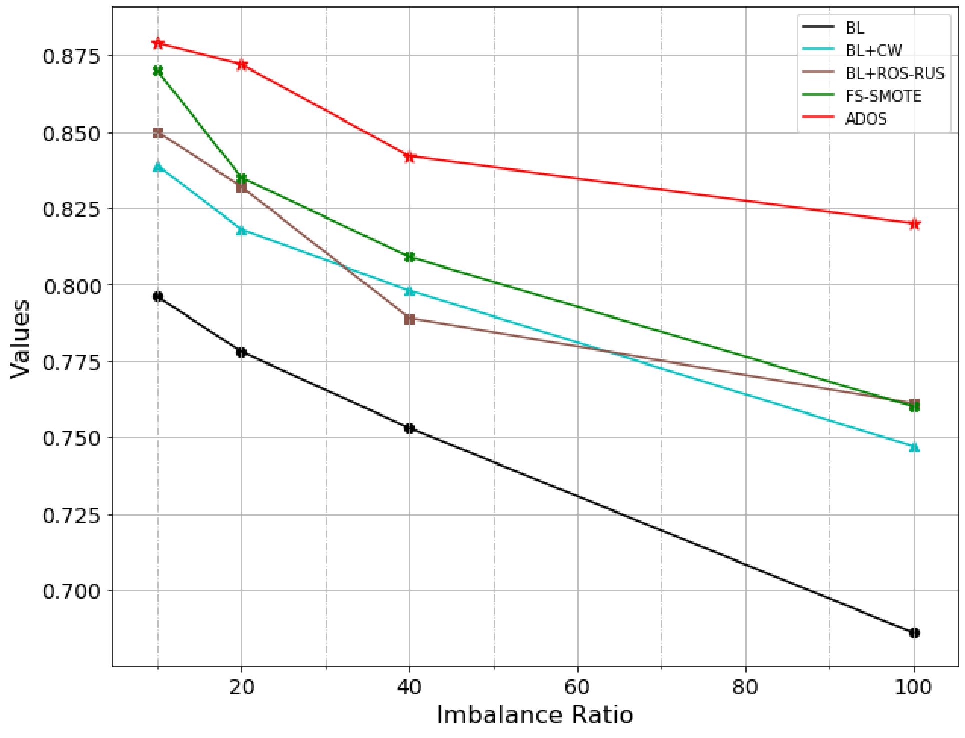


Figure 7.6: Average recall of all classes (ACSA) on Fashion-MNIST dataset

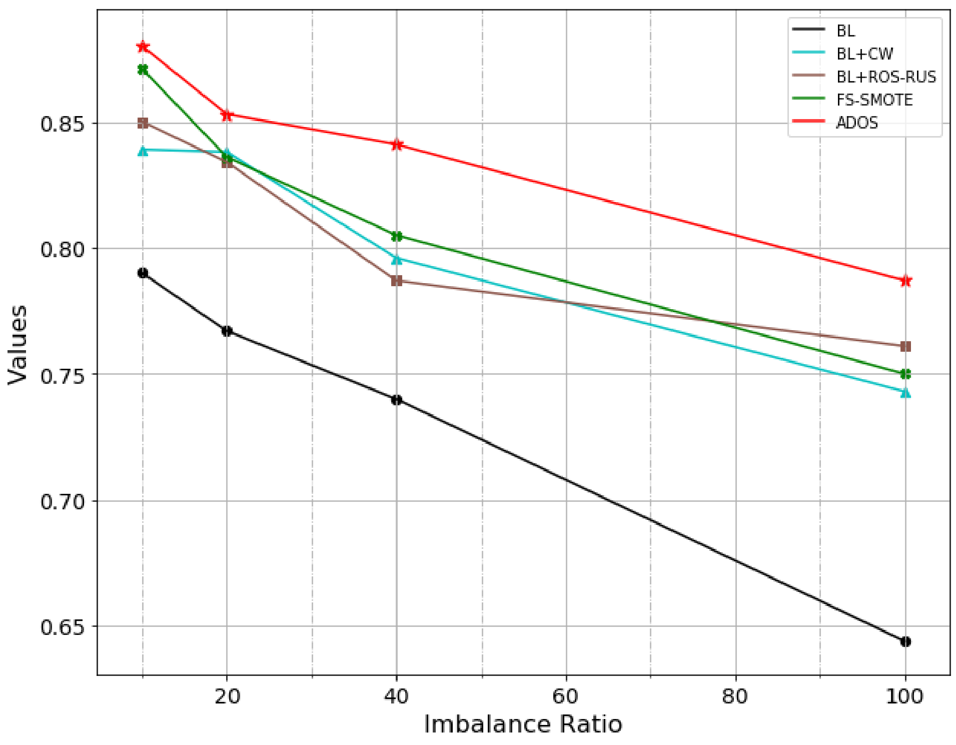


Figure 7.7: F1 score averaged over all classes on Fashion-MNIST dataset

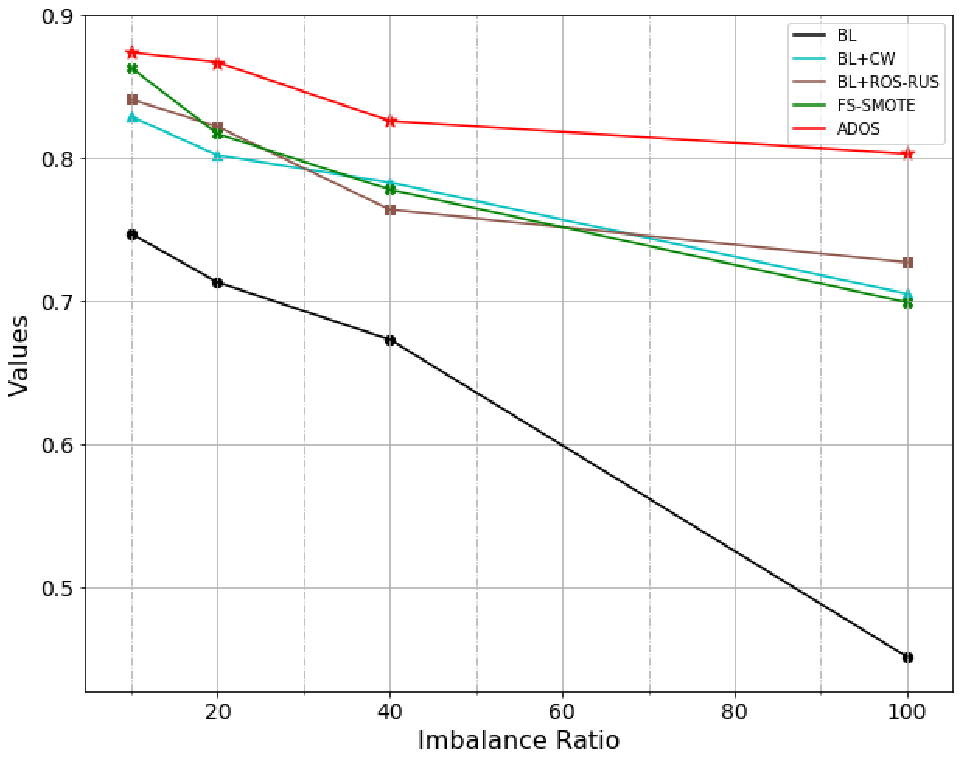


Figure 7.8: GMean on Fashion-MNIST dataset

7.3 Results on CIFAR-10 dataset

The results obtained for different methods in different imbalanced settings on CIFAR-10 dataset is shown below.

		Baseline			Baseline + Class Weight			Baseline + ROS - ROS			FS-SMOTE			**ADOS** *(Proposed Method)*		
Imb Ratio	Class	Re	F1	GMean	Re	F1	GMean	Re	F1	GMean	Re	F1	GMean	Re	F1	GMean
	Maj	0.748	0.619	0.731	0.605	0.574	0.588	0.669	0.603	0.658	0.762	0.654	0.758	0.731	0.671	0.725
10	Min	0.249	0.327	0.158	0.444	0.488	0.435	**0.426**	0.499	0.379	0.335	0.442	0.286	0.479	0.540	0.446
	Average	0.548	0.502	0.396	0.541	0.540	0.521	0.572	0.561	0.528	0.591	0.569	0.513	**0.630**	**0.619**	**0.597**
	Maj	0.747	0.592	0.742	0.572	0.564	0.559	0.688	0.566	0.679	0.749	0.623	0.747	0.736	0.637	0.732
20	Min	0.108	0.170	0.033	0.357	0.452	0.334	0.236	0.320	0.166	0.251	0.362	0.222	**0.475**	0.421	0.470
	Average	0.491	0.423	0.214	0.486	0.519	0.455	0.507	0.467	0.386	0.550	0.519	0.460	**0.632**	**0.551**	**0.613**
	Maj	0.754	0.595	0.750	0.518	0.484	0.457	0.702	0.566	0.696	0.744	0.608	0.742	0.723	0.613	0.718
40	Min	0.074	0.116	0.000	0.277	0.361	0.232	0.169	0.247	0.106	0.194	0.290	0.148	**0.433**	0.370	0.378
	Average	0.482	0.403	0.000	0.421	0.435	0.348	0.489	0.438	0.328	0.524	0.480	0.389	**0.607**	**0.516**	**0.556**
	Maj	0.745	0.569	0.739	0.427	0.385	0.363	0.703	0.548	0.697	0.702	0.544	0.697	0.667	0.562	0.662
100	Min	0.006	0.012	0.000	0.250	0.267	0.090	0.084	0.140	0.050	0.060	0.106	0.040	**0.303**	0.272	0.203
	Average	0.449	0.346	0.000	0.356	0.338	0.208	0.455	0.385	0.243	0.445	0.369	0.223	**0.521**	**0.446**	**0.412**

Table 7.3: Performance comparison of different methods on the imbalanced CIFAR-10 dataset of different imbalance ratio

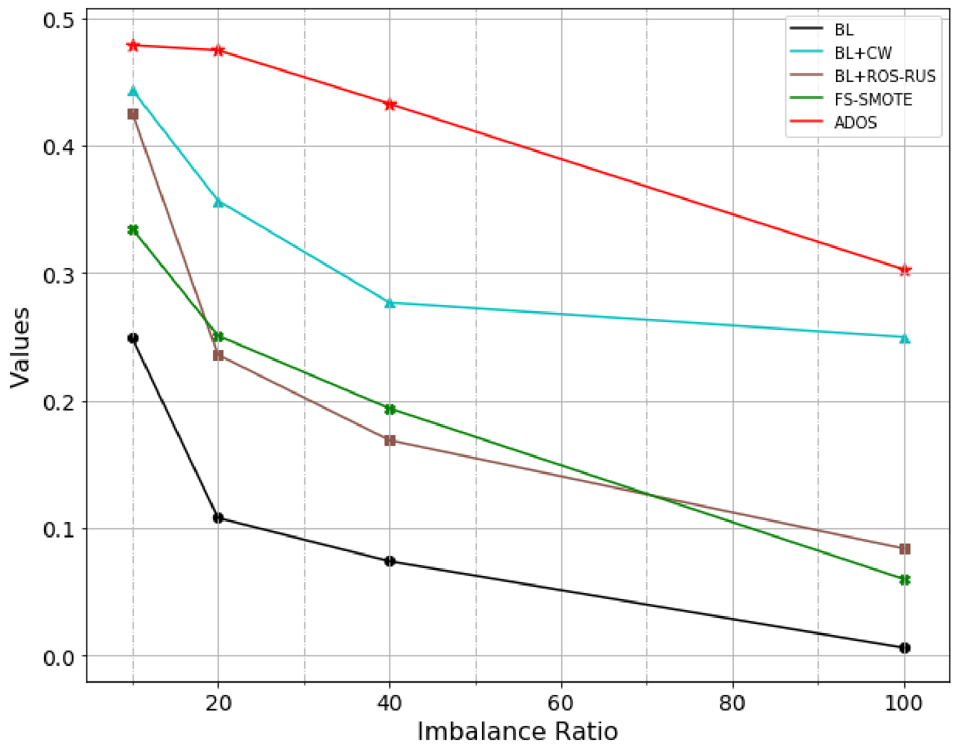


Figure 7.9: Average recall of minority classes on CIFAR-10 datset

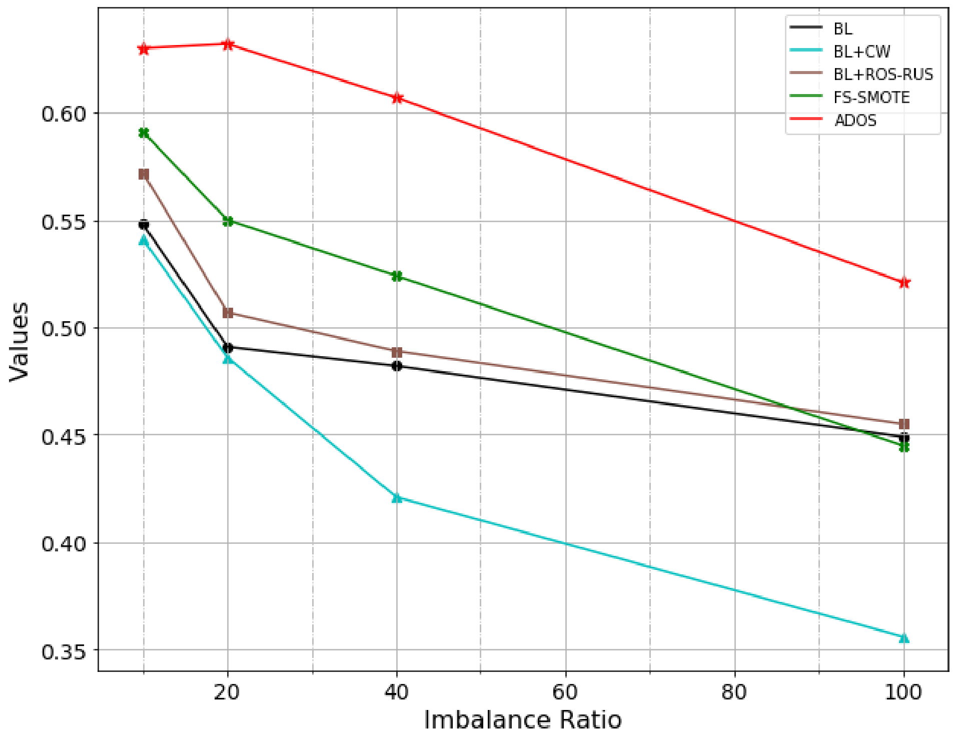


Figure 7.10: Average recall of all classes (ACSA) on CIFAR-10 datset

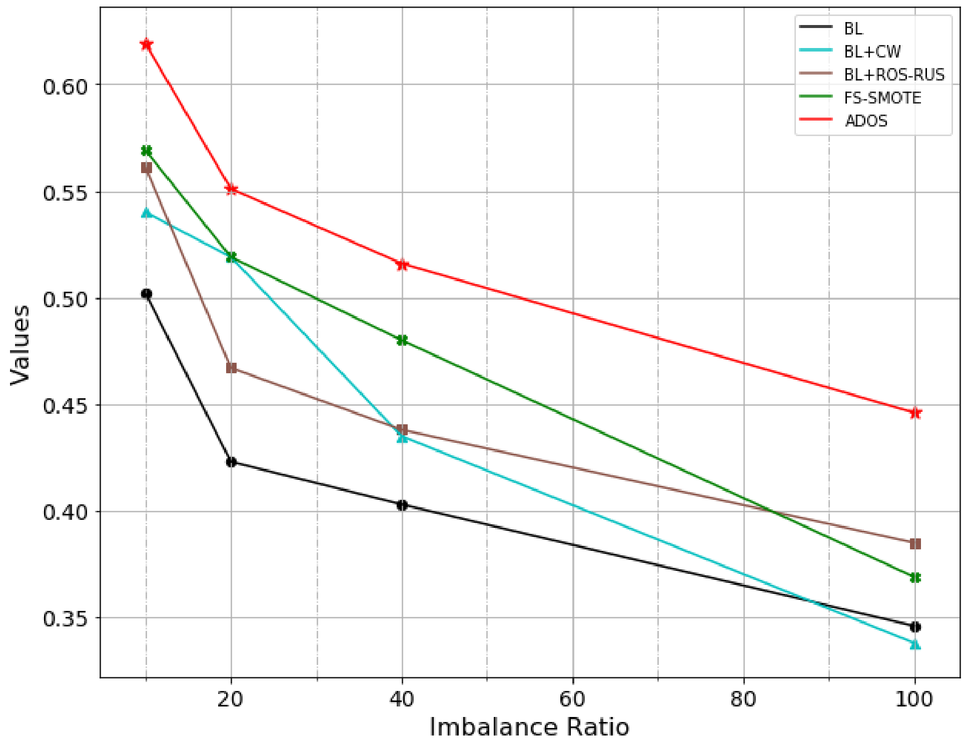


Figure 7.11: F1 score averaged over all classes on CIFAR-10 datset

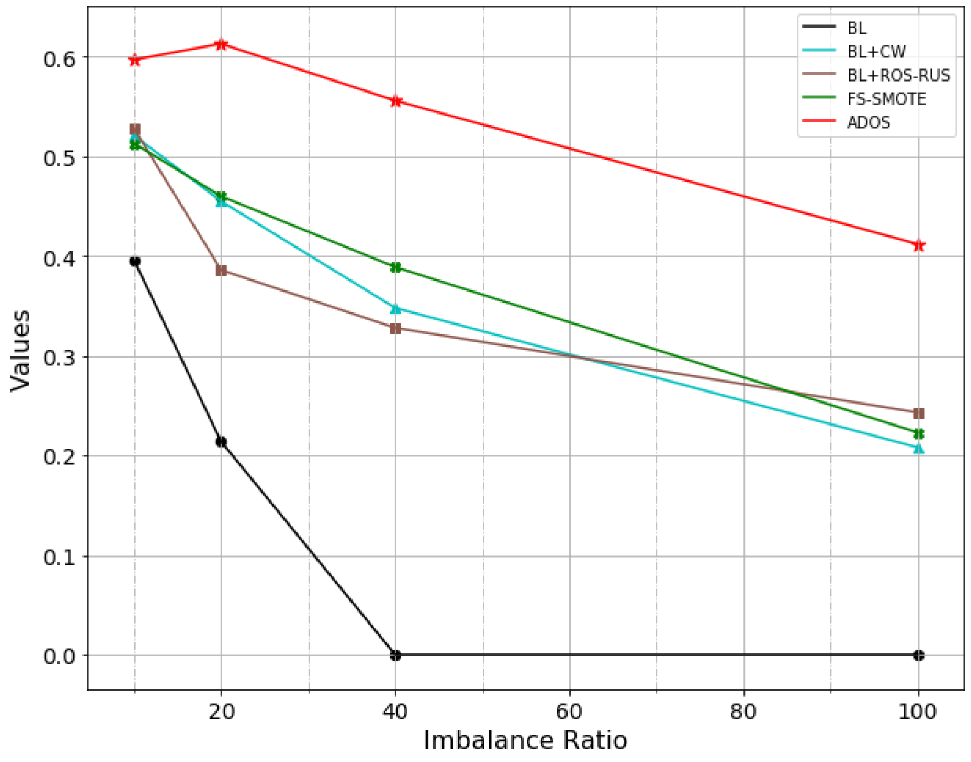


Figure 7.12: GMean on CIFAR-10 datset

7.4 Results on SVHN dataset

The results obtained for different methods in different imbalanced settings on SVHN dataset is shown below.

		Baseline			Baseline + Class Weight			Baseline + ROS - ROS			FS-SMOTE			DOS		ADOS *(Proposed Method)*		
Imb Ratio	Class	Re	F1	GMean	Re	F1	GMean	Re	F1	GMean	Re	F1	GMean	Re	F1	Re	F1	GMean
10	Maj	0.889	0.834	0.888	0.845	0.844	0.844	0.869	0.827	0.867	0.892	0.834	0.891	0.790	0.810	0.887	0.849	0.886
	Min	0.743	0.835	0.741	0.824	0.865	0.823	0.805	0.866	0.803	0.770	0.852	0.768	0.820	0.720	**0.865**	0.873	0.865
	Average	0.831	0.834	0.826	0.837	0.852	0.836	0.843	0.843	0.841	0.843	0.841	0.840	0.802	0.774	**0.878**	**0.859**	**0.877**
20	Maj	0.892	0.790	0.890	0.792	0.797	0.789	0.759	0.740	0.756	0.816	0.743	0.812	0.690	0.780	0.851	0.802	0.848
	Min	0.597	0.738	0.593	0.781	0.823	0.780	0.764	0.744	0.763	0.648	0.743	0.643	0.797	0.550	**0.840**	0.851	0.840
	Average	0.774	0.769	0.757	0.787	0.807	0.785	0.761	0.742	0.759	0.749	0.743	0.739	0.733	0.688	**0.847**	**0.821**	**0.844**
40	Maj	0.891	0.799	0.889	0.782	0.745	0.780	0.864	0.792	0.862	0.875	0.779	0.873	0.740	0.779	0.863	0.796	0.862
	Min	0.594	0.735	0.590	0.718	0.728	0.713	0.671	0.784	0.666	0.597	0.735	0.593	**0.727**	0.580	0.719	0.806	0.716
	Average	0.772	0.773	0.754	0.756	0.738	0.752	0.787	0.788	0.777	0.763	0.762	0.748	0.735	0.699	**0.805**	**0.800**	**0.800**
100	Maj	0.883	0.709	0.881	0.902	0.746	0.901	0.885	0.741	0.884	0.882	0.760	0.881	0.892	0.776	0.858	0.761	0.855
	Min	0.272	0.414	0.243	0.363	0.527	0.358	0.330	0.484	0.318	0.475	0.630	0.465	0.516	0.668	**0.550**	0.695	0.548
	Average	0.639	0.591	0.527	0.686	0.659	0.623	0.663	0.638	0.587	0.719	0.708	0.682	**0.742**	0.733	0.735	**0.735**	**0.716**

Table 7.4: Performance comparison of different methods on the imbalanced SVHN dataset of different imbalance ratio

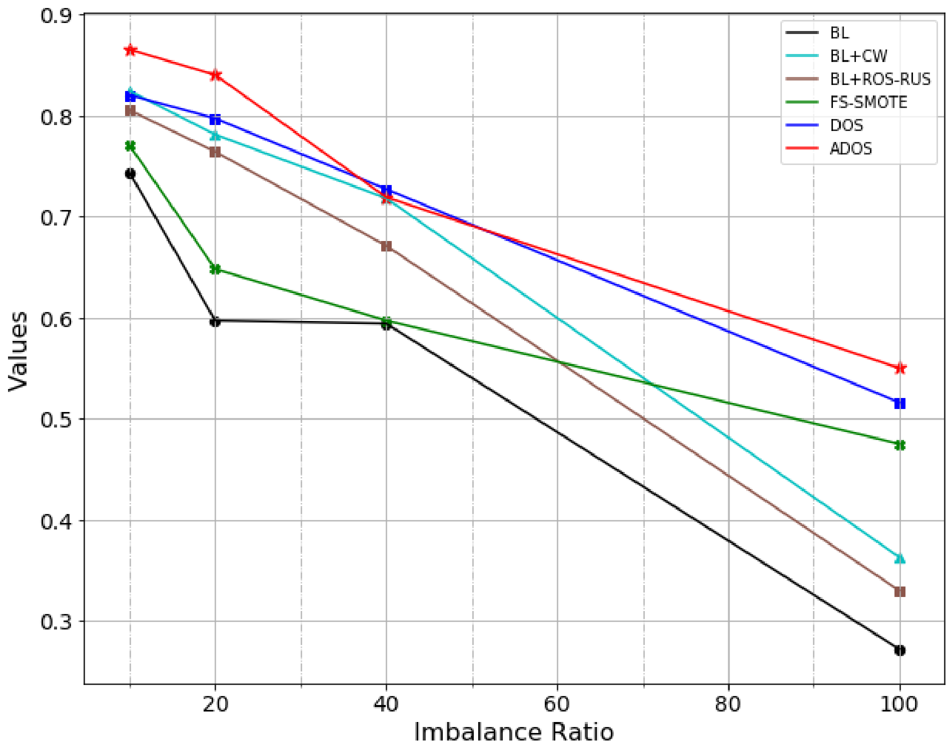


Figure 7.13: Average recall of minority classes on SVHN datset

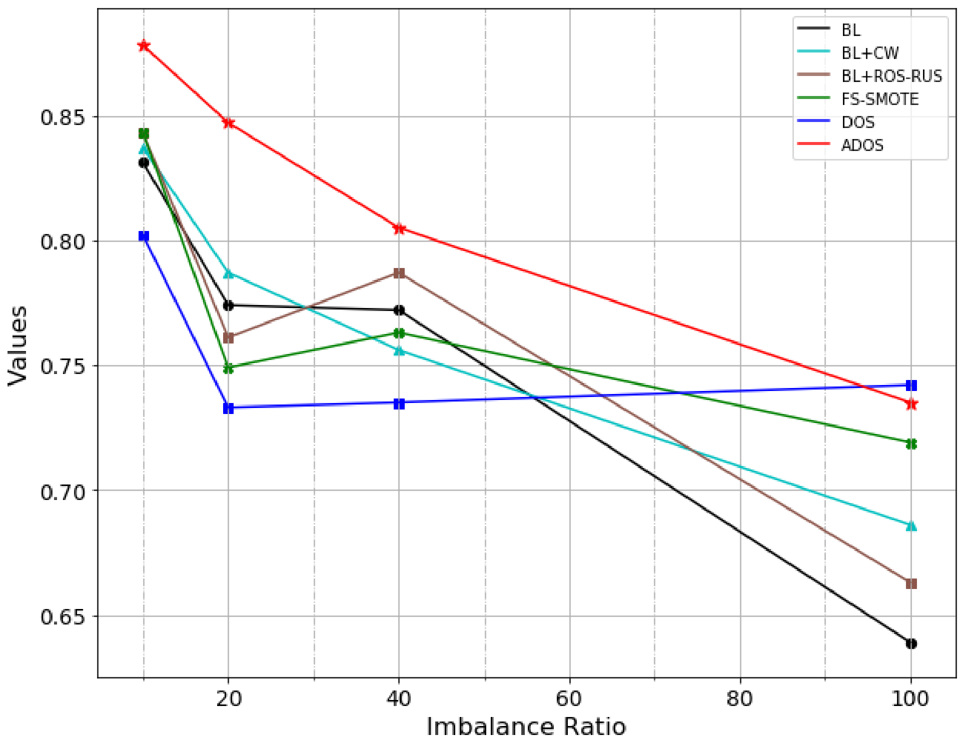


Figure 7.14: Average recall of all classes (ACSA) on SVHN datset

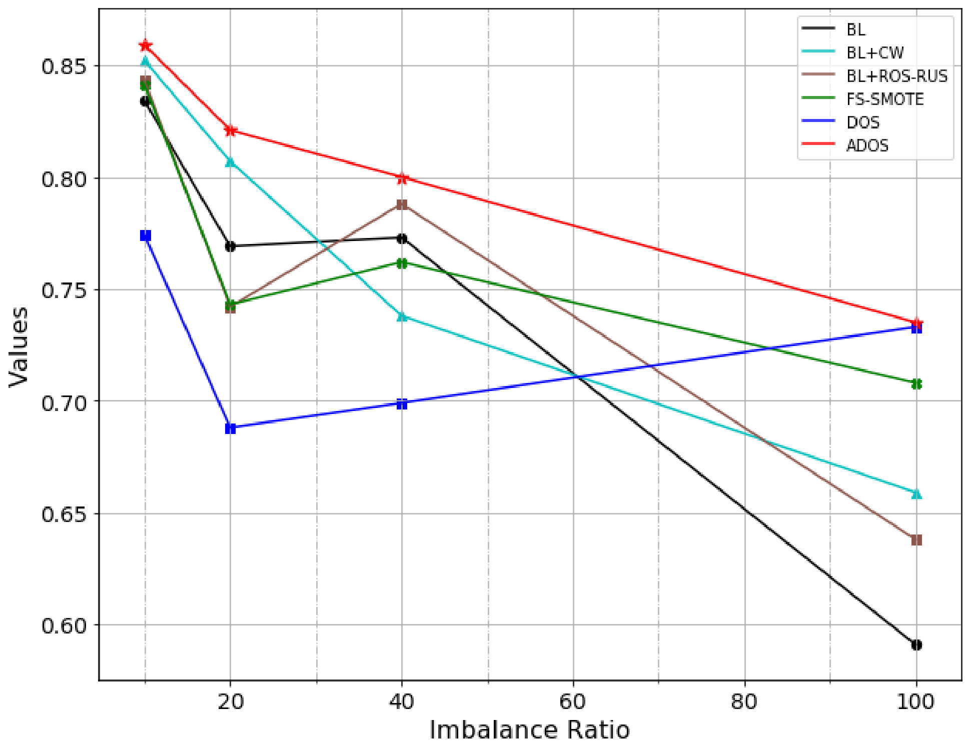


Figure 7.15: F1 score averaged over all classes on SVHN datset

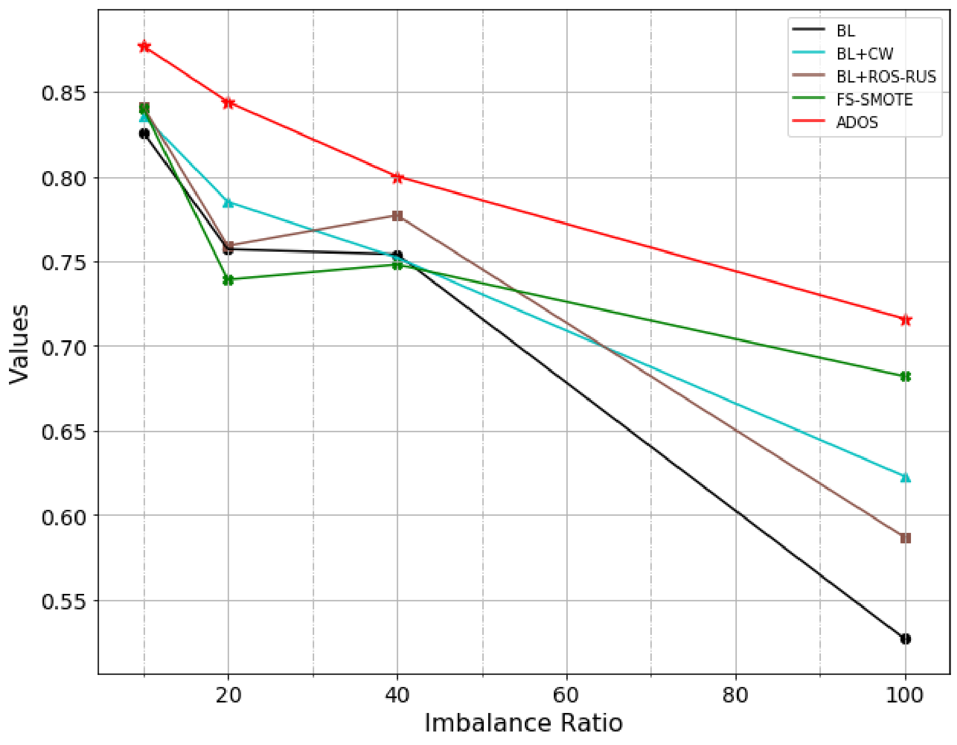


Figure 7.16: GMean on SVHN datset

7.5 Results on Re-sampled ImageNet dataset

The results obtained for Re-sampled ImageNet dataset is shown below.

	Baseline	Baseline + Class Weight	Baseline + ROS-RUS	FS-SMOTE	ADOS *(Proposed Method)*
min_pr	0.150	0.142	0.119	0.125	**0.220**
min_re	0.044	0.215	0.243	0.221	**0.274**
min_F1	0.059	0.158	0.160	0.138	**0.222**
maj_pr	0.148	0.141	0.153	0.155	**0.174**
maj_re	0.177	0.142	0.124	0.142	**0.232**
maj_F1	0.140	0.129	0.112	0.134	**0.193**
avg_pr	0.149	0.141	0.146	0.149	**0.184**
avg_re	0.149	0.157	0.156	0.159	**0.234**
avg_F1	0.123	0.135	0.122	0.134	**0.199**
test_acc	0.369	0.384	0.351	0.386	**0.402**
train_acc	0.434	0.435	0.446	0.581	**0.687**

Table 7.5: Performance comparison of different methods on the Resampled Imagenet dataset.

Graphically the performance on re-sampled imagenet dataset is following.

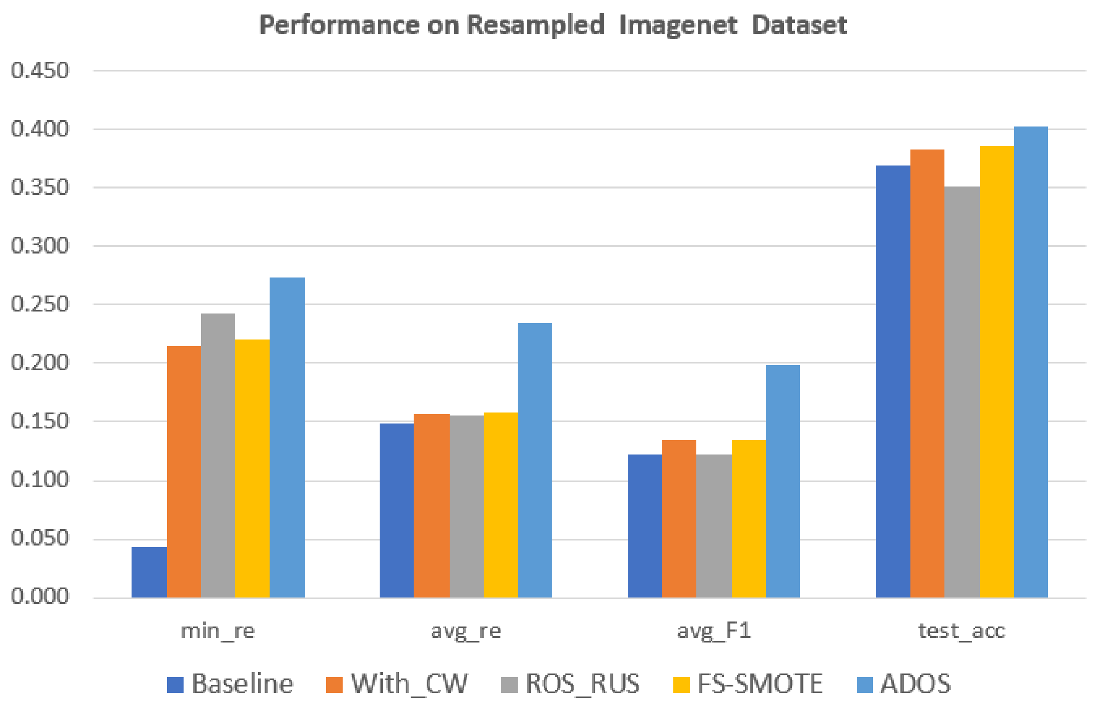


Figure 7.17: Performance comparison of different methods on resampled ImageNet dataset

Chapter 8

Conclusion and Scope of Future Work

8.1 Conclusion

From the results obtained from our experimental study it has been observed that:

- Our proposed method **A**daptive **D**ynamic **O**ver-**S**ampling (ADOS) Technique has performed significantly well as compared to other *state-of-the-art* technique.
- With the increased value of class imbalance ratio our proposed method (ADOS) has shown relatively smaller decline in the performance. Hence, our proposed algorithm is more resilient to class imbalance problem.

Hence we can say that our proposed method is a better technique to handle the class imbalance problem, as per the results produced by testing on different dataset.

8.2 Scope of Future Work

Despite its good performance, this method has few shortcomings. One of the shortcomings is that this method has an additional overhead of finding point specific number of nearest neighbours, thus it becomes computationally expensive. Extending our proposed methodology, following are list (not exhaustive) of research areas those can be explored:

- A comparative study can be made between supervised feature learning and unsupervised feature learning (using deep auto-encoders)
- Different kinds of weighting functions (like: radial basis function, wedge shaped function etc.) can be explored while creating synthetic neighbour for a datapoint.

- The idea of *Borderline*-SMOTE can be entangled with our proposed methodology to create samples near to the class boundary and thereby creating more discriminating power to the classifier.
- A study on the performance of our proposed method as compared to the existing ones for classifying general (not image) imbalanced datasets can be done.
- The idea of *manifold learning* can be intertwined with our proposed Over-Sampling algorithm to produce more locality sensitive synthetic samples.
- Some *clever* data structure can be proposed to improvise the overall time complexity of our proposed oversampling algorithm.

Bibliography

[1] Keinosuke Fukunaga. *Introduction to statistical pattern recognition*. Elsevier, 2013.

[2] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[3] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge & Data Engineering*, (9):1263–1284, 2008.

[4] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86 (11):2278–2324, 1998.

[5] Jianxiong Xiao, Krista A Ehinger, James Hays, Antonio Torralba, and Aude Oliva. Sun database: Exploring a large collection of scene categories. *International Journal of Computer Vision*, 119(1):3–22, 2016.

[6] Yu-Xiong Wang, Deva Ramanan, and Martial Hebert. Learning to model the tail. In *Advances in Neural Information Processing Systems*, pages 7029–7039, 2017.

[7] Chen Huang, Yining Li, Chen Change Loy, and Xiaoou Tang. Learning deep representation for imbalanced classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5375–5384, 2016.

[8] Yann LeCun Chris Burges, Corinna Cortes. The mnist database. URL `http://yann.lecun.com/exdb/mnist/`.

[9] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[12] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[13] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[14] Justin M Johnson and Taghi M Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):27, 2019.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[16] Keinosuke Fukunaga and L Hostetler. Optimization of k nearest neighbor density estimates. *IEEE Transactions on Information Theory*, 19(3):320–326, 1973.

[17] Rangachari Anand, Kishan G Mehrotra, Chilukuri K Mohan, and Sanjay Ranka. An improved algorithm for neural network classification of imbalanced training sets. *IEEE Transactions on Neural Networks*, 4(6):962–969, 1993.

[18] David Masko and Paulina Hensman. The impact of imbalanced training data for convolutional neural networks, 2015.

[19] Mateusz Buda, Atsuto Maki, and Maciej A Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:249–259, 2018.

[20] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[21] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-smote: a new over-sampling method in imbalanced data sets learning. In *International conference on intelligent computing*, pages 878–887. Springer, 2005.

[22] Chumphol Bunkhumpornpat, Krung Sinapiromsaran, and Chidchanok Lursinsap. Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 475–482. Springer, 2009.

[23] Salman H Khan, Munawar Hayat, Mohammed Bennamoun, Ferdous A Sohel, and Roberto Togneri. Cost-sensitive learning of deep feature representations from imbalanced data. *IEEE transactions on neural networks and learning systems*, 29(8):3573–3587, 2017.

[24] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[25] Haishuai Wang, Zhicheng Cui, Yixin Chen, Michael Avidan, Arbi Ben Abdallah, and Alexander Kronzer. Predicting hospital readmission via cost-sensitive deep learning. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 15(6):1968–1978, 2018.

[26] Chen Huang, Yining Li, Chen Change Loy, and Xiaoou Tang. Learning deep representation for imbalanced classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5375–5384, 2016.

[27] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pages 3730–3738, 2015.

[28] Shin Ando and Chun Yuan Huang. Deep over-sampling framework for classifying imbalanced data. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 770–785. Springer, 2017.

[29] Robert A Dunne. *A statistical approach to neural networks for pattern recognition*, volume 702. John Wiley & Sons, 2007.

[30] Thomas Wiatowski and Helmut Bölcskei. A mathematical theory of deep convolutional neural networks for feature extraction. *IEEE Transactions on Information Theory*, 64(3):1845–1866, 2017.

[31] Haohan Wang and Bhiksha Raj. On the origin of deep learning. *arXiv preprint arXiv:1702.07800*, 2017.

[32] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[33] Sankha Subhra Mullick, Shounak Datta, and Swagatam Das. Adaptive learning-based k-nearest neighbor classifiers with resilience to class imbalance. *IEEE transactions on neural networks and learning systems*, (99):1–13, 2018.

[34] Pratip Bhattacharyya and Bikas K Chakrabarti. The mean distance to the nth neighbour in a uniform distribution of random points: an application of probability theory. *European Journal of Physics*, 29(3):639, 2008.

[35] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on Machine learning*, pages 473–480. ACM, 2007.

[36] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[37] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[38] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[39] François Chollet et al. Keras, 2015.

[40] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.